

Vorlesung Informatik III – Theoretische Informatik

Formale Sprachen, Berechenbarkeit & Komplexitätstheorie

Prof. Dr. Peter Thiemann

Basierend auf einem Mitschrieb im WS 2015/16 von Ralph Lesch[†]

WS 2016/17

Zuletzt aktualisiert: 2020-03-04

[†]ralph.lesch@neptun.uni-freiburg.de

Inhaltsverzeichnis

1	Vorspann: Sprachen	3
2	Reguläre Sprachen und endliche Automaten	7
2.1	Endliche Automaten	7
2.2	Minimierung endlicher Automaten	12
2.2.1	Exkurs: Äquivalenzrelationen	13
2.3	nichtdeterministischer endlicher Automat (NEA)	21
2.4	Pumping Lemma (PL) für reguläre Sprachen	25
2.5	ε -Übergänge	26
2.6	Abschlusseigenschaften	32
2.7	Reguläre Ausdrücke	33
2.7.1	Motivation	33
3	Grammatiken und kontextfreie Sprachen	40
3.1	Kontextfreie Sprachen	43
3.2	Die Chomsky-Normalform für kontextfreie Sprachen	45
3.3	Wortproblem und Leerheitsproblem für kontextfreie Sprachen	53
3.4	Einschub: Typ-3-Sprachen	56
4	Kellerautomaten (PDA)	58
5	Kontextfreie Sprachen – Teil 2	67
5.1	Das Pumping Lemma für kontextfreie Sprachen	67
5.2	Abschlusseigenschaften für kontextfreie Sprachen	69
5.3	Deterministisch kontextfreie Sprachen	70
6	Turingmaschinen	73
6.1	Turingmaschine (informell)	73
6.2	Formalisierung der Turing-Maschine	75
6.3	Turing-Maschine (TM)-Programmierung mit Hilfe von Flussdiagrammen	78
6.4	Varianten von TMs	81
6.4.1	Mehrspurmaschinen	81
6.4.2	Mehrbandmaschinen	83
6.5	Universelle Turingmaschine	84
6.5.1	Codierung von Turingmaschinen	85
6.5.2	Arbeitsweise der universellen Turingmaschine	87
7	Berechenbarkeit	90
7.1	Church-Turing These	90
7.2	Nicht berechenbare Funktionen	90

7.3 Sprachakzeptanz und Berechenbarkeit	96
7.4 Das Halteproblem	97
7.5 Nichtdeterministische Turingmaschinen	104
8 Komplexitätstheorie	107
9 Einordnung von Sprachen in Chomsky-Hierarchie und Abschlusseigenschaften	122
Liste der Definitionen	128
Liste der Sätze	128
Abbildungsverzeichnis	128
Abkürzungsverzeichnis	128

1 Vorspann: Sprachen

Vorlesung:
19.10.2018

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



Zeichen sind hier beliebige abstrakte Symbole.

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

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

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

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



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

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

- **rambo** (Länge 5)
- **eis, ies** (beide Länge 3 aber ungleich)

[†]Dies ist eine Konvention 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.

^{††}Dieser Satz dient dazu, dass die Autoren dieses Skripts nicht jede Definition mit “Sei Σ ein Alphabet...” beginnen müssen.

^{†††}Wir schreiben also z.B. **einhorn** statt **e,i,n,h,o,r,n**.

^{††††}Eine analoge Konvention, die sie aus der Schule kennen: Verwende immer π als Symbol für die Kreiszahl.

- ε (Länge 0)

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

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

Bsp.:

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

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 $\text{rambo}\varepsilon$ 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, $\cdot : \Sigma^* \times \mathbb{N} \rightarrow \Sigma^*$, ist induktiv definiert durch

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

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

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

Bsp.:

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

Bsp.:

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

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

Def. 1.7 (Kleene-Abschluss, Kleene-Stern, Kleene-Plus): Gegeben eine Sprache $U \subseteq \Sigma^*$, definieren wir den *Kleene-Abschluss* U^* wie folgt.

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

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

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

 $\oplus$

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

2 Reguläre Sprachen und endliche Automaten

Vorlesung:
20.10.17

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

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

Ein weiteres Beispiel ist die folgende Zeile.

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

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

Die obige Darstellung der unendlichen Mengen L_{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*)

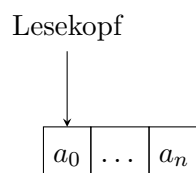


Abb. 1: Endliches Band

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

Akzeptierende Zustände $F \subseteq Q$

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

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

Bsp. 2.1: Aufgabe:

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



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

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

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

† 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

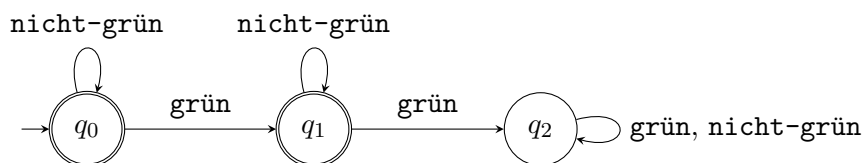
Def. 2.1 (DEA): Ein *deterministischer endlicher Automat* (DEA), (DFA $\hat{=}$ deterministic finite automaton) ist ein 5-Tupel

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

Dabei ist

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

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_{Macaron} aus Beispiel 2.1:



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

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

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

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

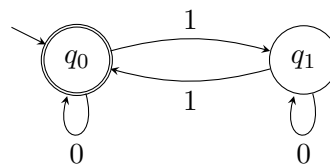
$$L(\mathcal{A}) = \{w \in \Sigma^* \mid \tilde{\delta}(q^{\text{init}}, w) \in F\}.$$

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

Bsp. 2.2: Ein möglicher DEA für die Sprache

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

aus der Einleitung dieses Kapitels hat die folgende graphische Repräsentation.



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

Vorlesung:
24.10.18

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

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

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

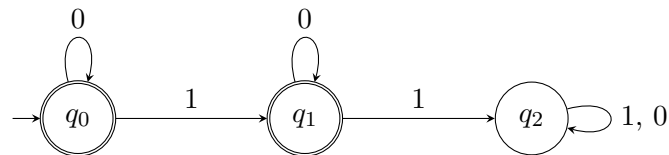
Def. 2.4 (Produktautomat für Schnitt): Seien $A_1 = (\Sigma, Q_1, \delta_1, q_1^{\text{init}}, F_1)$ und $A_2 = (\Sigma, Q_2, \delta_2, q_2^{\text{init}}, F_2)$ DEAs. Wir definieren den *Produktautomaten für Schnitt* als den DEA $A_\cap = (\Sigma, Q_\cap, \delta_\cap, q_\cap^{\text{init}}, F_\cap)$ mit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

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

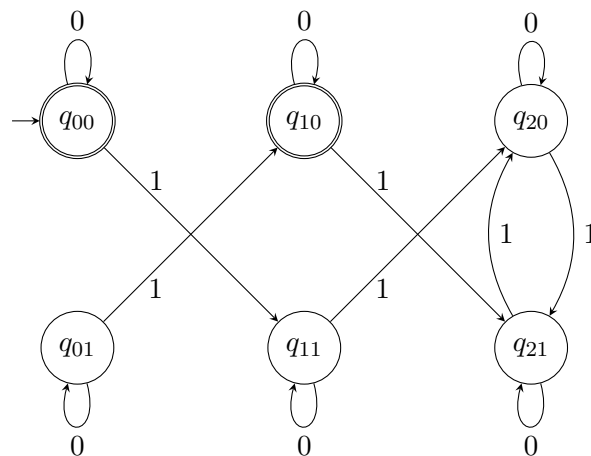
$\oplus$

Bemerkung: In der Vorlesung vom 24.10.2018 wurde zunächst das Beispiel und dann erst die Konstruktion präsentiert.

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



Bemerkung: In der Vorlesung vom 24.10.2018 wurde das gleiche Beispiel mit vertauschten Operanden präsentiert. (Die resultierende Graphstruktur war also identisch aber die Beschriftung der Knoten war eine andere.)

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

BEWEIS: [†] Seien $A_1 = (\Sigma, Q_1, \delta_1, q_1^{\text{init}}, F_1)$ und $A_2 = (\Sigma, Q_2, \delta_2, q_2^{\text{init}}, F_2)$ DEAs und $A_\cap = (\Sigma, Q_\cap, \delta_\cap, q_\cap^{\text{init}}, F_\cap)$ der zugehörige Produktautomat für Schnitt. Wir zeigen zunächst via

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

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.

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

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.

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

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

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

□

2.2 Minimierung endlicher Automaten

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

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

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

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

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:

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

Bemerkung: Der folgende Exkurs war Thema von Vorbereitungsblatt 01 und wurde in der der Vorlesung vom 24.10.2018 wurde nur kurz besprochen.

2.2.1 Exkurs: Äquivalenzrelationen

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

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

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

[†]<http://home.mathematik.uni-freiburg.de/junker/ss17/matheII.html>

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

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

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

Def. 2.7: 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 . $\oplus$

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*.

Vorlesung:
26.10.18

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). $\square$

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

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

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

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

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

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

BEWEIS:

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

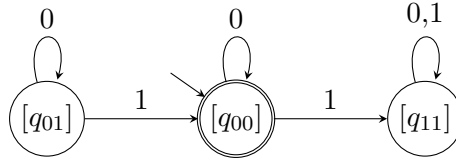
Wir formalisieren das „Verschmelzen“ von Zuständen wie folgt.

Def. 2.8: Der Äquivalenzklassenautomat $\mathcal{A}_{\equiv} = (Q_{\equiv}, \Sigma, \delta_{\equiv}, q_{\equiv}^{\text{init}}, F_{\equiv})$ zu einem DEA $\mathcal{A} = (Q, \Sigma, \delta, q^{\text{init}}, F)$ ist bestimmt durch:

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

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

[†]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.



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

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

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

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

□

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

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

Def. 2.9: Eine Äquivalenzrelation $R \subseteq \Sigma^* \times \Sigma^*$ heißt *rechtskongruent*, falls

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

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

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

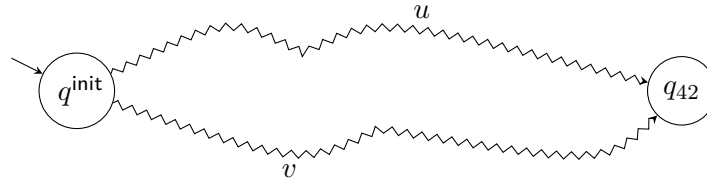


Abb. 2: Schematische Darstellung eines DEA, mit $\tilde{\delta}(q^{\text{init}}, u) = \tilde{\delta}(q^{\text{init}}, v) = q_{42}$

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

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

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

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

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

Def. 2.10: Für einen DEA $\mathcal{A}$ definiere

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

Beobachtung 1 R_A ist eine Äquivalenzrelation.

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

Beobachtung 2 R_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_A ist also die Anzahl der erreichbaren Zustände.

Für den DEA aus Bsp. 2.3 hat R_A die folgenden Äquivalenzklassen.

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

Def. 2.11: Für eine Sprache $L \subseteq \Sigma^*$ ist die *Nerode-Relation* wie folgt definiert.

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

Vorlesung:
31.10.18

Beobachtung 1 Die Nerode-Relation ist eine Äquivalenzrelation.

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

Beobachtung 2 Die Nerode-Relation ist rechtskongruent.

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

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

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

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

Bsp. 2.8: 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. 2.9: 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. 2.10: Sei $\Sigma = \{0, 1\}$. Die Sprache $L_{\text{centered}} = \{0^n 10^n \mid n \in \mathbb{N}\}$ hat bezüglich der Nerode-Relation die folgende Menge von Äquivalenzklassen:

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

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

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

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

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

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

Satz 2.5 (Myhill und Nerode): Für alle Sprachen $L \subseteq \Sigma^*$ sind die folgenden Aussagen äquivalent:

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

BEWEIS: Wir beweisen die paarweise Äquivalenz in drei Schritten:

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

[†]Dieses Beispiel wurde in der Vorlesung übersprungen.

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

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

Nun sind $\{w \mid \tilde{\delta}(q^{\text{init}}, w) = q\}$ genau die Äquivalenzklassen der Relation $R_{\mathcal{A}}$ aus Def. 2.10, einer rechtskongruenten Äquivalenzrelation. Der Index ist die Anzahl der erreichbaren Zustände und somit endlich: $\text{Index}(R_{\mathcal{A}}) \leq |Q| < \infty$.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.3 Nichtdeterministischer endlicher Automat (NEA)

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

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

Naiver Lösungsversuch:

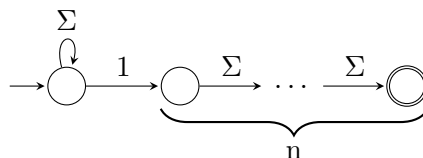


Abb. 3: Nichtdeterministischer Automat für L_n

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

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

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

Def. 2.12 (NEA): Ein *nichtdeterministischer endlicher Automat (NEA)*, ($\text{NFA} \triangleq \text{non-deterministic finite automaton}$) ist ein 5-Tupel

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

Dabei ist

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

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

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

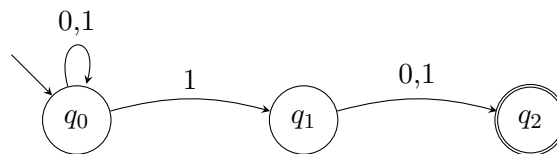
Def. 2.13: Wir nennen eine Folge von Zuständen $q_0 q_1 \dots q_n$ einen *Lauf von $\mathcal{N}$ über $w = a_1 \dots a_n$* , falls $q_i \in \delta(q_{i-1}, a_i)$ für alle i mit $1 \leq i \leq n$. Wir nennen einen Lauf *initial*, falls $q_0 = q^{\text{init}}$. Wir nennen einen Lauf *akzeptierend*, falls $q_n \in F$. $\oplus$

Def. 2.14: 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. $L(\mathcal{N}) = \{w \in \Sigma^* \mid \exists \text{ initialer, akzeptierender Lauf von } \mathcal{N} \text{ über } w\}$. $\oplus$

Bsp. 2.11: Der NEA für die Sprache

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

hat die folgende graphische Repräsentation.



Bemerkung: Die Frage, ob ein gegebenes Wort w akzeptiert wird (das „Wortproblem“), lässt sich für NEAs nicht mehr so leicht beantworten wie wir es von DEAs gewohnt sind. Ein sinnvolles Vorgehen scheint, jeden initialen Lauf zu betrachten, doch z.B. für das Wort 11 hat obiger NEA bereits drei verschiedene initiale Läufe: $q_0q_0q_0$, $q_0q_0q_1$ und $q_0q_1q_2$.

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

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

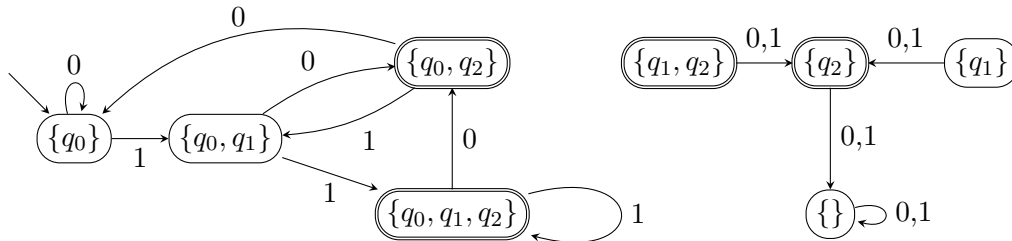
Satz 2.6 (Rabin und Scott): Zu jedem NEA $\mathcal{N}$ mit n Zuständen gibt es einen DEA $\mathcal{A}_{\mathcal{P}}$ mit 2^n Zuständen, sodass $L(\mathcal{A}_{\mathcal{P}}) = L(\mathcal{N})$ gilt.

Zur Vorbereitung des Beweises machen wir zunächst die folgende Definition.

Def. 2.15 (Potenzmengenautomat): Für einen gegebenen NEA $\mathcal{N} = (\Sigma, Q, \delta, q^{\text{init}}, F)$ ist der Potenzmengenautomat $\mathcal{A}_{\mathcal{P}}$ wie folgt definiert.

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

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



Die vier Zustände auf der rechten Seite sind nicht erreichbar.

Vorlesung:
7.11.18

BEWEIS (von Satz 2.6): Zeige $L(\mathcal{AP}) = L(\mathcal{N})$. Dafür beweisen wir zunächst via Induktion über die Länge von w die folgende Eigenschaft:

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

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

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

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

$$q \in \tilde{\delta}_{\mathcal{P}}(p, w) \Leftrightarrow q \in \tilde{\delta}_{\mathcal{P}}(\delta_{\mathcal{P}}(p, a), w')$$

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

$$\Leftrightarrow \exists \underbrace{q_1, q_2, \dots, q_{n+1}}_{\text{Lauf}} \in Q, \text{ sodass } n = |w'|, \exists q_0 \in p : q_1 \in \delta(q_0, a) \text{ und } q_{n+1} = q$$

$$\Leftrightarrow \exists \underbrace{q_0, q_1, q_2, \dots, q_{n+1}}_{\text{Lauf}} \in Q, \text{ sodass } n + 1 = |w|, q_0 \in p \text{ und } q_{n+1} = q$$

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

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

□

Bemerkung: Es gelten also die folgenden Äquivalenzen.

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

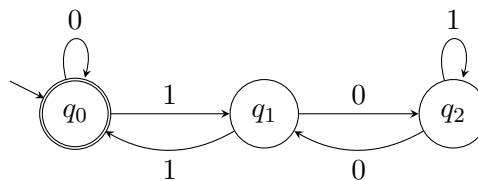
Bemerkung: NEAs sind eine exponentiell kompaktere Repräsentation von regulären Sprachen im folgenden Sinne:

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

2.4 Pumping Lemma (PL) für reguläre Sprachen

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

Bsp. 2.13: Betrachte den folgenden DEA, der die Sprache der Binärcodierung von durch drei teilbaren Zahlen akzeptiert: $L = \{w \in \{0,1\}^* \mid \text{bin}(w) \equiv_3 0\}$



Beobachtungen:

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

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

Lemma 2.7 (Pumping Lemma): Sei L eine reguläre Sprache. Dann gilt:

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

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

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

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

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

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

Bsp. 2.14: Die Sprache $L_{\text{centered}} = \{0^n 10^n \mid n \in \mathbb{N}\}$ ist nicht regulär.

Vorlesung:
9.11.18

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

Sei n die Konstante aus dem PL. Wähle $z = 0^n 10^n$. (Gültige Wahl, da $|z| = 2n + 1 \geq n$)

Laut PL existieren u, v, w , sodass $z = uvw$ mit $|v| \geq 1, |uv| \leq n$ und $\forall i \in \mathbb{N} : uv^i w \in L$.

Nach Wahl von z gilt nun

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

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

Zur Illustration:

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

2.5 ε -Übergänge

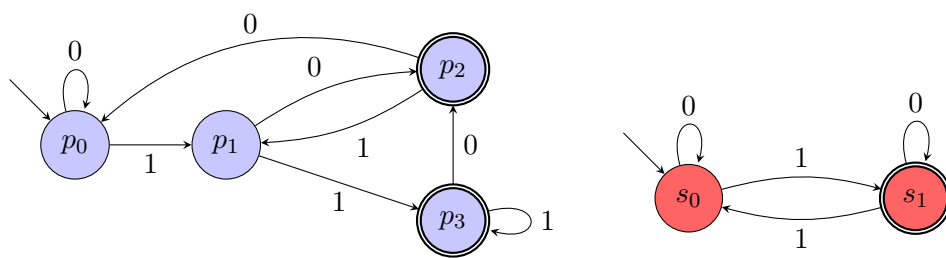
In diesem Unterkapitel führen wir mit dem ε -NEA ein weiteres Automatenmodell ein. Wir wollen ε -NEAs zunächst durch die folgende Fragestellung und anschließende Diskussion motivieren.

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

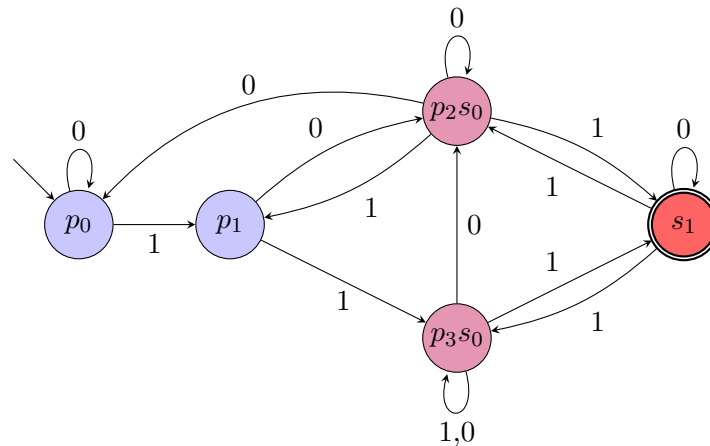
Idee: Gegeben DEA $\mathcal{A}_1$ mit $L(\mathcal{A}_1) = L_1$ und DEA $\mathcal{A}_2$ mit $L(\mathcal{A}_2) = L_2$, konstruiere NEA für $L_1 \cdot L_2$ durch „Hintereinanderschalten“ von $\mathcal{A}_1$ und $\mathcal{A}_2$; immer wenn wir in einem akzeptierenden Zustand von $\mathcal{A}_1$ sind, erlauben 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. 2.15: Links: DEA $\mathcal{A}_1$, der Automat aus Bsp. 2.11 eingeschränkt auf die erreichbaren Zustände. Rechts: DEA $\mathcal{A}_2$ mit der Sprache $\{w \in \{0,1\}^* \mid \text{Anzahl 1 in } w \text{ ungerade}\}$.



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



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

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

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

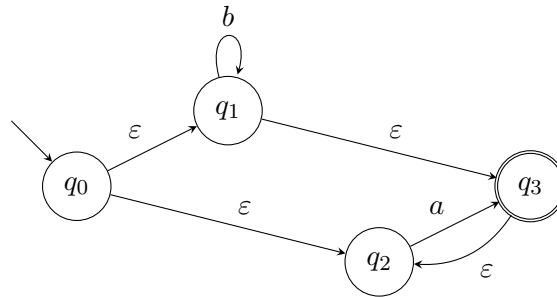
Def. 2.16 (ε -NEA): Ein *nichtdeterministischer endlicher Automat mit ε -Übergängen* ist ein 5-Tupel

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

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

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

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



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

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

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

Def. 2.17: Der ε -Abschluss $\text{ecl}_{\mathcal{B}} : Q \rightarrow \mathcal{P}(Q)$ ist die kleinste Abbildung, die für alle $q, q', q'' \in Q$ die folgenden Eigenschaften erfüllt:

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

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

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

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

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

Def. 2.18: Die *Erreichbarkeitsrelation* $\text{reach}_{\mathcal{B}} \subseteq Q \times \Sigma^* \times Q$ ist die kleinste Relation, die für alle $q, q', q'', q''' \in Q$ und für alle $w \in \Sigma^*$ die folgenden Eigenschaften erfüllt:

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

Für den ε -NEA aus Bsp. 2.16 können wir $\text{reach}_{\mathcal{B}}$ mit Hilfe der folgenden Tabelle angeben. Dabei bedeutet der Eintrag von einer Sprache L in Zeile q_i und Spalte q_j , dass für alle $w \in L$ das Tripel (q_i, w, q_j) in $\text{reach}_{\mathcal{B}}$ enthalten ist.

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

Def. 2.19: Ein Wort $w \in \Sigma^*$ wird von $\mathcal{B}$ *akzeptiert*, wenn $(q^{\text{init}}, w, q_f) \in \text{reach}_{\mathcal{B}}$ für ein $q_f \in F$. Die von $\mathcal{B}$ akzeptierte Sprache $L(\mathcal{B})$ ist die Menge der von $\mathcal{B}$ akzeptierten Wörter, d.h. $L(\mathcal{B}) = \{w \in \Sigma^* \mid \exists q \in F : (q^{\text{init}}, w, q) \in \text{reach}_{\mathcal{B}}\}$. $\oplus$

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 $\delta(q, \varepsilon) = \{q\}$ für alle $q \in Q$. Für die Sprachgleichheit zeigen wir via Induktion über die Länge von w , dass für alle Wörter $w \in \Sigma^*$ gilt:

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

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

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

Vorlesung:
14.11.18

Zur Vorbereitung des Beweises machen wir zunächst die folgende Definition.

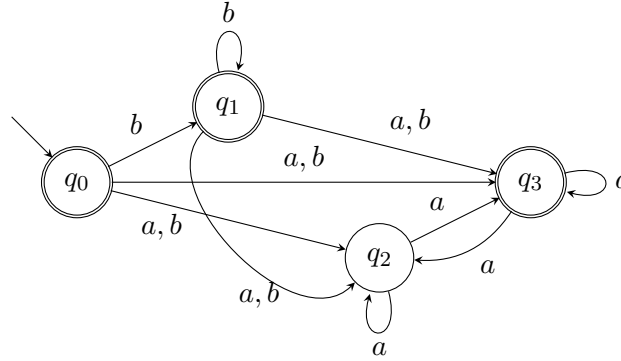
Def. 2.20 (ε -freier Automat): Für einen gegebenen ε -NEA $\mathcal{B} = (\Sigma, Q, \delta, q^{\text{init}}, F)$ definieren wir den NEA $\mathcal{N} = (\Sigma, Q, \delta_{\mathcal{N}}, q^{\text{init}}, F_{\mathcal{N}})$ mit

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

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

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

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



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

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

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

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

$$\begin{aligned} w \in L(\mathcal{B}) &\stackrel{\text{Def. 2.3}}{\Leftrightarrow} \exists q_f \in F : (q^{\text{init}}, w, q_f) \in \text{reach}_{\mathcal{B}} \\ &\Leftrightarrow \exists q_f \in F_{\mathcal{N}} : \underbrace{q_0, q_1, \dots, q_n}_{\text{Lauf}} \in Q, \text{ sodass } n = |w|, q_0 = q^{\text{init}} \text{ und } q_n = q_f \\ &\stackrel{\text{Def. 2.19}}{\Leftrightarrow} w \in L(\mathcal{N}) \end{aligned}$$

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

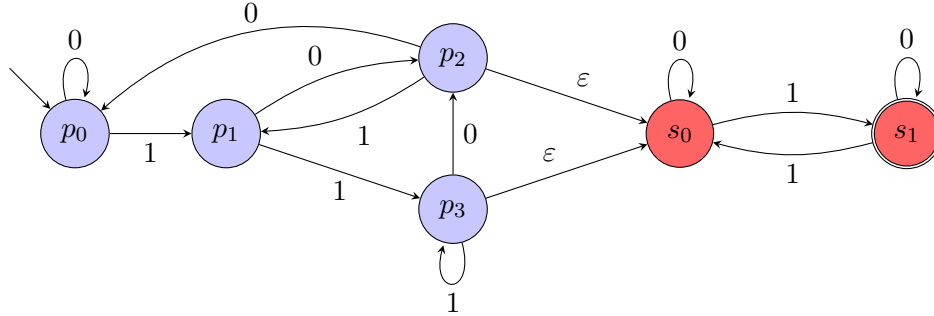
Mit Hilfe der ε -NEAs greifen wir nun die zu Beginn von Abschnitt 2.5 aufgeworfene Fragestellung wieder auf und zeigen, dass für je zwei reguläre Sprachen auch die Konkatination regulär ist.

Wir geben hierfür zunächst eine Konstruktion an.

Def. 2.21: Gegeben zwei ε -NEA $\mathcal{B}_i = (\Sigma, Q_i, \delta_i, q_i^{\text{init}}, F_i)$, $i = 1, 2$, definieren wir den ε -NEA für Konkatination $\mathcal{B} = (\Sigma, Q, \delta, q^{\text{init}}, F)$ wie folgt.

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

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



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

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

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

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

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

2.6 Abschlusseigenschaften

Def. 2.22: Eine Menge X heißt *abgeschlossen* unter Operation $f : X^n \rightarrow X$, falls $\forall x_1, \dots, x_n \in X : f(x_1, \dots, x_n) \in X$. $\oplus$

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

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

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

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

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

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

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

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

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

2.7 Reguläre Ausdrücke

2.7.1 Motivation

Vorlesung:
16.11.18

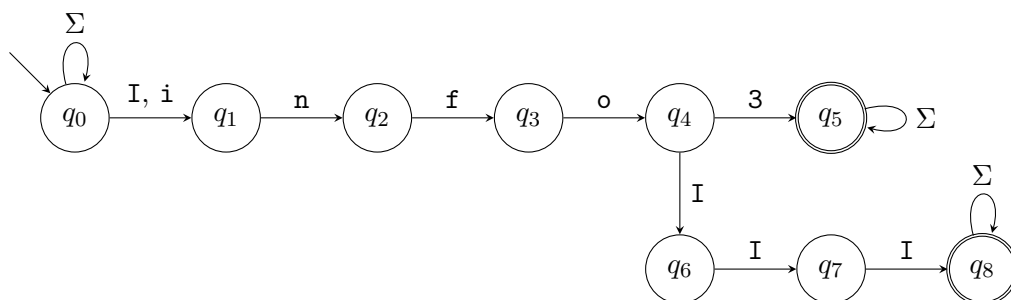
Problemstellung: Sie haben auf Ihrem Rechner irgendwo eine Textdatei mit Notizen zu dieser Vorlesung, können sich aber gerade nicht an den Pfad erinnern. Sie wissen aber noch, dass die Zeichenkette `info3` oder `infoIII` enthalten ist. Möglicherweise war das `i` am Anfang aber auch groß geschrieben. 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. 2.23: Die Menge $RE(\Sigma)$ der *regulären Ausdrücke* über Σ ist induktiv definiert durch:

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

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

Bsp. 2.19:

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

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

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

Def. 2.24: Die durch einen regulären Ausdruck *beschriebene Sprache* $\llbracket \cdot \rrbracket : RE(\Sigma) \rightarrow \mathcal{P}(\Sigma^*)$ ist induktiv definiert durch:

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

Bsp. 2.20:

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

Bsp. 2.21: Die aus Bsp. 2.11 bekannte Sprache

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

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

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

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

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

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

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

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

Def. 2.25: Sei $\mathcal{A} = (\Sigma, Q, \delta, q^{\text{init}}, F)$ ein DEA. Für einen Zustand $q \in Q$ ist die *Sprache des Zustands* $L_q = \{w \in \Sigma^* \mid \tilde{\delta}(q, w) \in F\}$ die Sprache der Wörter, die von Zustand q aus in einen akzeptierenden Zustand führen. $\oplus$

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

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

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

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

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

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

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

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

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

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

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

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

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

Vorlesung:
21.11.18

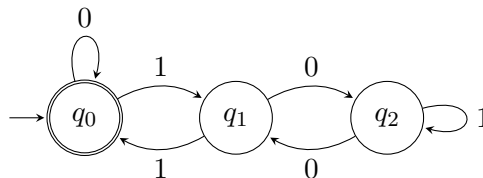


Abb. 4: DEA „modulo 3“

Wir erhalten das folgende Gleichungssystem mit drei Unbekannten.

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

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

Lemma 2.13 (Ardens Lemma):

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

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

BEWEIS:

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

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

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

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

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

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

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

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

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

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

Dann ist der reguläre Ausdruck

$$r_A^* r_B$$

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

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

Vorlesung:
23.11.18

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

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

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

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

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

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

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

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

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

(Zusammenlegen der Summen und Ausklammern von r_j)

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

Nach diesen Umformungen ergeben sich 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 $\llbracket r_0 \rrbracket = L_{q_0} = L_{q_{\text{init}}} = L(\mathcal{A})$. $\square$

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

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

Ardens Lemma auf r_2 anwenden:

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

Einsetzen in r_1 :

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

Ardens Lemma auf r_1 anwenden:

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

Einsetzen in r_0 :

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

Ardens Lemma auf r_0 anwenden:

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

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. $\oplus$

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

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

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

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

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

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

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

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

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

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

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

^{††}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: $1001 \in L(\mathcal{G})$.

Vorlesung:
28.11.18

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

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

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

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

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

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

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

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

Wir zeigen nun via Induktion über n die folgende Eigenschaft:

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

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: $d_j < 1$.[†]

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 $d(w_2) = 0$.

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

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

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

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

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

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

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

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

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

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

Def. 3.3 (Chomsky-Hierarchie):

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

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

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

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[†] *Typ-2-Grammatik* transformiert werden kann. Diese erfüllt dann auch alle Bedingungen einer *Typ-1-Grammatik*.

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

- Jede Typ-1-Grammatik ist auch eine Typ-0-Grammatik.

Im Lauf der Vorlesung werden wir die folgende Aussage zeigen:

Sei CHi die Menge der Typ- i Sprachen; dann gilt: $CH3 \subsetneq CH2 \subsetneq CH1 \subsetneq CH0$.

3.1 Kontextfreie Sprachen

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

Vorlesung:
30.11.2018

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

$$P = \{E \rightarrow a \mid E + E \mid E * E\}^\dagger$$

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

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

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

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

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

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

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

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

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

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

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

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

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

Das *abgeleitete Wort* zu einem $\mathcal{T} \in \text{Abl}(A)$, $Y(\mathcal{T})^\dagger$, ist definiert durch

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

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

⊕

Bemerkung: Nach dieser Definition gibt es also insbesondere für jede Regel π bei der nur Terminalsymbole auf der rechten Seite vorkommen einen Ableitungsbaum der die einelementige Knotenmenge $\{\pi\}$ und einer leeren Menge von Kanten hat.

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

$$\begin{array}{ccc} E \rightarrow a & \begin{array}{c} E \rightarrow E + E \\ \swarrow \quad \searrow \\ E \rightarrow a \quad E \rightarrow E * E \\ \quad \quad \swarrow \quad \searrow \\ \quad \quad E \rightarrow a \quad E \rightarrow a \end{array} & \begin{array}{c} E \rightarrow E * E \\ \swarrow \quad \searrow \\ E \rightarrow E + E \quad E \rightarrow a \\ \quad \swarrow \quad \searrow \\ E \rightarrow a \quad E \rightarrow a \end{array} \end{array}$$

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

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

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

(Ohne Beweis.)

Def. 3.5:

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

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).

3.2 Die Chomsky-Normalform für kontextfreie Sprachen

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

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

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

Dafür definieren wir die Größe einer Grammatik wie folgt.

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

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

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

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

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

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

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

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

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

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

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

BEWEIS: Ersetze jede Regel der Form

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

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

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

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

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

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

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

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

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

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

Vorlesung:
5.12.2018

BEWEIS: Definiere M_i als die Menge der Nichtterminalsymbole, aus denen sich ε mit einem Ableitungsbaum der Höhe $< i$ ableiten lässt:[†]

$$M_0 = \emptyset$$

$$M_{i+1} = M_i \cup \{A \mid A \rightarrow \alpha \in P \text{ und } \alpha \in M_i^*\}$$

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
    M_old = M
    foreach A → a ∈ P:
        if (A ∉ M ∧ a ∈ M_old*):
            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

[†] „ $\dot{\cup}$ “ steht für „disjunkte Vereinigung“

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

- es existiert $A \rightarrow A_1 \dots A_n \in P$ mit $A_j \in M_i$ für $1 \leq j \leq n$. Nach IV existieren Ableitungen

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

sodass auch eine Ableitung von A nach ε existiert:

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

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

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

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

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

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

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

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

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

BEWEIS:

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

5. Entferne alle Regeln $A \rightarrow \varepsilon$ für $A \neq S'$. □

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

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

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

Vorlesung:
7.12.2018

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

BEWEIS: (Beweisskizze)

- Zur Äquivalenz:

Wir zeigen dass jeden Schritt der die CFG ändert eine äquivalente CFG liefert. Dazu zeigen wir dass es für jedes Wort dass in der alten CFG abgeleitet werden konnte auch in der neuen CFG abgeleitet werden kann und umgekehrt. Für Schritt 6 können wir – es macht den Beweis etwas einfacher – zeigen dass Äquivalenz sogar für jede Iteration der mittleren for-Schleife gilt.

- Zur Kettenregelfreiheit:

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

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

- Zu Laufzeit und Größe des Resultates:

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

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

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

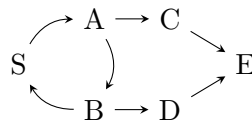
1. Setze $N' = N$ und $P' = P$
2. Betrachte den gerichteten Graphen G dessen Knotenmenge N und Kantenmenge $\{(A, B) \mid A \rightarrow B \in P'\}$ ist. (Es gibt also genau eine Kante pro Kettenregel.)
3. Suche, z.B. mittels Tiefensuche, einen Zyklus in G . Wenn kein Zyklus gefunden wurde, fahre mit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

Abb. 5: Algorithmus UNIT

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

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

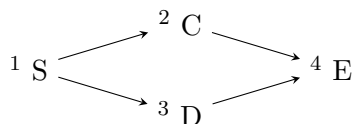
Nach der erstmaligen Anwendung von Schritt 2 erhalten wir den folgenden gerichteten Graphen.



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

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

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



^{††}gerichteter azyklischer Graph (DAG) (engl. *directed acyclic graph*).

^{††}Topologies sortieren ist ein aus der Informatik 2 Vorlesung bekannter Begriff und bezeichnet das Erweitern einer gegebenen partiellen Ordnung zu einer totalen Ordnung.

Am Ende von Schritt 6 sieht die Regelmengende P' wie folgt aus[†].

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

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

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

BEWEIS: (Beweisskizze) Ein Algorithmus der diese Eigenschaften erfüllt ist das Hintereinanderschalten von SEP, BIN, DEL und UNIT.^{††}

- Zur Korrektheit:

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

- Zu Laufzeit und Größe des Resultates:

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

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

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

[†] Das Beispiel zeigt dass das Resultat des Algorithmus völlig unnötige Regeln enthalten kann.

^{††} Es genügt sogar nur DEL und dann UNIT anzuwenden, da DEL schon SEP und BIN ausführt.

3.3 Wortproblem und Leerheitsproblem für kontextfreie Sprachen

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

BEWEIS (CYK-Algorithmus): Sei $w = a_1 \dots a_n \in \Sigma^*$. Der CYK-Algorithmus aus Abb. 6 erfüllt diese Eigenschaften. Wir wollten zunächst die Idee des Algorithmus erklären.

Definiere für $1 \leq i < j \leq n$:

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

Wir definieren also Mengen von Nichtterminalsymbolen die sich schön als obere Dreiecksmatrix darstellen lassen. Die Matrix enthält einen Eintrag pro Teilwort von w , genauer der Eintrag M_{ij} enthält das Nichtterminalsymbol A genau dann wenn wir von A das Teilwort vom i -ten Zeichen bis zum j -ten Zeichen ableiten können. Zum Lösen des eigentlichen Problems hilft dieses Vorgehen zunächst nichts. Ursprünglich wollte wir wissen ob wir das Wort W aus de Startsymbol ableiten können. Nun müssen wir diese Frage für alle Teilwörter und alle Nichtterminalsymbole beantworten.

Offensichtlich gilt: $w \in L(\mathcal{G})$ genau dann, wenn $S \vdash_{\mathcal{G}}^* w$ genau dann, wenn $S \in M_{1n}$. Zum Lösen des eigentlichen Problems hilft uns das betrachten der Matrix also zunächst nicht. Ursprünglich wollte wir wissen ob wir das Wort W aus de Startsymbol ableiten können. Nun müssen wir diese Frage für alle Teilwörter und alle Nichtterminalsymbole beantworten.

Es gilt für $i = j$

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

Die Einträge auf der Hauptdiagonalen hängen also nur von Regeln der Form $A \rightarrow a$ ab und wir können diese Einträge jeweils durch einmaliges Iterieren über alle Regeln bestimmen.

Es gilt für $i < j$

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

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

Ausgabe: “Ja” falls $a_1 \dots a_n \in L(\mathcal{G})$, “Nein” sonst.

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

```

for i=1 .. n do //  $O(|\mathcal{G}|) \cdot O(n)$ 
     $M_{ii} = \{A \mid A \rightarrow a_i\}$ 
for i=n-1 .. 1 do
    for j=i+1 .. n do
        for k=i .. j-1 do //  $O(|\mathcal{G}|) \cdot O(n^3)$ 
             $M_{ij} = M_{ij} \cup \{A \mid A \rightarrow BC, B \in M_{ik}, C \in M_{(k+1)j}\}$ 
return  $S \in M_{1n}$ 

```

Abb. 6: CYK Algorithmus

Wir können also die Berechnung der Matrixeinträge die nicht auf der Hauptdiagonalen liegen auf die Existenz bestimmter Regel der Form $A \rightarrow BC$ und Inhalte bestimmter anderer Matrixeinträge zurückführen.

Beobachtung: Der Wert von M_{ij} hängt nur von $M_{i'j'}$ mit $i' > i, j' \leq j$ oder $i' \geq i, j' < j$ ab. Wir können die M_{ij} also mit dem Verfahren aus Abb. 6 berechnen.

Es gibt also insbesondere keine zyklischen Abhängigkeiten und bei geeigneter Iterationsreihenfolge über alle Matrixeinträge kann jeder Eintrag direkt mit Hilfe der Einträge aus vorherigen Iterationen bestimmt werden.

Skizze der verbleibenden Beweisschritte.

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

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

Zum Speicherplatz: Wir benötigen weniger als $|w|^2$ Matrixeinträge und jeder Eintrag enthält höchstens alle Variablen die in Regeln der Grammatik vorkommen. $\square$

Bsp. 3.9: Sei $\Sigma = \{a, b, c\}$. Betrachte das Wort $w = aaabbbcc$ und die CFG $\mathcal{G} =$

$(\Sigma, \{S, X, Y\}, P, S)^\dagger$ deren Regeln P wie folgt definiert sind.

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

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

$$\begin{aligned} S &\rightarrow XY \\ X &\rightarrow AB \mid AZ \\ Z &\rightarrow XB \\ Y &\rightarrow CY \mid c \\ A &\rightarrow a, B \rightarrow b, C \rightarrow c \end{aligned}$$

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

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

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

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

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

BEWEIS: Aus Satz 3.8 wissen wir dass eine CFG in CNF konvertieren können sodass deren Größe höchstens quadratisch wächst. Anschließend wenden wir den CYK Algorithmus auf das Resultat an. $\square$

Um möglichst schnell zum Unterkapitel über Typ-3-Sprachen zu kommen wurde das restliche Unterkapitel in der Freitagsvorlesung übersprungen.

Vorlesung:
12.12.2018

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

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

Def. 3.10: Sei $\mathcal{G} = (\Sigma, N, P, S)$ eine beliebige Grammatik. Wir nennen ein Nichtterminalsymbol $A \in N$ co-erreichbar falls es ein Wort $w \in \Sigma$ existiert, sodass $A \vdash_{\mathcal{G}}^* w$ gilt. Ein Symbol ist also co-erreichbar genau dann wenn wir daraus irgendein Wort ableiten können. $\oplus$

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

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

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

Wir können M induktiv berechnen. Wir definieren hierfür:

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

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

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

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

3.4 Einschub: Typ-3-Sprachen

Satz 3.13: L ist regulär $\Leftrightarrow L$ ist Typ-3-Sprache.

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

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

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

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

[†]Gefragt ist, ob $L(\mathcal{G}) = \{w \in \Sigma^* \mid S \vdash_{\mathcal{G}}^* w\} \stackrel{?}{=} \emptyset$.

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

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

Wir machen nun die folgenden Modifikationen.

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

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

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

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

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

Lemma 3.14: $CH3 \subsetneq CH2$ (d.h., die regulären Sprachen sind eine *echte* Teilmenge der kontextfreien Sprachen).

BEWEIS: Betrachte $L_{\text{centered}} = \{0^n 10^n \mid n \in \mathbb{N}\} \subseteq \{0, 1\}^*$

Sowohl aus Bsp. 2.10 als auch aus Bsp. 2.14 ist bekannt, dass L_{centered} nicht regulär ist.

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

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

(ohne Beweis) □

4 Kellerautomaten pushdown automaton (PDA)

Die im letzten Kapitel vorstellten kontextfreien Sprachen können wir bisher nur mit Hilfe von kontextfreien Grammatiken darstellen. Für die regulären Sprachen aus Kapitel 2 haben wir dagegen mit DEAs, NEAs, ε -NEAs, regulären Ausdrücken und regulären Grammatiken verschiedene Darstellungsformen kennengelernt. Die Automaten hatten dabei für uns einen besonderen Reiz, da diese Darstellungsform Ähnlichkeiten mit Computern hat. In diesem Kapitel werden wir nun den Kellerautomaten, ein Automatenmodell für die kontextfreien Sprachen, kennenlernen.

Ein Kellerautomat ist im wesentlichen ein ε -NEA, der um einen Stapelspeicher mit unbeschränkter Kapazität erweitert wurde.

Bei jedem Zustandsübergang darf der Kellerautomat das oberste Element des Stapelspeichers inspizieren und durch eine (potentiell leere) Sequenz von Elementen ersetzen. Der Automat darf dabei die Entscheidung über den Folgezustand vom inspizierten Element abhängig machen. Zu Beginn liegt genau ein Element, das *Kellerbodensymbol*, auf dem Stapelspeicher.

Der Kellerautomat unterscheidet sich noch in einem weiteren Detail vom ε -NEA: Es gibt keine akzeptierende Zustände. Der Kellerautomat akzeptiert stattdessen, wenn am Ende eines Zustandsübergangs das Eingabewort komplett gelesen wurde und der Stapelspeicher leer ist.

Analog zu endlichen Automaten können wir auch Kellerautomaten mit Hilfe eines Zustandsdiagramms beschreiben. Ein Beispiel ist in Abb. 7 dargestellt. Alle Transitionen sind mit einem Tripel beschriftet. Die erste Komponente des Tripels ist dabei das gelesene Zeichen der Eingabe, die zweite Komponente ist das vom Stapelspeicher genommene Element und die dritte Komponente ist die Sequenz, die auf den Stapelspeicher gelegt wird.

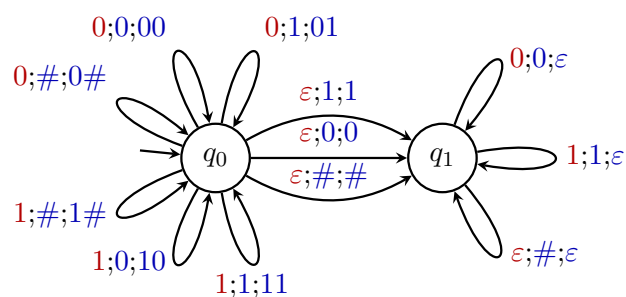


Abb. 7: Zustandsdiagramm eines Kellerautomaten

Der in Abb. 7 dargestellte Kellerautomat akzeptiert die Sprache $L = \{ww^R \mid w \in \{0,1\}^*\}$

und verwendet das Zeichen $\#$ als Kellerbodensymbol. In Zustand q_0 wird für jedes Eingabezeichen das entsprechende Symbol auf den Stapelspeicher gelegt. Zu jedem Zeitpunkt kann der Automat nichtdeterministisch (mit Hilfe einer ε -Transition) in den Zustand q_1 wechseln. Diese Transition sollte genau bei der Hälfte des Eingabeworts genommen werden. In Zustand q_1 vergleicht der Automat sukzessive die verbleibenden Eingabesymbole mit dem Inhalt des Stapelspeichers. Nur bei Übereinstimmung wird der Stapelspeicher bis zum Kellerbodensymbol geleert. Falls nur noch das Kellerbodensymbol auf dem Stapelspeicher liegt, kann der Automat dieses mit der Transition „ $\varepsilon; \#; \varepsilon$ “ entfernen und die Eingabe akzeptieren.

Auf Englisch heißen Kellerautomaten *pushdown automata* und wir werden die Abkürzung PDA für Kellerautomaten verwenden. In diesem Kapitel befassen wir uns zunächst nur mit nichtdeterministischen Kellerautomaten und definieren diese formal wie folgt.

Def. 4.1: Ein nichtdeterministischer Kellerautomat (NPDA)[†] ist ein 6-Tupel

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

Dabei ist

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

Bsp. 4.1: Wir beschreiben den in Abb. 7 abgebildeten Kellerautomaten formal wie folgt.

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

[†]Englisch: Nondeterministic Pushdown Automaton

Hat die graphische Darstellung eines Kellerautomaten sehr viele Transitionen müssen wir diese nicht alle explizit Zeichnen. Statt dessen können wir auch wie in Abb. 8 Variablen verwenden um mehrere Transitionen zusammenzufassen.

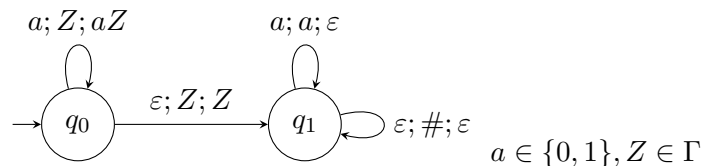


Abb. 8: Zustandsdiagramm der Kellerautomaten aus Abb. 7. Statt alle Transitionen explizit zu zeichnen verwenden wir die Variablen a und Z um eine Menge von Transitionen durch eine einzige Kante zu repräsentieren.

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

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

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

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

ist definiert durch

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

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

Vorlesung:
12.12.2018

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

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

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

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

[†] Zur Erinnerung: Die reflexive, transitive Hülle einer binären Relation R ist die kleinste Obermenge von R die reflexiv und transitiv ist. Bsp. Die reflexive, transitive Hülle der direkter-Nachfolger-Relation $\{(a, b) \in \mathbb{Z} \times \mathbb{Z} \mid b = a + 1\}$ ist die kleinergleich Relation " $\leq$ ".

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

- $(q_0, 0110, \#)$
- $\triangleright (q_0, 110, 0\#)$ („pushen“ des Eingabesymbols 0)
- $\triangleright (q_0, 10, 10\#)$ („pushen“ des Eingabesymbols 1)
- $\triangleright (q_1, 10, 10\#)$ (ε -Übergang von q_0 nach q_1)
- $\triangleright (q_1, 0, 0\#)$ („poppen“ des Eingabesymbols 1)
- $\triangleright (q_1, \varepsilon, \#)$ („poppen“ des Eingabesymbols 0)
- $\triangleright (q_1, \varepsilon, \varepsilon)$ (ε -Übergang zum Entfernen von $\#$)

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

Zum Führen des Beweises benötigen wir das folgende Lemma.

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

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

BEWEIS: Übungsaufgabe. □

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

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

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

- Beweisrichtung „ $\supseteq$ “

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

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

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

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

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

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

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

Mit Lemma 4.2 folgt schließlich:

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

• Beweisrichtung „ $\subseteq$ “

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

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

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

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

Es gibt zwei Fälle für Z :

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

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

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

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

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

Das folgende Lemma beantwortet diese Frage.

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

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

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

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

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

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

Definiere $\mathcal{G} = (\Sigma, N, P, S)$ mit $N = (Q \times \Gamma \times Q) \cup \{S\}$. Wir wählen diese Tripel als Nichtterminalsymbole, da wir eine Grammatik konstruieren wollen, welche die folgende Eigenschaft hat:

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

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

Als Menge der Regeln wählen wir $P = P_S \cup P_0 \cup P_1 \cup P_2$ mit:

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

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

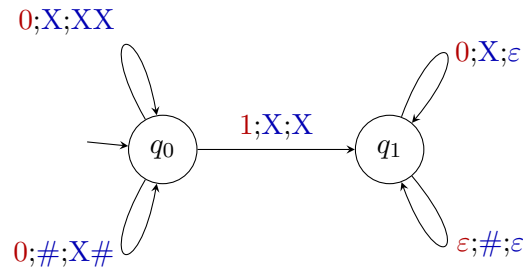
$$P_1 = \{[q, Z, q'] \rightarrow x[q'', Z', q'] \mid (q'', Z') \in \delta(q, x, Z), q' \in Q, x \in \Sigma \cup \{\varepsilon\}\},$$

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

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

Beispiel:

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



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

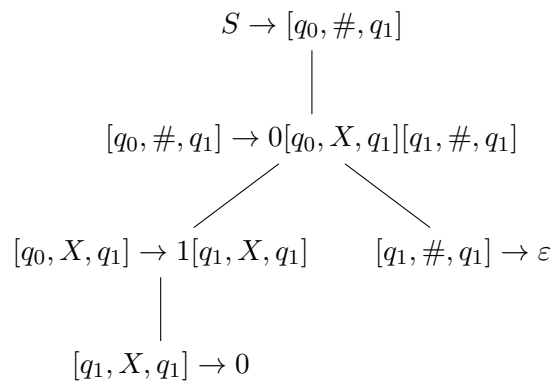
$$P_S = \{S \rightarrow [q_0, \#, q_0] \\ S \rightarrow [q_0, \#, q_1]\}$$

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

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

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

Ableitungsbaum für 010:



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

Es bleibt zu zeigen, dass die Eigenschaft $(*)$ gilt. Wir zeigen dafür via Induktion über n dass für alle $n \in \mathbb{N}$ die folgende Eigenschaft gilt:

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

I.A. $n = 1$: Folgt direkt aus den P_0 -Regeln (und der Tatsache, dass keine der anderen Regeln ein einzelnes Symbol oder ε ableiten kann).

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

- Beweisrichtung „ $\Rightarrow$ “

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

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

Dann existiert $x \in \Sigma \cup \{\varepsilon\}$ und $w' \in \Sigma^*$ sodass $w = xw'$ und $[q'', Z', q'] \vdash^n w'$. Nach I.V. gilt nun, dass $(q'', w', Z') \triangleright^n (q', \varepsilon, \varepsilon)$.

Nach Konstruktion gilt $(q'', Z') \in \delta(q, x, Z)$ und somit $(q, w, Z) \triangleright (q'', w', Z') \triangleright^n (q', \varepsilon, \varepsilon)$.

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

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

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

Per I.V. folgt

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

Mit Lemma 4.2 folgt auch

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

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

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

Somit gilt:

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

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

– Fall 1: $(q, w, Z) \triangleright (q'', w', Z')$ mit $w = xw'$ für $x \in \Sigma \cup \{\varepsilon\}$

Aus $(q'', w', Z') \triangleright^n (q', \varepsilon, \varepsilon)$ folgern wir mit I.V. $[q'', Z', q'] \vdash^n w'$. Nach Konstruktion können wir $[q, Z, q'] \vdash x[q'', Z', q']$ ableiten.

– Fall 2: $(q, w, Z) \triangleright (q_1, w', Z_1Z_2)$ mit $w = xw'$ für $x \in \Sigma \cup \{\varepsilon\}$

Aus $(q_1, w', Z_1Z_2) \triangleright^n (q', \varepsilon, \varepsilon)$ folgern wir: $\exists k_1, k_2 \in \mathbb{N}, q_2 \in Q, w_1, w_2 \in \Sigma^*$, sodass gilt:

1. $k_1 + k_2 = n$
2. $w' = w_1w_2$
3. $(q_1, w', Z_1Z_2) \triangleright^{k_1} (q_2, w_2, Z_2)$
4. $(q_2, w_2, Z_2) \triangleright^{k_2} (q', \varepsilon, \varepsilon)$

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

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

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

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

Somit gibt es auch eine Ableitung

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

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

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

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

5 Kontextfreie Sprachen – Teil 2

5.1 Das Pumping Lemma für kontextfreie Sprachen

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

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

Lemma 5.2: In jedem Binärbaum[†] mit $\geq 2^k$ Blättern gibt es mindestens ein Blatt sodass der Pfad von der Wurzel zu diesem Blatt Länge $\geq k$ hat. (ohne Beweis)

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

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

Folge ζ Rückwärts vom Blatt in Richtung Wurzel, bis sich eine linke Regelseite A das erste Mal wiederholt. Das geschieht nach $\leq |N|$ Schritten. Teile $z = uvwxy$ wie in Abb. 9a skizziert. Es gilt:

- $|vx| \geq 1$, da ζ entweder durch den B -Nachfolger oder durch C -Nachfolger läuft. Angenommen, ζ verläuft durch den B -Nachfolger. Somit muss $C \stackrel{*}{\vdash} x$ gelten. In CNF ist $C \stackrel{*}{\vdash} \varepsilon$ nicht möglich und somit ist $|x| \geq 1$. Der Fall, dass ζ durch den C -Nachfolger verläuft, ist analog.
- $|vwx| \leq 2^{|N|} = n$, da der obere Knoten dessen linke Regelseite mit A beschriftet ist höchstens $|N|$ Schritte von der Blattebene entfernt und $\mathcal{T}$ ein Binärbaum ist.
- Für die Aussage $\forall i \in \mathbb{N} : uv^iwx^iy \in L$ unterscheiden wir drei Fälle:

[†] Sie kennen Bäume bereits aus der Informatik 2 Vorlesung. In unserer Vorlesung sind Bäume immer gerichtet Ein *Baum* ist ein gerichteter Graph mit einer ausgezeichneten Knoten den wir *Wurzel* nennen und für den die folgenden Eigenschaften gelten: Die Wurzel hat keine eingehenden Kanten und von der Wurzel können wir jeden anderen Knoten über genau einen Pfad erreichen. Wir nennen Knoten ohne Ausgehende Kanten *Blätter*. Knoten die keine Blätter sind heißen *innere Knoten*. Ein Binärbaum ist ein Baum bei dem jeder innere Knoten zwei Nachfolger hat.

^{††} Dieser Beweis wurde in der Vorlesung mit Hilfe von Abb. 9 erklärt aber nicht an die Tafel geschrieben.

- $i = 0$: Aus dem „unteren“ A haben wir w abgeleitet, aus dem oberen A haben wir vw abgeleitet. Wir könnten also auch aus dem oberen A direkt w ableiten und erhalten $S \vdash^* uwy$. (Siehe Abb. 9b).
- $i = 1$: Gilt trivialerweise, da $uv^1wx^1y = z$.
- $i \geq 1$: Aus dem „oberen“ A haben wir vAx abgeleitet, aus dem unteren A haben wir w abgeleitet. Wir könnten also auch aus dem unteren A nochmal vAx ableiten. Wenn wir dies oft genug wiederholen, können wir jedes $uv^iwx^i y$ mit $i \geq 2$ erzeugen (siehe auch Abb. 9c).

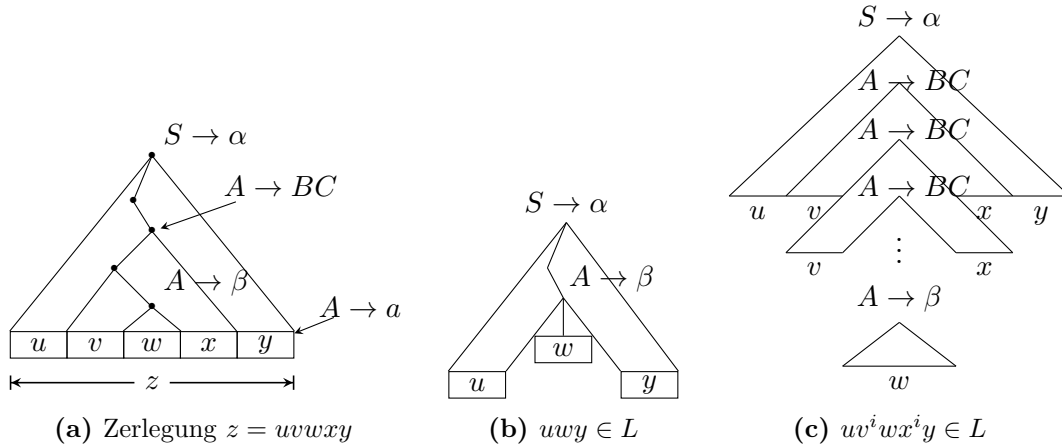


Abb. 9: Illustration des Argumentes aus dem Beweis von Satz 5.1

□

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

BEWEIS: Wir nehmen an, L wäre kontextfrei. Sei $n \in \mathbb{N}$ mit $n > 0$ (die Zahl aus dem PL). Wir betrachten $z = a^n b^n c^n$. Es gilt also $|z| = 3n \geq n$. Sei $z = uvwxy$ eine Zerlegung mit $|vx| \geq 1$ und $|vwx| \leq n$. Wir zeigen nun, dass $\forall i \in \mathbb{N} : uv^iwx^iy \in L$ nicht gilt (und folgern, dass unsere Annahme falsch war).

Wegen $|vwx| \leq n$ trifft einer folgenden fünf Fälle zu.

- $vwx = a^j$ mit $j > 0$: Es gilt $|v^0wx^0| < j$ und somit $uv^0wx^0y \notin L$, da die Anzahl der a nicht mit der Anzahl der b übereinstimmt.
- $vwx = a^k b^j$ mit $k > 0$ und $j > 0$.
 - Falls $v = a^{k'}$ und $x = b^{j'}$, dann gilt $uv^0wx^0y \notin L$ da die Anzahl der a oder die Anzahl der b nicht mehr mit der Anzahl der c übereinstimmt.

- Falls v oder x die Form a^+b^+ hat, gilt $uv^2wx^2y \notin L$ da uv^2wx^2y ein Teilwort der Form $a^+b^+a^+$ enthält.
- $vw x = b^j$: analog zu Fall a^j
- $vw x = b^k c^j$: analog zu Fall $a^k b^j$
- $vw x = c^j$: analog zu Fall a^j

Die Sprache L ist also nicht kontextfrei, denn wir erhalten in jedem der fünf Fälle einen Widerspruch zur Aussage des Pumping Lemma. $\square$

Lemma 5.4: Die Menge der kontextfreien Sprachen (CH2) ist eine echte Teilmenge der Menge der kontextsensitiven Sprachen (CH1).

BEWEIS: Aus wissen wir bereits, dass $CH2 \subseteq CH1$ gilt. Aus Bsp. 3.3 wissen wir, dass $L = \{a^n b^n c^n \mid n \geq 1\}$ von einer kontextsensitiven Grammatik erzeugt wird. Mit Hilfe von Lemma 5.3 folgern wir $CH2 \subsetneq CH1$. $\square$

5.2 Abschlusseigenschaften für kontextfreie Sprachen

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

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

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

$\square$

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

BEWEIS: für $n \geq 1$

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

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

Satz 5.7: Die CFL sind *nicht* unter Komplement ($\bar{\cdot}$) abgeschlossen.

BEWEIS: Angenommen, CFL wäre abgeschlossen unter $\bar{\cdot}$.

Falls $L_1, L_2 \in \text{CFL}$, dann ist $\bar{L}_1, \bar{L}_2 \in \text{CFL}$ nach Annahme.

$\Rightarrow \overline{\bar{L}_1 \cup \bar{L}_2} \in \text{CFL}$ wegen Abschluss unter „ $\cup$ “.

$\Rightarrow \overline{\bar{L}_1 \cup \bar{L}_2} = L_1 \cap L_2 \in \text{CFL}$ – Widerspruch zu Nicht-Abschluss unter „ $\cap$ “.

□

5.3 Deterministisch kontextfreie Sprachen

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

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

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

Weiterhin definieren wir das Akzeptanzverhalten folgendermaßen:

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

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

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

⊕

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

Def. 5.2: Wir nennen sowohl einen gewöhnlichen Kellerautomaten als auch einen Kellerautomaten mit akzeptierenden Zuständen *deterministisch*, wenn für alle $q \in Q, a \in \Sigma, Z \in \Gamma$ gilt, dass $|\delta(q, a, Z)| + |\delta(q, \varepsilon, Z)| \leq 1$.

Wir verwenden die Abkürzung deterministischer Kellerautomat (DPDA) für den deterministischen Kellerautomaten mit akzeptierenden Zuständen.

⊕

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

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

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

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

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

und Transitionsfunktion δ_S gegeben durch

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

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

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

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

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

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

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

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

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

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

Es verbleibt zu zeigen:

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

Def. 5.3: Wir nennen eine Sprache L *deterministisch kontextfrei* wenn es einen DPDA gibt der L akzeptiert. ⊕

Satz 5.10: Die deterministisch kontextfreien Sprachen sind unter Komplement abgeschlossen.

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

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

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

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

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

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

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

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

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

$$\delta'((q, 0), \varepsilon, Z) = \{(q', acc(q'), \gamma)\}$$

$$\delta'((q, 1), \varepsilon, Z) = \{(q', 1, \gamma)\}$$

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

$$\delta'((q, 0), a, Z) = \{((q, 2), Z)\}$$

$$\delta'((q, 2), a, Z) = \{(q', acc(q'), \gamma)\}$$

$$\delta'((q, 1), a, Z) = \{(q', acc(q'), \gamma)\}$$

□

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

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

6 Turingmaschinen

Vorlesung:
09.01.2019

Wir werden in diesem Kapitel ein weiteres Maschinenmodell, das der *Turingmaschine* [†], kennenlernen. Im Gegensatz zu endlichen Automaten und Kellerautomaten sind Turingmaschinen so “mächtig” dass die Arbeitsweise jeder Rechenmaschine durch eine Turingmaschine modelliert werden kann.

Diese “Mächtigkeit” stellt uns vor neue Herausforderungen: Eine Turingmaschine kann nicht-triviale Endlosschleifen erlauben und wir werden im Allgemeinen nicht entscheiden können ob eine Berechnung überhaupt terminiert oder statt dessen unendlich lange läuft.

Die Turingmaschine basiert auf der Idee, dass man die Arbeitsweise eines rechnenden Menschen abstrakt und idealisiert wie folgt beschreiben könnte: Wir bewegen uns mit einem Bleistift und einem Radiergummi auf einem Karopapier, das in alle Richtungen unbeschränkt ist (weil wir immer zusätzliches Papier anlegen könnten wenn der Seitenrand erreicht ist). In jedem Rechenschritt lesen wir zunächst das Zeichen das in dem Kästchen unter der Bleistiftspitze steht. Anschließend radieren wir das Zeichen aus, beschriften das Kästchen neu und bewegen die Bleistiftspitze zu einem benachbarten Kästchen oder lassen die Position unverändert. Unser Gehirn hat nur endlich viele verschiedene Zustände und steuert welches Zeichen wir schreiben und wohin die Bleistiftspitze bewegt wird. Der Zustand unseres Gehirns kann sich in jedem Rechenschritt ändern und wird auch durch das gelesene Zeichen beeinflusst.

Im Gegensatz zu diesem Modell eines rechnenden Menschen hat die Turingmaschine ein in beide Richtungen unbeschränktes Band statt eines Karopapiers und einen Schreib-Lesekopf statt Bleistift und Radiergummi.

6.1 Turingmaschine (informell)

Die Turing-Maschine (TM) ist eine Erweiterung des endlichen Automaten mit den folgenden Zusätzen.

- Die Eingabe steht auf einem Band, das rechts und links der Eingabe unendlich viele Zellen hat.

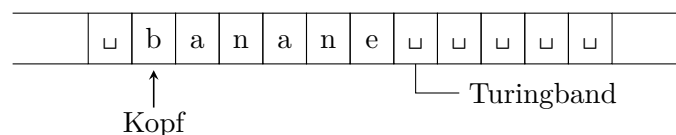


Abb. 10: Turingband

[†]Alan Mathison Turing *1912 †1954

- Der Kopf der TM kann nicht nur Zeichen lesen, sondern auch Zeichen schreiben. Der Kopf steht wie bei endlichen Automaten immer auf genau einer Zelle und liest in jedem Rechenschritt das Zeichen, welches auf dieser Zelle geschrieben steht. Im Gegensatz zu endlichen Automaten kann der Kopf aber auch Zeichen schreiben; er tut dies in jedem Rechenschritt. Dabei wird das aktuelle Zeichen der Zelle durch ein neues Zeichen ersetzt. Neues und altes Zeichen dürfen aber auch identisch sein.
- Wir haben nicht nur ein Eingabealphabet Σ , sondern auch ein Bandalphabet Γ . Jede Zelle des Bands ist mit einem Zeichen des Bandalphabetes beschriftet. Wir haben in unserem Bandalphabet ein spezielles Zeichen $\sqcup \in \Gamma$, das wir *Blank* nennen und welches ein Feld als „leer“ markiert. Zu Beginn sind alle Zellen, auf denen keine Eingabe steht, mit einem Blank-Zeichen beschriftet.
- Der Kopf der TM kann nicht nur nach rechts bewegt werden, er darf auch nach links bewegt werden oder seine Position beibehalten.

Wie bei endlichen Automaten steht der Kopf zu Beginn auf dem ersten (am weitesten links stehenden) Zeichen der Eingabe.

Wir geben die Transitionsfunktion einer Turingmaschine mit Hilfe einer Tabelle, der sogenannten *Turingtabelle* an.

$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
q	a	q'	a'	d
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Die Bedeutung eines Tabelleneintrags ist dabei: Wenn die TM in Zustand q ist und der Kopf liest gerade das Symbol $a \in \Gamma$, dann wechsele in den Zustand q' , schreibe a' (über altes a) und bewege den Kopf gemäß $d \in \{R, L, N\}$.

Eine Turingmaschine arbeitet schrittweise. In jedem Rechenschritt wird eine Zeile der Turingtabelle angewendet. Bei nichtdeterministischen Turingmaschinen kann es Situationen geben, bei denen mehrere Zeilen der Turingtabelle angewendet werden können, bei deterministischen Turingmaschinen kann immer höchstes ein Tabelleneintrag angewendet werden. In beiden Fällen (deterministisch/nichtdeterministisch) darf es auch den Fall geben, dass *keine* Zeile angewendet werden kann. In solch einem Fall hält die Turingmaschine an. Nachdem die Turingmaschine angehalten hat, prüfen wir, ob der aktuelle Zustand ein akzeptierender Zustand ist. Falls ja, wird die Eingabe akzeptiert, falls nein, wird die Eingabe verworfen.

Wurde eine Eingabe nicht akzeptiert, kann dies also zwei Ursachen haben:

1. Die Turingmaschine war nach dem Anhalten nicht in einem akzeptierenden Zustand.
2. Die Turingmaschine hat niemals angehalten.

Wir werden deterministische Turingmaschinen nicht nur verwenden, um *Sprachen* zu definieren (Akzeptanz), sondern auch um (*partielle*) *Funktionen* vom Typ $\Sigma^* \rightarrow \Sigma^*$ zu definieren. Das Funktionsargument ist dabei das Wort, welches zu Beginn auf dem Band steht. Das Resultat ist das Wort, welches am Ende auf dem Band steht. Um sicherzustellen, dass das Resultat ein Element von Σ^* ist, projizieren wir den Inhalt jeder Zelle nach dem Halten auf Σ .

Da die TM in endlich vielen Schritten nur endlich viele Zeichen schreiben kann, ist klar, dass das Resultat nach dem Halten ein endliches Wort ist. Hält die TM nicht, ist der Funktionswert für die entsprechende Eingabe undefiniert (daher beschreiben Turingmaschinen im Allgemeinen nur *partielle* Funktionen).

Bsp. 6.1: Aufgabe: Verschiebe das erste Zeichen der Eingabe an das Ende der Eingabe.

			□	b							
	□	□	b	a	n	a	n	e	u	□	□

Diese Aufgabe wird von einer TM gelöst die für jedes Symbol x des Eingabealphabetes Σ einen eigenen Zustand q_x und die folgende Turingtabelle hat.

q_0	x	q_x	$\sqcup$	R	$x \neq \sqcup$
q_x	$\sqcup$	q_3	x	L	
q_x	y	q_x	y	R	$y \neq \sqcup$
q_3	y	q_3	y	L	$y \neq \sqcup$
q_3	$\sqcup$	q_4	$\sqcup$	R	

Sie finden zwei weitere Beispiele für Turingmaschinen in den Folien, die auf der Vorlesungswebseite verlinkt sind.

<https://swt.informatik.uni-freiburg.de/teaching/WS2018-19/info3>

6.2 Formalisierung der Turing-Maschine

Def. 6.1: Eine *Turing-Maschine* (TM) ist ein 7-Tupel

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

mit den folgenden Komponenten:

- Σ ist ein Alphabet, das wir *Eingabealphabet* nennen,
- Q ist eine endliche Menge, deren Elemente wir *Zustände* nennen,

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

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

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

Die *Konfiguration* einer Turingmaschine wird durch drei Dinge beschrieben:

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

Die scheinbar „natürlichste“ Beschreibung der Konfiguration wäre also ein Tripel aus $Q \times (\mathbb{Z} \rightarrow \Gamma) \times \mathbb{Z}$. Dies würde aber erfordern, dass wir jede Zelle des Bands mit einer ganzen Zahl identifizieren und wir müssten uns eine endliche Repräsentation für die Abbildung $\mathbb{Z} \rightarrow \Gamma$ überlegen. Wir verwenden stattdessen den folgenden „Trick“ und beschreiben eine Konfiguration durch ein Tripel (v, q, w) . Dabei beschreibt

- v ein endliches Suffix der Bandhälfte links des Kopfs, das alle nicht-Blank-Zeichen (dieser Hälfte) enthält,
- q den aktuellen Zustand und
- w ein endliches Präfix der verbleibenden Bandhälfte (also unter Kopf und rechts davon), das alle nicht-Blank-Zeichen (dieser Hälfte) enthält.

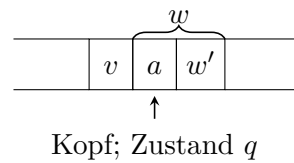


Abb. 11: Informelle graphische Darstellung einer Turingmaschinenkonfiguration

Def. 6.2: Die Menge der *Konfigurationen* einer TM ist $\text{Konf}(\mathcal{M}) = \Gamma^* \times Q \times \Gamma^+$. Die *Startkonfiguration* bei Eingabe w ist $(\varepsilon, q^{\text{init}}, w)$ für $w \neq \varepsilon$ und $(\varepsilon, q^{\text{init}}, \sqcup)$ für $w = \varepsilon$. Eine *Haltekonfiguration* ist eine Konfiguration (v, q, aw) , bei der $\delta(q, a) = \emptyset$ gilt. $\oplus$

Notation: Wenn aus dem Kontext klar wird, welcher Buchstabe zu welcher Komponente des Tripels gehört, dürfen wir die Klammern und Kommata auch weglassen. Wir schreiben z.B. die Startkonfiguration als $q^{\text{init}}w$.

Def. 6.3: Die *Rechenschrittrelation*

$$\vdash \subseteq \text{Konf}(\mathcal{M}) \times \text{Konf}(\mathcal{M})$$

ist definiert durch:

1. $vqaw \vdash vq'a'w$ falls $\delta(q, a) = (q', a', N)$
 2. $vqaw \vdash va'q'w$ falls $\delta(q, a) = (q', a', R), w \neq \varepsilon$
 $va'q' \sqcup \quad \quad \quad \text{"} \quad \quad \quad \text{---}, w = \varepsilon$
 3. $qaw \vdash q' \sqcup a'w$ falls $\delta(q, a) = (q', a', L)$
 $vbqaw \vdash vq'ba'w \quad \text{---} \quad \text{"} \quad \text{---} \quad b \in \Gamma$
- $\oplus$

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

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

Def. 6.4 (Akzeptanz): Wir sagen $\mathcal{M}$ *akzeptiert* das Wort $w \in \Sigma^*$, wenn $u, v \in \Gamma^*$ und $q \in Q$ existieren sodass die folgenden drei Eigenschaften gelten.

1. $q^{\text{init}}w \vdash^* uqv$
2. uqv Haltekonfiguration
3. $q \in F$

Die von $\mathcal{M}$ akzeptierte Sprache $L(\mathcal{M})$ ist definiert als $\{w \in \Sigma^* \mid \mathcal{M} \text{ akzeptiert } w\}$. $\oplus$

Def. 6.5: Eine DTM $\mathcal{M}$ *hält* auf einer Eingabe w , falls es eine Haltekonfiguration uqv gibt, sodass $q^{\text{init}}w \vdash^* uqv$ gilt. $\oplus$

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

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

Def. 6.6 (berechnete Funktion): Sei $\mathcal{M}$ eine DTM. Dann ist die von $\mathcal{M}$ *berechnete partielle[†] Funktion* $f_{\mathcal{M}} : \Sigma^* \rightharpoonup \Sigma^*$ wie folgt definiert.

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

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

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

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

6.3 TM-Programmierung mit Hilfe von Flussdiagrammen

Auch für einfache Turingmaschinen können die Turingtabellen sehr groß werden. Wir führen deshalb einen Formalismus ein, der es uns erlaubt, mehrere kleine Turingmaschinen zu einer großen zusammenzubauen.

Idee: Wir nehmen einen gerichteten Graphen und beschriften dessen Knoten mit Turingmaschinen und dessen Kanten mit einer Teilmenge des Bandalphabetes. Eine Kante $\mathcal{M}_1 \xrightarrow{\{a,b,c\}} \mathcal{M}_2$ bedeutet dann: Wenn die TM $\mathcal{M}_1$ hält und a , b oder c unter dem Kopf steht, dann kann TM $\mathcal{M}_2$ beginnend mit ihren Startzustand auf dem aktuellen Bandinhalt weitermachen. Voraussetzung ist, dass alle TMs das gleiche Bandalphabet und das gleiche Blanksymbol verwenden.

Bsp. 6.2: Wir definieren uns zunächst für beliebiges Σ und Γ drei sehr einfache Turingmaschinen.

- 1-Schritt-Rechtsmaschine $\mathcal{M}_r$:

Geht für beliebigen Bandinhalt einen Schritt nach rechts und hält dann.

$$\mathcal{M}_r = (\Sigma, \{q_0, q_1\}, \Gamma, \delta, q_0, \sqcup, \{\}) \text{ mit } \delta(q, x) = \begin{cases} \{(q_1, x, R)\} & \text{falls } q = q_0 \\ \{\} & \text{falls } q = q_1 \end{cases}$$

[†]Im Gegensatz zu einer Funktion muss eine partielle Funktion nicht für alle Elemente des Urbilds definiert sein. Ist f keine Funktion, sondern nur eine partielle Funktion von X nach Y , so schreiben wir $f : X \rightharpoonup Y$ statt $f : X \rightarrow Y$.

- 1-Schritt-Linksmaschine $\mathcal{M}_l$:

Geht für beliebigen Bandinhalt einen Schritt nach links und hält dann.

$$\mathcal{M}_l = (\Sigma, \{q_0, q_1\}, \Gamma, \delta, q_0, \sqcup, \{\}) \text{ mit } \delta(q, x) = \begin{cases} \{(q_1, x, L)\} & \text{falls } q = q_0 \\ \{\} & \text{falls } q = q_1 \end{cases}$$

- Für jedes Zeichen $y \in \Gamma$ die Druckmaschine $\mathcal{D}_y$:

Schreibt das Zeichen y in die Zelle, über der sich der Kopf befindet, und hält dann.

$$\mathcal{D}_y = (\Sigma, \{q_0, q_1\}, \Gamma, \delta, q_0, \sqcup, \{\}) \text{ mit } \delta(q, x) = \begin{cases} \{(q_1, y, N)\} & \text{falls } q = q_0 \\ \{\} & \text{falls } q = q_1 \end{cases}$$

Bsp. 6.3: Mit Hilfe von Flussdiagrammen definieren wir uns nun zwei weitere Turingmaschinen.

- Rechtsmaschine $\mathcal{M}_R$:

Geht für beliebigen Bandinhalt nach rechts, bis ein Blanksymbol erreicht ist, und hält dann. Es wird mindestens ein Schritt gemacht.

$$\rightarrow \mathcal{M}_r \hookrightarrow \Gamma \setminus \{\sqcup\}$$

- Linksmaschine $\mathcal{M}_L$:

Geht für beliebigen Bandinhalt nach links, bis ein Blanksymbol erreicht ist, und hält dann. Es wird mindestens ein Schritt gemacht.

$$\rightarrow \mathcal{M}_l \hookrightarrow \Gamma \setminus \{\sqcup\}$$

Wir definieren das Flussdiagramm nun formal wie folgt.

Def. 6.7: Ein *Flussdiagramm* ist ein 8-Tupel $G = (\Sigma, \Gamma, \sqcup, V, E, v^{\text{init}}, L_V, L_E)$. Dabei ist

- Σ ein Alphabet, das wir *Eingabealphabet* nennen,
- $\Gamma \supsetneq \Sigma$ ein Alphabet, das wir *gemeinsames Bandalphabet* nennen,
- $\sqcup \in \Gamma \setminus \Sigma$ ein Zeichen, das wir *gemeinsames Blanksymbol* nennen,
- (V, E) ein gerichteter Graph,
- $v^{\text{init}} \in V$ ein Knoten, den wir *Startknoten* nennen,
- L_V eine Abbildung, die jedem Knoten $v \in V$ eine Turingmaschine zuordnet und
- $L_E : E \rightarrow \mathcal{P}(\Gamma)$ eine Abbildung, die jeder Kante $(v, v') \in E$ eine Teilmenge des Bandalphabetes zuordnet. $\oplus$

Notation: Bis zum Ende des Kapitels schreiben wir $\mathcal{M}_v = (\Sigma, Q_v, \Gamma, \delta_v, q_v^{\text{init}}, \sqcup, F_v)$ für die Turingmaschine $L_V(v)$, also die Turingmaschine, die dem Knoten v zugeordnet ist.

Def. 6.8: Die von einem Flussdiagramm $G = (\Sigma, \Gamma, \sqcup, V, E, v^{\text{init}}, L_V, L_E)$ definierte Turingmaschine ist $\mathcal{M} = (\Sigma, Q, \Gamma, \delta, q^{\text{init}}, \sqcup, F)$, wobei

- $Q = \dot{\bigcup}_{v \in V} Q_v$ [†]
- $q^{\text{init}} = q_{v^{\text{init}}}^{\text{init}}$
- $\delta(q, x) = \begin{cases} \delta_v(q, x) & \text{falls } q \in Q_v \text{ und } \delta_v(q, x) \neq \emptyset \\ \left\{ (q'', x, N) \mid \begin{array}{l} q \in Q_v, (v, v') \in E, \\ x \in L_E((v, v')), q'' = q_{v'}^{\text{init}} \end{array} \right\} & \text{sonst} \end{cases}$
- $F = \dot{\bigcup}_{v \in V} F_v.$ ⊕

Notation: Analog zu Startzuständen von endlichen Automaten verwenden wir einen eingehenden Pfeil, um den Startknoten eines Flussdiagramms zu kennzeichnen.

Bsp. 6.4: Wir wollen eine TM konstruieren, welche die folgende Subtraktion von natürlichen Zahlen implementiert.

Vorlesung:
11.01.2019

$$f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \quad f(x, y) = \begin{cases} x - y & \text{falls } x \geq y \\ 0 & \text{sonst} \end{cases}$$

Wir verwenden dabei eine auf Strichsymbolen basierende Unärcodierung ^{††} für die Operanden und benutzen das Zeichen $\#$ um die Operanden x und y in der Eingabe zu trennen. Wir haben also $\Sigma = \{I, \#\}$ und für Minuend x und Subtrahend y hat die Eingabe die folgende Form.

$$\underbrace{I \dots I}_{x \text{ Mal}} \# \underbrace{I \dots I}_{y \text{ Mal}}$$

Idee: Lösche für jeden y -Strich einen x -Strich. Lösche falls nötig (Fall $y > x$) verbleibende y -Striche. Lösche anschließend das Trennzeichen $\#$. Wir verwenden die übliche Notation für das Blanksymbol, benötigen keine weiteren Zeichen für unsere Berechnung und verwenden daher das Bandalphabet $\Gamma := \Sigma \cup \{\sqcup\}$.

[†]Um eine disjunkte Vereinigung zu erreichen, müssen wir ggf. Zustände umbenennen.

^{††}Unärcodierungen von natürlichen Zahlen wurden in Aufgabe 1 von Übungsblatt 8 eingeführt.

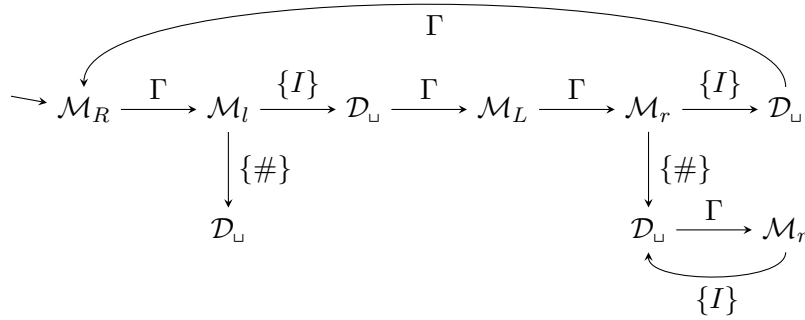


Abb. 12: Durch Flussdiagramm beschriebene TM

Das Flussdiagramm aus Abb. 12 implementiert diese Idee. Die obere Schleife beschreibt das simultane Entfernen von Strichen in x und y . Die erste Abzweigung nach unten wird genommen, wenn alle Striche von y entfernt wurden – also ursprünglich y kleiner ($\leq$) als x war. Die untere Schleife wird ausgeführt, wenn y echt größer als x war; sie dient dazu, die verbleibenden Zeichen auf dem Band zu löschen, um die unärcodierte 0 (also das leere Wort) zu erzeugen.

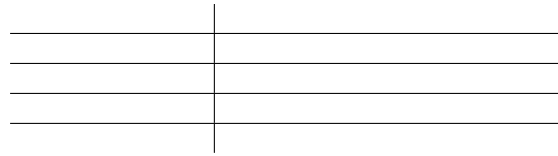
6.4 Varianten von TMs

Es gibt viele Varianten von Turingmaschinen. Diese sind typischerweise äquivalent zu Def. 6.1 in dem Sinne, dass sie die gleiche Klasse von Sprachen akzeptieren und die gleiche Menge von Funktionen berechnen können. Welche Variante am bequemsten zu benutzen ist, hängt von der Anwendung ab. Wir stellen in diesem Unterkapitel zwei alternative Varianten von TMs vor. Die Beschreibungen der Maschinen, die Behauptungen und zugehörigen Beweisskizzen sind in diesem Unterkapitel eher informell gehalten. Wir ermutigen die Leserschaft darüber nachzudenken wie man die Inhalte analog zu den vorigen Kapiteln formal präsentieren könnte.

6.4.1 Mehrspurmaschinen

Eine k -Spur-Turingmaschine arbeitet wie eine (gewöhnliche) TM, aber das Band hat hier k Spuren. Der Kopf liest und schreibt also immer einen k -Vektor von Zeichen. Die Transitionsfunktion hat den Typ $\delta : Q \times \Gamma^k \longrightarrow \mathcal{P}(Q \times \Gamma^k \times \{R, L, N\})$.

Die Eingabe steht zu Beginn auf der ersten Spur. Zum Ermitteln der Ausgabe betrachten wir auch nur die erste Spur.

**Abb. 13:** Mehrspurmaschine

Bsp.: binäre Addition und Multiplikation nach der Schulmethode

Bemerkung: Eine k -Spur-TM kann von einer (gewöhnlichen) TM simuliert werden.

Idee: Verwende Bandalphabet $\Gamma_{\text{sim}} = \Sigma \dot{\cup} (\Gamma^k \times \{\sqcup, B\})$ und Blanksymbol $\sqcup_{\text{sim}} = \sqcup^{k+1}$. Hierbei steht B für „besucht“.

1. Konvertieren der Eingabe:

Laufe einmal über die Eingabe und ersetze $a \in \Sigma$ durch $\begin{pmatrix} a \\ \sqcup \\ \vdots \\ \sqcup \end{pmatrix} \in \Gamma_{\text{sim}}$. Bewege den

Kopf zurück in die Ausgangsposition (also die am weitesten links liegende Zelle, die verschieden von $\sqcup_{\text{sim}}$ ist).

2. Simulation:

Exaktes Nachahmen jedes Rechenschritts der k -Spur-TM, bis diese anhält. Schreibe dabei als $k + 1$ -te Komponente des Tupels immer ein B .

3. Konvertieren der Ausgabe:

Idee: Ersetze jedes $\begin{pmatrix} a_1 \\ \vdots \\ a_k \\ x \end{pmatrix} \in \Gamma_{\text{sim}}$ durch a_1 , falls $a_1 \in \Sigma$, und sonst durch $\begin{pmatrix} \sqcup \\ \sqcup \\ \vdots \\ \sqcup \end{pmatrix}$.

(Zur Ausgabe gehören schließlich nur die Elemente aus Σ .)

Problem: Woher wissen wir, wie weit wir nach rechts laufen müssen?

Lösung: Wir achten auf die letzte Komponente des Tupels. Wir laufen so lange nach rechts, wie dort ein B steht, denn alle diese Zellen haben wir mindestens einmal besucht.

6.4.2 Mehrbandmaschinen

Eine k -Band-TM hat k Bänder, die jeweils einen eigenen Kopf haben. In jedem Schritt wird jeder Kopf lesen, schreiben und sich bewegen. Dabei darf sich jeder Kopf in eine andere Richtung bewegen.

Die Transitionsfunktion hat den Typ $\delta : Q \times \Gamma^k \longrightarrow \mathcal{P}(Q \times \Gamma^k \times \{R, L, N\}^k)$.

Bemerkung: Eine k -Band-TM $\mathcal{M}$ kann von einer $2k + 1$ -Spur-TM $\mathcal{M}_{\text{sim}}$ simuliert werden.

Idee: Codiere die Konfiguration einer k -Band-TM mit Hilfe von $2k$ Spuren. Simuliere einen Schritt der k -Band-TM in mehreren Schritten. Verwende dabei eine zusätzliche Spur, um das bisher bearbeitete Band zu kennzeichnen. Der Zustand von $\mathcal{M}$ wird im Zustand von $\mathcal{M}_{\text{sim}}$ codiert. Das Bandalphabet der k -Spur-TM sei $\Gamma_{\text{sim}} = \Gamma \dot{\cup} \{\#, B\}$.

- Spur $2i - 1$ simuliert Band i .
- Spur $2i$ ist leer bis auf einen Marker $\#$, welcher die Position des Kopfs auf Band i markiert.
- Spur $2k + 1$ enthält das Zeichen B an jeder Position, die bereits bearbeitet wurde.

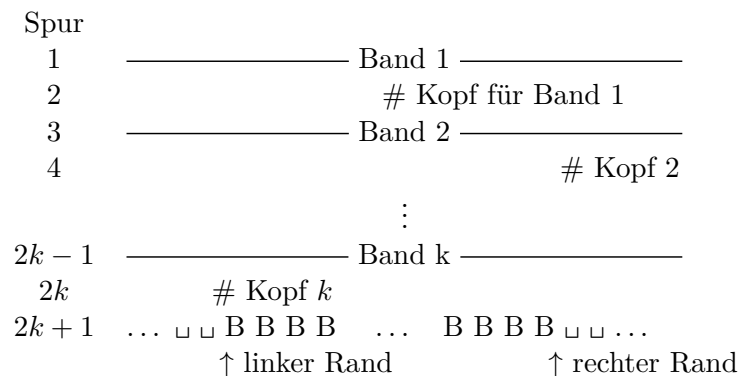


Abb. 14: Simulation einer Mehrbandmaschine durch eine Mehrspurmaschine

Die Mehrspurmaschine arbeitet dabei in zwei Phasen.

1. Herstellen der Start-Konfiguration:

- Schreibe Kopfmarkierung (Zeichen $\#$) auf allen geraden Spuren und B auf Spur $2k + 1$.

Spur 1	$\dots \sqcup a_1 a_2 \dots a_n \sqcup \dots$	(= Eingabe)
2	$\dots \sqcup \# \sqcup \dots$	
3	$\dots \sqcup \sqcup \sqcup \dots$	
4	$\dots \sqcup \# \sqcup \dots$	
$\vdots$		
2k-1	$\dots \sqcup \sqcup \sqcup \dots$	
2k	$\dots \sqcup \# \sqcup \dots$	
2k+1	$\dots \sqcup B \sqcup \dots$	

Abb. 15: Startkonfiguration einer Mehrbandmaschine codiert auf Mehrspurmaschine

2. Simulation:

Der Kopf der Mehrspurmaschine startet jeweils bei der linken Begrenzung, also beim am weitesten links stehenden B auf Spur $2k + 1$.

- Laufe bis zum rechten Rand. Sammle dabei die Symbole unter den Köpfen (markiert mit $\#$) und merke dir diese im Zustand (Vektorcodierung $\vec{\gamma} \in \Gamma^k$).
- Bestimme Resultat von Transitionsfunktion $\delta(q, \vec{\gamma}) = (q', \vec{\gamma}', \vec{d})$ neuer Zustand q' , für jeden Kopf ein neues Symbol $\vec{\gamma}'$ und Richtung $\vec{d}$.
- Laufe zurück nach links. Schreibe dabei das jeweilige Zeichen $\vec{\gamma}'$ am entsprechenden Kopf und versetze diesen gemäß $\vec{d}$.

Falls eine Kopfbewegung den Rand auf Spur $2k + 1$ überschreitet, dann schreibe ein weiteres B .

Beim Rücklauf: Teste auf Haltekonfiguration der k -Band-TM. Falls ja, so springe in eine Haltekonfiguration der k -Spur-TM.

Bemerkung: Angenommen, für ein Wort der Länge n benötige die k -Band-TM $\mathcal{M} T(n)$ Schritte und $S(n)$ Zellen auf den Bändern. Dann benötigt $\mathcal{M}_{\text{sim}}$ höchstens $O(S(n))$ Zellen und $O(S(n) \cdot T(n)) = O(T(n)^2)$ Schritte.

6.5 Universelle Turingmaschine

Die bisher betrachteten Turingmaschinen waren (ebenso wie endliche Automaten oder Kellerautomaten) auf einen Einsatzzweck beschränkt und konnten nur eine bestimmte Sprache akzeptieren oder eine bestimmte Funktion berechnen. Dies entspricht nicht unserer Vorstellung eines Computers, denn dieser kann normalerweise beliebige Programme ausführen.

Vorlesung:
16.01.2019

In diesem Kapitel konstruieren wir nun eine TM $\mathcal{M}_U$, die als Eingabe sowohl eine Codierung $\lceil \mathcal{M} \rceil$ einer beliebigen TM $\mathcal{M}$ als auch deren Eingabe w nimmt, sodass die folgenden drei Eigenschaften gelten.

$$w \in L(\mathcal{M}) \Leftrightarrow \lceil \mathcal{M} \rceil \# w \in L(\mathcal{M}_U) \quad (5)$$

$$\mathcal{M} \text{ hält auf Eingabe } w \Leftrightarrow \mathcal{M}_U \text{ hält auf Eingabe } \lceil \mathcal{M} \rceil \# w \quad (6)$$

$$f_{\mathcal{M}}(w) = f_{\mathcal{M}_U}(\lceil \mathcal{M} \rceil \# w) \quad (7)$$

Wir nennen $\mathcal{M}_U$ die *universelle Turingmaschine*.

6.5.1 Codierung von Turingmaschinen

Zunächst wollen wir uns eine Codierung $\lceil \mathcal{M} \rceil$ für $\mathcal{M} = (\Sigma, Q, \Gamma, \delta, q^{\text{init}}, \sqcup, F)$ überlegen. Die Effizienz unserer Codierung ist uns dabei nicht wichtig und wir versuchen der Einfachheit wegen mit Unärcodierungen zu arbeiten.

Uns ist nicht wichtig dass alle Details von $\mathcal{M}$ codieren uns ist nur wichtig, dass wir eine TM mit dem Verhalten von $\mathcal{M}$ codieren. Deshalb können wir uns mit den folgenden Überlegungen die Codierung vereinfachen.

- Da Q und Γ endlich sind, können wir jedem ihrer Elemente eine natürliche Zahl zuordnen und in der Codierung dann nur diese Zahl verwenden.
- Wir müssen q^{init} und $\sqcup$ nicht explizit codieren; wir verwenden die Konvention, dass q^{init} der Zustand mit der Nummer 1 und $\sqcup$ das Zeichen mit der Nummer 1 ist.
- Wir müssen Σ , Q und Γ nicht explizit in die Codierung mit aufnehmen. Für das Ausführen einer TM sind nur die Elemente aus Σ , Q und Γ relevant, die auch in der Transitionsfunktion δ vorkommen.

Wir möchten einer Konvention und der darauf folgenden Definition dafür sorgen dass wir die obigen Überlegungen umsetzen können. [†]

Konvention: Wir wollen von nun an ein Alphabet $\Sigma = \{a_1, \dots, a_{n_\Sigma}\}$ festhalten. Somit ist immer wenn wir von der Folge $a_1, \dots, a_{n_\Sigma}$ sprechen die gleiche Reihenfolge der Symbole gemeint.

Seien $q_1, q_2, \dots$ und $b_1, b_2, \dots$ zwei unendlich lange Folgen die jeweils für jede positive natürliche Zahl ein Element enthalten. In der folgenden Definition werden wir endliche Präfixe dieser Folgen als Zustände bzw. Symbole des Bandalphabetes verwenden.

[†]In der Vorlesung vom 16.01.2019 gab es Fragen zur Umsetzung einer eindeutigen Codierung von TMs. Ich habe deshalb diesen grünen Text nachträglich hinzugefügt und die Definition der *normierten TM* in der Vorlesung vom 18.01.2019 vorgestellt.

Def. 6.9: Wir nennen eine TM $\mathcal{M} = (\Sigma, Q, \Gamma, \delta, q^{\text{init}}, \sqcup, F)$ *normiert* falls

- $Q = \{q_1, \dots, q_{|Q|}\}$
- $\Gamma = \{b_1, \dots, b_{|\Gamma|-|\Sigma|}, a_1, \dots, a_{n_\Sigma}\}$
- $q^{\text{init}} = q_1$
- $\sqcup = b_1$
- Für alle $q \in Q$ existiert ein $x \in \Gamma$ sodass $\delta(q, x) \neq \{\}^\dagger$
- Für alle $x \in \Gamma \setminus (\Sigma \cup \{\sqcup\})$ existiert ein $q \in Q$ sodass $\delta(q, x) \neq \{\}^\ddagger$

Wir schreiben $\mathbb{M}$ für die Menge aller normierten TMs.

$\oplus$

Lemma 6.1: Für Zu jeder TM $\mathcal{M}$ existiert eine normierte TM $\mathcal{M}'$ dass

- $f_{\mathcal{M}'} = f_{\mathcal{M}}$
- $L(\mathcal{M}') = L(\mathcal{M})$

BEWEIS: Umbenennen und Löschen von Zuständen und von Symbolen des Bandalphabetes. $\square$

Idee: Codiere Elemente aus Q und Γ unär und verwende statt eines Strichs das Zeichen 0. Verwende das Zeichen 1, um unärcodierte Zahlen zu trennen.

- Zustände:

Seien $q_1, \dots, q_n$ die Elemente von Q , sodass $q_1 = q^{\text{init}}$.

Definiere $\lceil q_i \rceil := 0^i$.

- Menge der akzeptierenden Zustände:

Seien $q_{k_1}, \dots, q_{k_n}$ die Elemente von F , in der Reihenfolge sodass $k_1 < \dots < k_n$ gilt.

Definiere $\lceil F \rceil := \lceil q_{k_1} \rceil 1 \lceil q_{k_2} \rceil 1 \dots 1 \lceil q_{k_n} \rceil$.

- Bandalphabet:

Seien $b_1, \dots, b_n$ die Elemente von Γ , sodass $b_1 = \sqcup$.

Definiere $\lceil b_j \rceil := 0^j$.

[†] Anschaulich: Jeder Zustand kommt mindestens ein mal in der Turingtabelle vor.

^{††} Anschaulich: Jedes Symbol des Bandalphabetes ist entweder das Blank, Teil der Eingabealphabetes oder es kommt mindestens ein mal in der Turingtabelle vor.

- Richtung (in die der Schreib-Lesekopf bewegt wird):

Definiere $\ulcorner L \urcorner := 0$, $\ulcorner N \urcorner := 00$ und $\ulcorner R \urcorner := 000$.

- Transitionsfunktion:

Seien $(q_{t_1}, b_{t_1}, q'_{t_1}, b'_{t_1}, d_{t_1}), \dots, (q_{t_n}, b_{t_n}, q'_{t_n}, b'_{t_n}, d_{t_n})$ die Elemente von δ , die nicht auf $\{\}$ abgebildet werden. Diese Elemente seien durch eine lexikographische Sortierung gegeben

Definiere

$$\ulcorner \delta \urcorner := \ulcorner q_{t_1} \urcorner 1 \ulcorner a_{t_1} \urcorner 1 \ulcorner q'_{t_1} \urcorner 1 \ulcorner a'_{t_1} \urcorner 1 \ulcorner d_{t_1} \urcorner 11 \dots 11 \ulcorner q_{t_n} \urcorner 1 \ulcorner a_{t_n} \urcorner 1 \ulcorner q'_{t_n} \urcorner 1 \ulcorner a'_{t_n} \urcorner 1 \ulcorner d_{t_n} \urcorner.$$

- Turingmaschine:

$$\ulcorner \mathcal{M} \urcorner := 111 \ulcorner \delta \urcorner 111 \ulcorner F \urcorner 111$$

Bsp. 6.5: Wir bestimmen nun die Codierung $\ulcorner \mathcal{M}_r \urcorner$ der 1-Schritt-Rechtsmaschine aus Bsp. 6.2 für $\Sigma = \{0, 1\}$ und $\Gamma = \{0, 1, \sqcup\}$. Wir nehmen dabei für die Elemente von Q , Γ und δ die folgenden Reihenfolgen an.

Q	q_0, q_1
Γ	$\sqcup, 0, 1$
δ	$(q_0, \sqcup, q_1, \sqcup, R), (q_0, 0, q_1, 0, R), (q_0, 1, q_1, 1, R)$

Wir erhalten:

- Zustände: $\ulcorner q_0 \urcorner = 0$, $\ulcorner q_1 \urcorner = 00$
- Menge der akzeptierenden Zustände: $\ulcorner F \urcorner = \varepsilon$ (da $F = \{\}$)
- Bandalphabet: $\ulcorner \sqcup \urcorner = 0$, $\ulcorner 0 \urcorner = 00$, $\ulcorner 1 \urcorner = 000$
- Turingmaschine: $\ulcorner \mathcal{M}_r \urcorner =$

$$111 \overbrace{010100101000}^{\text{erste Transition}} 11 \overbrace{01001001001000}^{\text{zweite Transition}} 11 \overbrace{0100010010001000}^{\text{dritte Transition}} 111 \underbrace{\hspace{1cm}}_{\ulcorner F \urcorner} 111$$

6.5.2 Arbeitsweise der universellen Turingmaschine

Da die universelle Turingmaschine $\mathcal{M}_U$ recht komplex ist, wollen wir diese hier nicht formal definieren, sondern geben nur eine informelle Beschreibung der Arbeitsweise an.

Wir konstruieren $\mathcal{M}_U$ als 3-Band-TM, wobei die Bänder B_1 , B_2 und B_3 wie folgt genutzt werden.

B_1 : Zu Beginn steht hier die Eingabe für $\mathcal{M}_U$. Nach der Initialisierung von $\mathcal{M}_U$ verwenden wir B_1 , um das Band der Eingabeturingmaschine $\mathcal{M}$ nachzuahmen.

B_2 : Speichere die Codierung der Eingabeturingmaschine $\lceil \mathcal{M} \rceil$.

B_3 : Speichere den aktuellen Zustand von $\mathcal{M}$ (0^k für Zustand q_k).

Die universelle Turingmaschine $\mathcal{M}_U$ beginnt zunächst mit einer Initialisierung, die aus den folgenden drei Teilen besteht.

1. Prüfe, ob der erste Teil der Eingabe, also $\lceil \mathcal{M} \rceil$, eine gültige Turingmaschine codiert.

Die Menge der gültigen Codierungen kann mit Hilfe einer kontextfreien Grammatik beschrieben werden (siehe Übungen). Die kontextfreie Grammatik kann in einen Kellerautomaten übersetzt werden (siehe Lemma 4.1) und

2. Verschiebe $\lceil \mathcal{M} \rceil$ auf B_2 . Schreibe dabei Blanksymbole auf die Bandzellen von B_1 . Überschreibe anschließend das Symbol $\#$, das die Eingabeturingmaschine und das Eingabewort trennt, durch ein Blanksymbol.
3. Schreibe 0 (also die Codierung des Startzustands von $\mathcal{M}$) auf B_3 .

Nach der Initialisierung ist $\mathcal{M}_U$ also in der folgenden Konfiguration.

$$\begin{array}{ll} B_1 : & w \\ B_2 : & \lceil \mathcal{M} \rceil \\ B_3 : & 0 \end{array}$$

Nach der Initialisierung wird $\mathcal{M}_U$ das Verhalten von $\mathcal{M}$ nachahmen. Für jeden Schritt von $\mathcal{M}$ macht $\mathcal{M}_U$ dabei die folgenden Schritte.

Suche ein Element (q, a, q', a', d) der Transitionsfunktion δ von $\mathcal{M}$, das zum aktuellen Zustand q und zum aktuellen Bandsymbol a passt.

Wir laufen hierfür einmal über das komplette Band B_2 . Zustände werden zeichenweise mit dem Inhalt von B_3 verglichen. Die Zuordnung von Alphabetsymbolen a zur Codierung $\lceil a \rceil$ ist fest in $\mathcal{M}_U$ eingebaut.

- Falls solch ein Element der Transitionsfunktion existiert, schreibe auf B_1 das Zeichen a' und bewege den Kopf wie durch d definiert. Ersetze außerdem die Codierung $\lceil q \rceil$ auf B_3 durch $\lceil q' \rceil$. Fahre anschließend mit dem Nachahmen des nächsten Schritts von $\mathcal{M}$ fort.
- Falls kein solches Element der Transitionsfunktion existiert, hat $\mathcal{M}$ eine Haltekonfiguration erreicht. Wir vergleichen nun den aktuellen Zustand (auf B_3) mit den auf B_2 gespeicherten akzeptierenden Zuständen und akzeptieren oder verwerfen die Eingabe entsprechend.

Satz 6.2: Die universelle TM $\mathcal{M}_U$ erfüllt die Eigenschaften (5), (6) und (7).

7 Berechenbarkeit

In diesem Kapitel werden wir zeigen, dass es Sprachen gibt, die von keiner TM akzeptiert werden können, und dass es Funktionen gibt, die von keiner TM berechnet werden können.

7.1 Church-Turing These

Def. 7.1: Eine (partielle) Funktion $f : \Sigma^* \rightarrow \Sigma^*$ heißt *berechenbar*, wenn es eine Turingmaschine $\mathcal{M}$ gibt, sodass f die von $\mathcal{M}$ berechnete Funktion ist. $\oplus$

These (Church-Turing These): Jede intuitiv berechenbare Funktion ist mit einer TM (in formalem Sinn) berechenbar.

Diese nach Alonzo Church[†] und Alan Turing benannte These vergleicht zwei Begriffe von denen nur einer formal definiert ist. Wir können diese These daher weder beweisen noch formal widerlegen.

Unter dem Begriff der “intuitiv berechenbaren Funktion” versteht man aber typischerweise eine Funktion die von einem Algorithmus berechnet wird und unter einem *Algorithmus* versteht man eine präzise mit endlichen vielen Zeichen beschriebene Handlungsvorschrift bei der jeder Schritt effektiv durchgeführt werden kann. Man könnte also versuchen ein Rechenverfahren zu finden dass allgemein als Algorithmus anerkannt ist aber nicht auf einer TM berechnet werden kann. Solch ein Rechenverfahren würde die Church-Turing These widerlegen bzw. aufzeigen wie der Begriff des Algorithmus präzisiert werden sollte. Bisher ist es nicht gelungen solch ein Rechenverfahren zu finden.

Bemerkung: In der Literatur wird in Def. 7.1 oft auch nicht der Begriff *berechenbar*, sondern die Begriffe *Turing-berechenbar* oder *rekursiv* verwendet. Die Intention dahinter ist, den Begriff *berechenbar* für „intuitiv berechenbar“ aufzusparen. Wir wollen uns hier aber ganz auf die Churchsche These verlassen und zur Vereinfachung den Begriff auch formal besetzen.

7.2 Nicht berechenbare Funktionen

Wir werden nun zeigen, dass es Funktionen gibt, die nicht berechenbar sind. Unser Vorgehen wird dabei das folgende sein.

- Wir zeigen, dass es so viele Turingmaschinen wie natürliche Zahlen gibt.

[†]Alonzo Church *1903 †1995

- Wir zeigen, dass es mehr Funktionen als natürliche Zahlen gibt.

Da es unendlich viele natürliche Zahlen, Turingmaschinen und Funktionen gibt, sind die aus der Schule bekannten Formalismen zum Vergleichen von Mengen (die Größe einer Menge ist die Anzahl ihrer Elemente) nicht anwendbar. Wir werden daher über die Existenz von bijektiven Abbildungen argumentieren.

Bemerkung: Zum Verständnis dieses Kapitels ist es nötig, sich mit den Begriffen *injektiv*, *surjektiv* und *bijektiv* vertraut zu machen, z.B. mit Hilfe von Aufgabe 1 von Vorbereitungsblatt 11.

Vorlesung:
18.01.2019

Wir führen zunächst die folgenden drei Begriffe ein.

Def. 7.2: Eine Menge X heißt

- *abzählbar unendlich*, wenn es eine bijektive Abbildung zwischen X und $\mathbb{N}$ gibt,
- *abzählbar*, wenn X entweder endlich oder abzählbar unendlich ist und
- *überabzählbar*, wenn X unendlich ist und es keine bijektive Abbildung zwischen X und $\mathbb{N}$ gibt. $\oplus$

Ein klassisches Vorgehen, um die Existenz nicht berechenbarer Funktionen zu zeigen, besteht aus den folgenden Schritten.

1. Die Menge der Turingmaschinen ist abzählbar.
2. Die Menge aller Funktionen ist überabzählbar.
3. Es kann daher nicht zu jeder Funktion auch eine Turingmaschine geben.

Wir wollen hier das folgende, leicht andere, Vorgehen wählen. Dieses ist weniger allgemein, erlaubt aber etwas leichter nachzuvollziehende Beweise.

1. Es gibt für jede Turingmaschine eine natürliche Zahl.
2. Es gibt für jedes Element der Potenzmenge der natürlichen Zahlen $\mathcal{P}(\mathbb{N})$ eine Funktion.
3. Die Potenzmenge $\mathcal{P}(\mathbb{N})$ hat „sehr viel mehr“ Elemente als $\mathbb{N}$. Es kann daher nicht zu jeder Funktion auch eine Turingmaschine geben.

Zur Vereinfachung werden wir uns in diesem Unterkapitel auf das Alphabet $\Sigma = \{0, 1\}$ beschränken.

Erinnerung: Die Funktion $\text{bin} : \Sigma^* \rightarrow \mathbb{N}$ ist die in Unterabschnitt 2.4 eingeführte Decodierung von Binärstrings (Wörtern über $\{0, 1\}$) in natürliche Zahlen.

Wir möchten zunächst eine bijektive Abbildung zwischen Binärstrings und natürlichen Zahlen definieren. Die Funktion bin ist hierfür ungeeignet, da führende Nullen ignoriert werden und z.B. sowohl 1 als auch 01 auf die 1 abgebildet werden.

Wir führen deshalb die *Standardnummerierung* ein, die auf den folgenden Ideen basiert.

- Nummeriere alle Binärstrings der Länge nach; bei gleicher Länge gibt die Funktion bin die Reihenfolge vor.
- Es gibt 2^n Binärstrings der Länge n .
- Für einen Binärstring der Länge n gibt es also $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ Binärstrings mit kürzerer Länge.

Lemma 7.1: Die Standardnummerierung, definiert als $\text{stdnum} : \Sigma^* \rightarrow \mathbb{N}$

$$\text{stdnum}(a_1 \dots a_n) = \begin{cases} 0 & \text{falls } a_1 \dots a_n = \varepsilon \\ 2^n - 1 + \text{bin}(a_1 \dots a_n) & \text{sonst} \end{cases},$$

ist eine bijektive Abbildung.

Beweis: Siehe Übungsblatt 12 Aufgabe 3a.

$w \in \{0, 1\}^*$	$\text{stdnum}(w)$	Berechnung
ε	0	
0	1	$2^0 + 0$
1	2	$2^0 + 1$
00	3	$2^0 + 2^1 + 0$
01	4	$2^0 + 2^1 + 1$
110	13	$2^0 + 2^1 + 2^2 + 6$
$\ulcorner \mathcal{M}_r \urcorner$	72057594037927935	$2^{55} - 1 + 32977455905055295$

Dabei ist $\ulcorner \mathcal{M}_r \urcorner$ die in Bsp. 6.5 bestimmte Codierung der 1-Schritt-Rechtsmaschine.

Abb. 16: Standardnummerierung für einige Beispielwerte.

Def. 7.3: Sei $w \in \{0, 1\}^*$. Die *durch w codierte TM* $\mathcal{M}_w$ ist wie folgt definiert.

$$\mathcal{M}_w = \begin{cases} \mathcal{M} & \text{falls } w \text{ Code einer TM ist und } \ulcorner \mathcal{M} \urcorner = w \\ \mathcal{D}_{\sqcup} & \text{sonst} \end{cases}$$

Dabei ist $\mathcal{D}_{\sqcup}$ die in Bsp. 6.2 definierte Druckmaschine.[†]

⊕

[†]Für den weiteren Verlauf ist es nicht wichtig, dass wir hier die Druckmaschine gewählt haben. Jede andere TM wäre ebenso gut geeignet gewesen.

Bemerkung: Im Folgenden genügt es, wenn wir uns auf TMs aus $\mathbb{M}$ einschränken. Gibt es eine TM, die eine Funktion f berechnet, so gibt es auch eine TM aus $\mathbb{M}$, die f berechnet. (Siehe Lemma 6.1)

Lemma 7.2: Die Abbildung $\text{bin2tm} : \Sigma^* \rightarrow \mathbb{M}$ von Binärstrings auf normierte TMs[†] definiert durch

$$\text{bin2tm}(w) = \mathcal{M}_w$$

ist surjektiv.

Lemma 7.3: Die Abbildung $\text{func2power} : (\Sigma^* \rightarrow \Sigma^*) \rightarrow \mathcal{P}(\mathbb{N})$ definiert durch

$$\text{func2power}(f) = \{\text{stdnum}(w) \mid f(w) = 1\}$$

ist surjektiv.

Beweis: Siehe Übungsblatt 12 Aufgabe 3b.

f	$\{w \mid f(w) = 1\}$	$\text{func2power}(f)$
Inkrement von Binärzahlen	$\{\varepsilon, 0\}$	$\{0, 1\}$
Identitätsfunktion	$\{1\}$	$\{2\}$
„alles auf eins“	Σ^*	$\mathbb{N}$

Abb. 17: Abbildung func2power für einige Beispielwerte

Satz 7.4 (Cantor): Es gibt keine surjektive Abbildung einer Menge in ihre Potenzmenge.

Den Beweis dieses Satzes werden wir später diskutieren.

Satz 7.5: Es gibt Funktionen, die nicht berechenbar sind.

BEWEIS: Angenommen, jede Funktion $f : \Sigma^* \rightarrow \Sigma^*$ sei berechenbar. Dann gibt es für jede Funktion f eine TM $\mathcal{M}$ mit $f_{\mathcal{M}} = f$ und somit eine surjektive Abbildung $g : \mathbb{M} \rightarrow (\Sigma^* \rightarrow \Sigma^*)$. Nach Lemma 7.2 und Lemma 7.3 wäre dann aber auch die Abbildung $\text{func2power} \circ g \circ \text{bin2tm} \circ \text{stdnum}^{-1} : \mathbb{N} \rightarrow \mathcal{P}(\mathbb{N})$ surjektiv. Dies steht im Widerspruch zu Satz 7.4. $\square$

[†]Siehe Def. 6.9

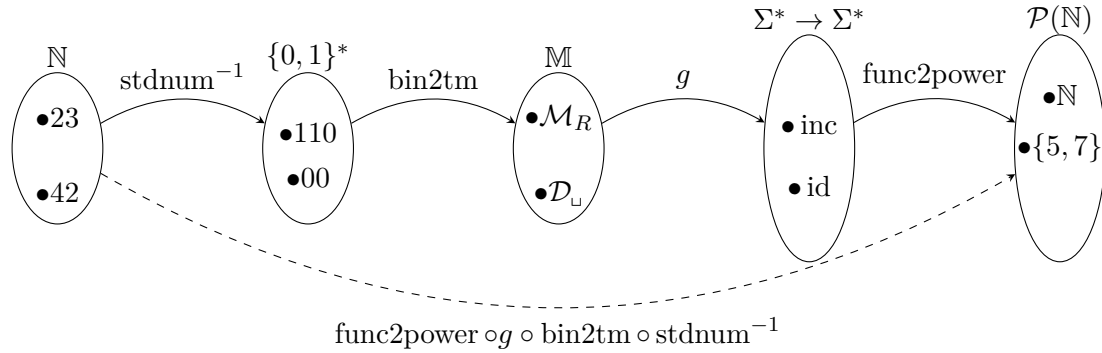


Abb. 18: Konkatenierte Abbildung aus dem Widerspruchsbeweis zu Satz 7.5

Ein zentraler und nicht offensichtlicher Teil dieses Beweises war Satz 7.4. Wir wollen diesen im restlichen Unterkapitel diskutieren. Zunächst geben wir einen formalen Beweis.

BEWEIS (Von Satz 7.4): Sei X eine beliebige Menge. Angenommen, es gäbe eine surjektive Abbildung $f : X \rightarrow \mathcal{P}(X)$. Betrachte die Menge $Z = \{x \in X \mid x \notin f(x)\} \in \mathcal{P}(X)$. Da f surjektiv ist, gibt es ein $z \in X$, sodass $f(z) = Z$. Nach Definition der Menge Z gilt nun aber $z \in f(z) \Leftrightarrow z \notin f(z)$. Widerspruch! $\square$

Ein Beweis dieser Form wird auch *Diagonalargument* genannt und ist nicht einfach zu verstehen. Wir werden nun den Beweis für den Spezialfall dass X eine abzählbar unendliche Menge ist illustrieren.

Da X abzählbar ist, können wir dessen Elemente als unendliche Sequenz $x_0, x_1, \dots$ aufschreiben. Wir nehmen nun an, es gäbe eine surjektive Abbildung $f : X \rightarrow \mathcal{P}(X)$. Dann können wir f mit Hilfe der folgenden Matrix darstellen. Zeile i beschreibt dabei das Bild $f(x_i)$. Wir haben eine Spalte für jedes $x_j \in X$. Gilt $x_j \in f(x_i)$, so steht in Eintrag (i, j) ein Y . Ansonsten steht dort ein N .

Da f surjektiv ist, muss es auch für die im formalen Beweis oben erwähnte Menge $Z = \{x \in X \mid x \notin f(x)\} \in \mathcal{P}(X)$ eine Zeile geben. Diese Zeile – wir nehmen an, es sei die Zeile für $f(x_k)$ – enthält nun in Spalte j immer genau das Inverse des Diagonaleintrags (j, j) . Da der Diagonaleintrag (k, k) nicht invers zu sich selbst sein kann, haben wir einen Widerspruch zu unserer Annahme.

	x_0	$\dots$	x_i	$\dots$	x_k	$\dots$	x_j	$\dots$
$f(x_0)$	N	$\dots$	Y	$\dots$	Y	$\dots$	N	$\dots$
$\vdots$								
$f(x_i)$	N	$\dots$	Y	$\dots$	Y	$\dots$	Y	$\dots$
$\vdots$								
$f(x_k)$	Y	$\dots$	N	$\dots$	?	$\dots$	Y	$\dots$
$\vdots$								
$f(x_j)$	Y	$\dots$	Y	$\dots$	N	$\dots$	N	$\dots$
$\vdots$								

Abb. 19: Illustration des Diagonalarguments beim Beweis des Satzes von Cantor

Zum Ende des Unterkapitels möchten wir Diagonalargumente nochmal aus einem anderen Blickwinkel, mit Hilfe eines Rätsels, betrachten.

Rätsel: Der legendäre *Barbier von Waldkirch* rasierte genau die Männer, die sich nicht selbst rasierten. Gab es den Barbier von Waldkirch wirklich?

Wir betrachten zum Lösen dieses Rätsels die folgende quadratische Matrix, die je eine Zeile und eine Spalte für jeden Mann aus Waldkirch hat. Eintrag (i, j) enthält genau dann ein Y, wenn der i -te Mann den j -ten Mann rasiert. Gab es den Barbier von Waldkirch, so muss er in einer Zeile zu finden sein. Nehmen wir an, das sei die k -te Zeile. In dieser Zeile enthält nun der Eintrag in der i -ten Spalte immer genau das Inverse des Diagonaleintrags (i, i) . Da der Diagonaleintrag (k, k) nicht invers zu sich selbst sein kann, kann es keine Zeile für den Barbier von Waldkirch geben.

	Hans	...	Horst	...	M_k	...	Hubert	...
Hans	N	...	Y	...	Y	...	N	...
⋮								
Horst	N	...	Y	...	Y	...	Y	...
⋮								
M_k	Y	...	N	...	?	...	Y	...
⋮								
Hubert	Y	...	Y	...	N	...	N	...
⋮								

Abb. 20: Illustration des Diagonalarguments beim Rätsel des Barbier von Waldkirch

7.3 Sprachakzeptanz und Berechenbarkeit

Wir möchten nun die Idee von Berechenbarkeit auch auf die zweite Aufgabe von Turingmaschinen, das Akzeptieren von Sprachen, übertragen. Wir definieren dafür die Menge der Sprachen, für die eine Lösung für das Wortproblem (also $w \in L?$) berechnet werden kann, wie folgt.

Def. 7.4: Sei $\mathcal{M}$ eine TM. Wir sagen $\mathcal{M}$ *entscheidet* $L \subseteq \Sigma^*$, falls die folgenden beiden Eigenschaften gelten:

1. $\mathcal{M}$ akzeptiert L .
2. $\mathcal{M}$ hält für jede Eingabe an.

Eine Sprache $L \subseteq \Sigma^*$ heißt *entscheidbar*, falls es eine TM gibt, die L entscheidet.[†]

⊕

Satz 7.6: Es gibt Sprachen, die nicht entscheidbar sind.

Vorlesung:
23.01.2019

BEWEIS: Wir definieren die Abbildung $\text{lang2power} : \mathcal{P}(\Sigma^*) \rightarrow \mathcal{P}(\mathbb{N})$

$$\text{lang2power}(L) = \{\text{stdnum}(w) \mid w \in L\}$$

[†] Ein alternativer in der Literatur auch oft verwendeter Begriff für „entscheidbar“ ist „rekursiv“.

(analog zu Lemma 7.3) und zeige, dass diese surjektiv ist. Führe anschließend analog zum Beweis von Satz 7.5 einen Widerspruchsbeweis. Annahme: Jede Sprache $L \subseteq \Sigma^*$ sei entscheidbar. Also gibt es eine surjektive Abbildung $g : \mathbb{M} \rightarrow \mathcal{P}(\Sigma^*)$. Somit wäre auch

$$\text{lang2power} \circ g \circ \text{bin2tm} \circ \text{stdnum}^{-1} : \mathbb{N} \rightarrow \mathcal{P}(\mathbb{N})$$

surjektiv. Dies steht im Widerspruch zu Satz 7.4. $\square$

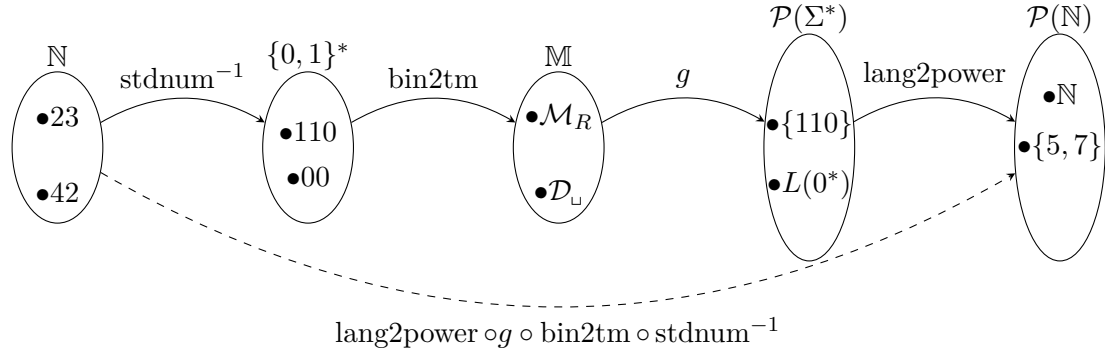


Abb. 21: Konkatenierte Abbildung aus dem Widerspruchsbeweis zu Satz 7.6

Def. 7.5: Sei $L \subseteq \Sigma^*$ eine Sprache. Die *charakteristische Funktion*[†] $\chi_L : \Sigma^* \rightarrow \{0,1\}$ ist wie folgt definiert:

$$\chi_L(w) = \begin{cases} 1 & \text{falls } w \in L \\ 0 & \text{falls } w \notin L \end{cases} \quad \oplus$$

Der folgende Satz zeigt uns nun einen direkten Zusammenhang zwischen den beiden Aufgaben einer Turingmaschine, dem Berechnen von Funktionen und dem Akzeptieren von Wörtern.

Satz 7.7: Sei $\{0,1\} \subseteq \Sigma$. Eine Sprache $L \subseteq \Sigma^*$ ist genau dann entscheidbar, wenn die charakteristische Funktion χ_L berechenbar ist.

Beweis: Siehe Vorbereitungsblatt 12 Aufgabe 2.

7.4 Das Halteproblem

Aus den bisherigen Sätzen wissen wir nur, dass es nicht berechenbare Funktionen und unentscheidbare Sprachen geben muss. Es besteht also noch Hoffnung, dass kein praxisrelevantes Problem betroffen ist. In diesem Kapitel werden wir nun aber für ein ganz konkretes praxisrelevantes Problem die Unentscheidbarkeit nachweisen.

[†] Die charakteristische Funktion wurde auf Vorbereitungsblatt 12 eingeführt.

Def. 7.6: Das spezielle Halteproblem besteht aus allen Codes von TMs, die anhalten, falls sie auf den eigenen Code angesetzt werden.

$$K = \{w \in \{0, 1\}^* \mid M_w \text{ angesetzt auf } w \text{ hält}\} \quad \oplus$$

Satz 7.8: Das spezielle Halteproblem ist unentscheidbar.

Wir geben zunächst einen kurzen formalen Beweis mit Hilfe eines Diagonalarguments und diskutieren diesen anschließend.

BEWEIS: Angenommen, $\mathcal{M}_\epsilon$ wäre eine TM, die K entscheidet. Dann gibt es auch eine TM $\mathcal{M}_{\text{evil}}$, (siehe Abb. 22) die zunächst $\mathcal{M}_\epsilon$ auf ihre Eingabe anwendet.[†] Falls $\mathcal{M}_\epsilon$ akzeptiert, dann geht $\mathcal{M}_{\text{evil}}$ in eine Endlosschleife. Falls $\mathcal{M}_\epsilon$ nicht akzeptiert, dann hält $\mathcal{M}_{\text{evil}}$ an. Nun gilt:

$$\begin{array}{lll} \mathcal{M}_\epsilon \text{ akzeptiert } \ulcorner \mathcal{M}_{\text{evil}} \urcorner & \begin{array}{l} \text{Def. von } K \\ \text{gdw.} \end{array} & \mathcal{M}_{\text{evil}} \text{ angesetzt auf } \ulcorner \mathcal{M}_{\text{evil}} \urcorner \text{ hält} \\ & \begin{array}{l} \text{Def. von } \mathcal{M}_{\text{evil}} \\ \text{gdw.} \end{array} & \mathcal{M}_\epsilon \text{ akzeptiert } \ulcorner \mathcal{M}_{\text{evil}} \urcorner \text{ nicht} \end{array}$$

Ein Widerspruch. □

Definition der TM $\mathcal{M}_{\text{evil}}$ aus dem Beweis von Satz 7.8 mit Hilfe eines Flussdiagramms. Die TM $\mathcal{M}_{\chi_K}$ welche die charakteristische Funktion von K implementiert, verhält sich wie $\mathcal{M}_\epsilon$, aber statt zu akzeptieren (bzw. nicht zu akzeptieren), schreibt diese 1 (bzw. 0) auf das Band und hält (siehe Satz 7.7).

$$\hookrightarrow \mathcal{M}_{\chi_K} \xrightarrow{1} \mathcal{D}_\sqcup \hookrightarrow \Gamma$$

Abb. 22: Definition der TM $\mathcal{M}_{\text{evil}}$ aus dem Beweis von Satz 7.8.

Wir wollen diesen Beweis nun wieder graphisch illustrieren. Die folgende Matrix enthält eine Spalte für jedes Wort $w \in \{0, 1\}^*$. Wir wählen z.B. $w_i = \text{stdnum}^{-1}$ und aus der Bijektivität von stdnum folgt, dass jedes $w \in \{0, 1\}^*$ einmal vorkommt. Wir haben außerdem unendlich viele Zeilen. In Zeile i steht die durch w_i codierte TM. Aus Lemma 7.2 folgt also, dass jede TM in mindestens einer Zeile vorkommt. In Eintrag (i, j) steht ein Y, wenn $\mathcal{M}_{w_i}$ angesetzt auf w_j hält, und ansonsten ein N. (Wir können diese Tabelleneinträge zwar nicht berechnen, doch dies ist hier nicht wichtig.) Die TM $\mathcal{M}_\epsilon$, deren

[†] Der Name $\mathcal{M}_\epsilon$ soll den hohen Wert dieser TM betonen. Gäbe es einen Algorithmus zur Terminierungsanalyse, könnten wir damit viele Probleme in der Softwareentwicklung lösen. Der Name $\mathcal{M}_{\text{evil}}$ soll suggerieren, dass diese TM unsere entsprechenden Hoffnungen zunichte macht.

Existenz wir im Beweis angenommen haben (die für alle $w \in \{0,1\}^*$ entscheidet, ob $\mathcal{M}_w$ angesetzt auf w hält) ist eine TM, die alle Diagonaleinträge in unserer Matrix bestimmen kann. Die daraufhin konstruierte TM $\mathcal{M}_{\text{evil}}$ (siehe Abb. 22) muss in irgendeiner Zeile vorkommen. Sei w_k das Wort, das die TM $\mathcal{M}_{\text{evil}}$ codiert (also $\mathcal{M}_{w_k} = \mathcal{M}_{\text{evil}}$). Die Definition von $\mathcal{M}_{\text{evil}}$ sagt uns, dass die Zeile $\mathcal{M}_{w_k}$ in Spalte i immer genau das Gegenteil (Y/N) des Diagonaleintrags (i, i) enthält. Die Betrachtung des Diagonaleintrags (k, k) zeigt uns den Widerspruch. Die Annahme der Existenz von $\mathcal{M}_{\epsilon}$ war also falsch.

	w_0	$\dots$	w_i	$\dots$	w_k	$\dots$	w_j	$\dots$
$\mathcal{M}_{w_0}$	Y	$\dots$	Y	$\dots$	Y	$\dots$	N	$\dots$
$\vdots$								
$\mathcal{M}_{w_i}$	N	$\dots$	N	$\dots$	Y	$\dots$	Y	$\dots$
$\vdots$								
$\mathcal{M}_{w_k}$	N	$\dots$	Y	$\dots$	?	$\dots$	Y	$\dots$
$\vdots$								
$\mathcal{M}_{w_j}$	Y	$\dots$	Y	$\dots$	N	$\dots$	N	$\dots$
$\vdots$								

Abb. 23: Illustration des Diagonalarguments beim Beweis der Unentscheidbarkeit des speziellen Halteproblems

Korollar 7.9: Das Komplement von K

$$\overline{K} = \{w \in \{0,1\}^* \mid \mathcal{M}_w \text{ angesetzt auf } w \text{ hält nicht}\}$$

ist nicht entscheidbar. $\oplus$

BEWEIS: Angenommen, $\overline{K}$ sei entscheidbar durch eine TM $\mathcal{M}$. Betrachte TM $\mathcal{M}'$, die zunächst $\mathcal{M}$ ausführt und dann das Ergebnis negiert. $\mathcal{M}'$ entscheidet K . Widerspruch zu Satz 7.8. $\square$

Def. 7.7: Eine Sprache $L \subseteq \Sigma^*$ heißt *semi-entscheidbar*, falls es eine TM gibt, die L akzeptiert.[†] $\oplus$

[†] Ein alternativer in der Literatur auch oft verwendeter Begriff für „semi-entscheidbar“ ist „rekursiv aufzählbar“.

Auf Vorbereitungsblatt 12 wurde in Aufgabe 3 durch Konstruktion einer TM gezeigt dass die von dieser akzeptierten Sprache semi-entscheidbar ist.

Lemma 7.10: K ist semi-entscheidbar.

BEWEIS: Die wie folgt konstruierte TM $\mathcal{M}$ akzeptiert K . Sei w die Eingabe. Führe die universelle TM für $\mathcal{M}_w$ mit Eingabe w aus. Falls diese Simulation stoppt, geht $\mathcal{M}$ in einen akzeptierenden Zustand und hält. $\square$

Satz 7.11: Wenn eine Sprache L und ihr Komplement $\bar{L}$ semi-entscheidbar sind, dann ist L entscheidbar.

BEWEIS: Sei $\mathcal{M}$ eine TM, die L akzeptiert, und sei $\bar{\mathcal{M}}$ eine TM, die $\bar{L}$ akzeptiert. Führe $\mathcal{M}$ und $\bar{\mathcal{M}}$ „parallel“ mit der gleichen Eingabe aus (z.B. mit einer 2-Band-TM). Eine TM muss anhalten, da für alle Eingaben w entweder $w \in L$ oder $w \in \bar{L}$ gilt.

- Falls $\mathcal{M}$ akzeptiert – halte und akzeptiere.
- Falls $\bar{\mathcal{M}}$ akzeptiert – halte aber akzeptiere nicht. $\square$

Korollar 7.12: Das Komplement des speziellen Halteproblems $\bar{K}$ ist nicht semi-entscheidbar. $\oplus$

BEWEIS: Nach Lemma 7.10 ist K semi-entscheidbar. Angenommen, $\bar{K}$ sei auch semi-entscheidbar. Dann wäre K nach Satz 7.11 entscheidbar. Widerspruch zu Satz 7.8. $\square$

Def. 7.8 (Reduktion): Seien $U, V \subseteq \Sigma^*$ Sprachen. U ist auf V *reduzierbar*, falls eine totale, berechenbare Funktion $f : \Sigma^* \rightarrow \Sigma^*$ existiert, sodass

$$\forall w \in \Sigma^* : w \in U \iff f(w) \in V$$

gilt. Wir nennen f *Reduktionsfunktion* außerdem nennen wir die zugehörige binäre Relation auf Sprachen die *Reduktionsrelation* und schreiben $U \preceq V$ wenn U auf V reduzierbar ist. $\oplus$

Vorlesung:
25.01.2019

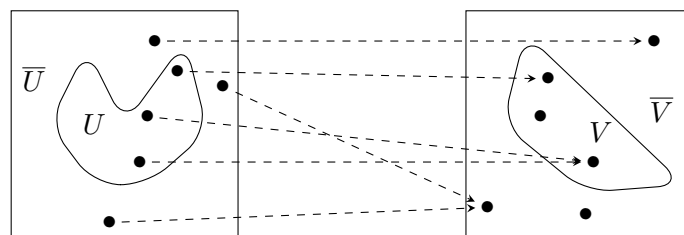


Abb. 24: Illustration einer Reduktion $U \preceq V$ von einer Sprache U auf eine Sprache V

Lemma 7.13: Falls $U \preceq V$ gilt und die Sprache V (semi-)entscheidbar ist, dann ist auch die Sprache U (semi-)entscheidbar.

BEWEIS: Sei $\mathcal{M}_V$ eine TM, die ein (Semi-)Entscheidungsverfahren für V implementiert, und sei $\mathcal{M}_f$ eine TM, die f berechnet. Konstruiere $\mathcal{M}_U$ wie folgt:

- Wende erst $\mathcal{M}_f$ auf die Eingabe w an.
- Führe dann $\mathcal{M}_V$ auf dem Ergebnis $f(w)$ aus.

Da f total ist, hält $\mathcal{M}_f$ immer. $\mathcal{M}_U$ hält also genau dann, wenn $\mathcal{M}_V$ hält. $\square$

Bemerkung: Mit der Kontraposition von Lemma 7.13 können wir zeigen, dass eine Sprache *nicht* (semi-)entscheidbar ist.

- $U \preceq V$ und U unentscheidbar $\Rightarrow V$ unentscheidbar.
- $U \preceq V$ und U nicht semi-entscheidbar $\Rightarrow V$ nicht semi-entscheidbar.

Def. 7.9: Das (*allgemeine*) *Halteproblem* ist definiert durch

$$H = \{u\#w \mid u, w \in \{0,1\}^* \text{ und } \mathcal{M}_u \text{ angesetzt auf } w \text{ hält}\}. \quad \oplus$$

Satz 7.14: H ist unentscheidbar.

BEWEIS: Wir zeigen $K \preceq H$. Betrache die Funktion $f(w) = w\#w$. Offensichtlich ist f total und berechenbar. Außerdem gilt:

$$w \in K \text{ gdw. } \mathcal{M}_w \text{ hält bei Eingabe } w \text{ an} \quad \text{gdw.} \quad \underbrace{w\#w}_{f(w)} \in H \quad \square$$

Satz 7.15: H ist semi-entscheidbar.

BEWEIS: Die wie folgt konstruierte TM $\mathcal{M}$ akzeptiert H . Zunächst simuliert $\mathcal{M}$ die universelle TM auf der Eingabe $v\#w$. Falls diese Simulation stoppt, geht $\mathcal{M}$ in einen akzeptierenden Zustand und hält. $\square$

Def. 7.10: Das *Halteproblem auf dem leerem Band* ist definiert durch

$$H_\varepsilon = \{u \in \{0,1\}^* \mid \mathcal{M}_u \text{ angesetzt auf das leere Band hält}\}. \quad \oplus$$

Satz 7.16: H_ε ist unentscheidbar.

BEWEIS: Wir zeigen die Reduktion $H \preceq H_\varepsilon$ mittels der Funktion f , die in der Eingabe das Trennsymbol $\#$ sucht und für die zerlegte Eingabe $u\#w$ die wie folgt definierte TM $\lceil \mathcal{M}_{u,w} \rceil$ konstruiert.

- Zuerst schreibt $\mathcal{M}_{u,w}$ das Wort w auf das leere Band (und läuft zum Anfang).
- Dann wendet $\mathcal{M}_{u,w}$ die durch u codierte TM $\mathcal{M}_u$ auf w an.

Enthält die Eingabe das Zeichen $\#$ nicht so bildet f auf den Code irgendeiner TM ab, deren Code nicht in H_ε liegt.[†]

Die Funktion f ist total und berechenbar und es gilt für alle $v \in \Sigma^*$ $v \in H$ gdw. $f(v) \in H_\varepsilon$. $\square$

Bemerkung: Es gilt auch umgekehrt, dass H_ε auf H reduzierbar ist (ohne Beweis). Mit Satz 7.15 und Lemma 7.13 ist H_ε damit auch semi-entscheidbar.

Satz 7.17: Die Reduktionsrelation $\preceq$ ist reflexiv und transitiv.

BEWEIS: Siehe Übungsblatt 13. $\square$

Nun betrachten wir TMs vom Blickwinkel der von ihnen berechneten (partiellen) Funktionen.^{††}

Satz 7.18 (Satz von Rice): Sei R die Menge aller (partiellen) berechenbaren Funktionen und S eine nichtleere, echte Teilmenge davon (also $\emptyset \subsetneq S \subsetneq R$). Dann ist die Sprache

$$C_S = \{u \in \{0,1\}^* \mid \mathcal{M}_u \text{ berechnet eine Funktion aus } S\}$$

unentscheidbar. $\lceil$

- Welche Bedeutung hat dieser Satz für die Praxis?
 - Es ist unentscheidbar, ob ein Programm tut, was es soll (die Spezifikation S), egal wie einfach die Eigenschaft ist, z.B. die Identitätsfunktion zu berechnen.
- Was ist mit den beiden Fällen $S = \emptyset$ und $S = R$? $\rightarrow$ trivial entscheidbar $\rfloor$

BEWEIS: Sei S beliebig sodass $\emptyset \subsetneq S \subsetneq R$ gilt. Angenommen, es gibt eine TM $\mathcal{M}_\epsilon$, die C_S entscheidet. Sei $\Omega \in R$ die überall undefinierte Funktion. Wir betrachten zwei Fälle.

[†] Wir könnten z.B. die durch das folgende Flussdiagramm beschriebene TM nehmen, denn diese hält auf keiner Eingabe: $\rightarrow \mathcal{D}_\sqcup \hookrightarrow \Gamma$

^{††}Henry Gordon Rice *1920 †2003

- Fall 1: $\Omega \in S$

Da $S \subsetneq R$ gilt, gibt es eine berechenbare Funktion $f \in R \setminus S$. Sei $\mathcal{M}_f$ eine TM, die f berechnet. Sei $w \in \{0,1\}^*$ beliebig. Wir definieren zunächst die TM $\mathcal{M}_{w,f}$ wie folgt: $\mathcal{M}_{w,f}$ führt zunächst die durch w codierte TM $\mathcal{M}_w$ auf der leeren Eingabe aus. Falls $\mathcal{M}_w$ anhält, wendet $\mathcal{M}_{w,f}$ dann $\mathcal{M}_f$ auf die tatsächliche Eingabe an. ($\mathcal{M}_{w,f}$ sollte also zunächst eine Kopie der Eingabe speichern, z.B. als 2-Band-TM, die ein Band für die Eingabe verwendet.)

Die von $\mathcal{M}_{w,f}$ berechnete Funktion ist also:

$$f_{\mathcal{M}_{w,f}} = \begin{cases} f & \text{falls } \mathcal{M}_w \text{ auf dem leeren Band hält,} \\ \Omega & \text{sonst.} \end{cases}$$

Wir definieren nun die TM $\mathcal{M}_{\text{Evil}}$ (abhängig von w) wie folgt: $\mathcal{M}_{\text{Evil}}$ nimmt die Eingabe w , konstruiert die TM $\mathcal{M}_{w,f}$ und wendet $\mathcal{M}_\epsilon$ auf $\ulcorner \mathcal{M}_{w,f} \urcorner$ an. Nun gilt:

$$\begin{aligned} \mathcal{M}_\epsilon \text{ akzeptiert } \ulcorner \mathcal{M}_{w,f} \urcorner &\iff \mathcal{M}_{w,f} \text{ berechnet eine Funktion in } S \\ &\iff \mathcal{M}_{w,f} \text{ berechnet } \Omega \\ &\iff \mathcal{M}_w \text{ hält nicht auf dem leeren Band} \end{aligned}$$

Also entscheidet $\mathcal{M}_{\text{Evil}}$ die Sprache H_ϵ . Widerspruch zu Satz 7.16.

- Fall 2: $\Omega \notin S$

Analog zu Fall 1, wobei dir diesmal $f \in S$ wählen. □

Bemerkung: Gibt es noch „schwerere“ Sprachen als das Halteproblem? In gewissem Sinne ja, denn es gibt Sprachen L , sodass sowohl L als auch $\bar{L}$ nicht semi-entscheidbar sind. Ein Beispiel ist das Halteproblem für *alle* Eingaben:

Def. 7.11: Das *universelle Halteproblem* ist definiert durch

$$H_{\forall w} = \{u \in \{0,1\}^* \mid \mathcal{M}_u \text{ angesetzt auf jedes beliebige Wort } w \in \Sigma^* \text{ hält}\}. \quad \oplus$$

Satz 7.19: Sowohl $H_{\forall w}$ als auch $\bar{H}_{\forall w}$ sind nicht semi-entscheidbar.

Vorlesung:
30.01.2019

Satz 7.20: Seien L_1, L_2 entscheidbare Sprachen. Dann sind auch $L_1 \cup L_2$, $L_1 \cap L_2$ und $\bar{L}_1$ entscheidbar.

BEWEIS: Seien L_1, L_2 beliebige entscheidbare Sprachen und $\mathcal{M}_1, \mathcal{M}_2$ TMs, welche die Sprachen entscheiden.

„ $\cup$ “: Simuliere für eine Eingabe w zunächst $\mathcal{M}_1$ und dann $\mathcal{M}_2$. Akzeptiere, falls eine der beiden TMs akzeptiert.

„ $\cap$ “: Analog zu „ $\cup$ “; siehe Übungsblatt 12 Aufgabe 4.

„ $\overline{L_1}$ “: Simuliere $\mathcal{M}_1$ und akzeptiere genau dann, wenn $\mathcal{M}_1$ nicht akzeptiert. $\square$

Satz 7.21: Seien L_1, L_2 semi-entscheidbare Sprachen. Dann sind auch $L_1 \cup L_2$ und $L_1 \cap L_2$ semi-entscheidbar.

BEWEIS: Seien L_1, L_2 beliebige semi-entscheidbare Sprachen und $\mathcal{M}_1, \mathcal{M}_2$ TMs, welche die Sprachen entscheiden. „ $\cup/\cap$ “: Simuliere für eine Eingabe w zunächst $\mathcal{M}_1$ und $\mathcal{M}_2$ „parallel“ (z.B. mit einer 2-Band-TM). Akzeptiere entsprechend zu $\cup/\cap$. $\square$

Sprache	ist semi-entscheidbar	Komplement ist semi-entscheidbar
Spezielles Halteproblem K	✓	-
Allgemeines Halteproblem H	✓	-
Halteproblem auf dem leeren Band H_ε	✓	-
Universelles Halteproblem $H_{\forall w}$	-	-

Abb. 25: Semi-Entscheidbarkeit der in diesem Kapitel vorgestellten unentscheidbaren Sprachen

7.5 Nichtdeterministische Turingmaschinen

Zunächst definieren wir einige Begriffe um das Verhalten von NTMs zu beschreiben. Insbesondere wollen wir den Begriff des *haltens*, der bisher nur für DTMs definiert war auf NTMs erweitern.

Def. 7.12 (Berechnung): Sei $\mathcal{M}$ eine TM mit Eingabealphabet Σ und Rechenschrittrelation $\vdash$. Sei $n \in \mathbb{N}$. Wir nennen eine Folge von Konfigurationen $u_0q_0v_0, \dots, u_nq_nv_n$, sodass $u_0q_0v_0$ eine Startkonfiguration ist, eine *Berechnung von $\mathcal{M}$ in n Schritten* wenn $u_iq_iv_i \vdash u_{i+1}q_{i+1}v_{i+1}$ für alle i mit $0 \leq i < n$ gilt. Die *Anzahl der benutzten Bandzellen* dieser Berechnung ist $|u_n| + |v_n|$. Wir nennen eine unendliche Folge von Konfigurationen $u_0q_0v_0, u_1q_1v_1, \dots$, sodass $u_0q_0v_0$ eine Startkonfiguration ist, eine *unendliche Berechnung von $\mathcal{M}$* wenn $u_iq_iv_i \vdash u_{i+1}q_{i+1}v_{i+1}$ für alle $i \in \mathbb{N}$ gilt. Wir sagen $\mathcal{M}$ *hält* auf Eingabe w wenn es keine unendliche Berechnung mit Startkonfiguration $q^{\text{init}}w$ gibt. $\oplus$

Satz 7.22: Sei $\mathcal{M}$ eine NTM. Dann existiert eine DTM $\mathcal{M}'$, sodass gilt:

- $\mathcal{M}'$ akzeptiert $L(\mathcal{M})$.
- $\mathcal{M}'$ hält genau dann, wenn $\mathcal{M}$ hält.

BEWEIS (Skizze): Idee: In jedem Schritt gibt es nur endlich viele Möglichkeiten, eine Transition zu wählen. Für eine feste Schrittzahl n ist die Anzahl der möglichen Konfigurationsfolgen also endlich. Wir legen ein n fest und probieren nacheinander alle möglichen Konfigurationsfolgen der Länge n durch. (Anders formuliert: Wir laufen in Breitensuche durch den (endlichen) Suchbaum und zählen alle Möglichkeiten auf. Dann simulieren wir nacheinander alle Möglichkeiten systematisch.) Finden wir eine Haltekonfiguration, so kann $\mathcal{M}'$ auch halten. Ansonsten müssen wir n erhöhen und eine neue Suche starten.

Sei r der höchste Grad an Nichtdeterminismus von $\mathcal{M}$. Wir konstruieren $\mathcal{M}'$ mit 3 Bändern, die auf Band 1 das Eingabewort w unverändert speichert. Auf Band 2 werden nach und nach alle Wörter über $\{1, \dots, r\}$ der Länge nach und bei gleicher Länge in lexikografischer Reihenfolge erzeugt: $\varepsilon, 1, 2, \dots, r, 11, 12, \dots, 1r, 21, \dots, rr, 111, \dots$. Auf Band 3 wird jeweils die Simulation der aktuellen Auswahl durchgeführt.

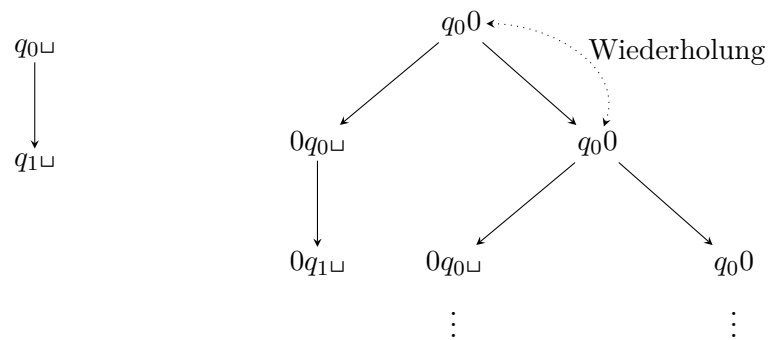
Wenn es eine gültige Konfigurationenfolge gibt, an deren Ende $\mathcal{M}$ akzeptiert, so wird $\mathcal{M}'$ diese irgendwann aufzählen und ebenfalls akzeptieren. Wenn nicht, gibt es zwei Möglichkeiten:

- Der Suchbaum ist endlich, d.h., jede mögliche Konfigurationenfolge von $\mathcal{M}$ führt irgendwann in eine Haltekonfiguration. Dann hält $\mathcal{M}'$ irgendwann.
- Es gibt nicht-terminierende Konfigurationenfolgen von $\mathcal{M}$. Dann hält $\mathcal{M}'$ nie, sondern zählt immer längere Reihenfolgen auf. $\square$

Bsp.: Betrachte die NTM $\mathcal{M} = (\{0, 1\}, \{q_0, q_1\}, \{0, 1, \sqcup\}, \delta, q_0, \sqcup, \{q_0\})$ mit

$$\begin{aligned}\delta(q_0, 0) &= \{(q_0, 0, R), (q_0, 0, N)\} \\ \delta(q_0, \sqcup) &= \{(q_1, \sqcup, N)\}.\end{aligned}$$

Auf manchen Eingaben hält $\mathcal{M}$ immer (z.B. auf $w_1 = \varepsilon$), auf manchen gibt es aber auch nicht-terminierende Ausführungen (z.B. auf $w_2 = 0$). Unten sind die Suchbäume für die Konfigurationsfolgen für w_1 und w_2 skizziert.



Bemerkung: Es werden u.U. Reihenfolgen aufgezählt, die nicht durchführbar sind. Das kann aber erkannt werden; in diesem Fall wird die Simulation einfach übersprungen.

Der Suchaufwand ist exponentiell (Suchbaum mit Verzweigungsgrad r).

Wir wollen im Folgenden Probleme als Sprachen darstellen und beginnen zunächst mit dem Erfüllbarkeitsproblem der Aussagenlogik.

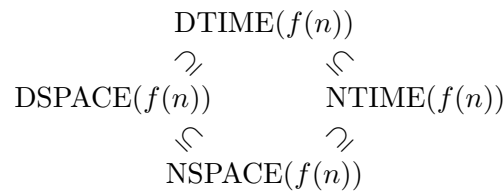


Abb. 26: Inklusionsrelationen zwischen Komplexitätsklassen

Aussagenlogische Formeln als Sprachen

Wir gehen davon aus, dass die Leser des Skripts mit Aussagenlogik (AL)[†] vertraut sind, stellen die von uns verwendete Syntax vor und bitten den Leser, sich die entsprechende Semantik zu erschließen.

Im Folgenden möchten wir Mengen von AL-Formeln als Sprachen beschreiben. Dafür müssen wir zunächst ein geeignetes Alphabet wählen. Dabei nehmen wir „0“ und „1“ für die Konstanten, „¬“, „∧“ und „∨“ für die Junktoren und runde Klammern „(“ und „)“. Bei der Darstellung der Aussagenvariablen stehen wir nun vor der folgenden Herausforderung: Es gibt unendliche viele Aussagenvariablen $A_0, A_1, A_2, A_3, \dots$, aber unser Alphabet kann nur endlich viele Zeichen haben. Unsere Lösung dafür ist sehr ähnlich wie die, die wir auch auf Papier verwenden. Wir stellen eine Aussagenvariable durch ein „A“ dar, das von einer Ziffernfolge gefolgt wird. Die Aussagenvariable A_{1337} wird so durch das Wort $A1337$ der Länge 5 repräsentiert. Wir müssen unser Alphabet also noch um „A“, „2“, ... „9“ ergänzen und erhalten

$$\Sigma_{\text{AL}} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, \neg, \wedge, \vee, (,)\}.$$

Def. 8.3: Die Menge der AL-Formeln ist die Sprache der kontextfreien Grammatik $\mathcal{G}_{\text{AL}} = (\Sigma_{\text{AL}}, N, P, S)$ mit

$$\begin{aligned}
 N &= \{S, Z\} \\
 P &= \{S \rightarrow 0 \mid 1 \mid AZ \mid \neg S \mid (S \wedge S) \mid (S \vee S) \\
 &\quad Z \rightarrow 0Z \mid 1Z \mid 2Z \mid 3Z \mid 4Z \mid 5Z \mid 6Z \mid 7Z \mid 8Z \mid 9Z \mid \\
 &\quad 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\}.
 \end{aligned}
 \quad \oplus$$

Wir nehmen nun unsere Definition von Formeln, um ein bekanntes Problem mit Hilfe einer Sprache zu beschreiben.

[†]Eine Einführung in Aussagenlogik finden Sie im Skript zur Vorlesung *Logik für Studierende der Informatik* aus dem WS 2017/18. Wir werden in dieser Vorlesung eine sehr ähnliche Notation verwenden. <http://home.mathematik.uni-freiburg.de/junker/ws17/logik-info.html>

Def. 8.4: Wir nennen die Menge der erfüllbaren AL-Formeln **SAT!**.

$$\mathbf{SAT!} = \{F \mid F \in L(\mathcal{G}_{\text{AL}}) \text{ und } F \text{ erfüllbar}\} \quad \oplus$$

Wir können nun also die Frage, ob die Formel F erfüllbar ist, als Wortproblem „ $F \in \mathbf{SAT!}$?“ darstellen. Wir wollen im Folgenden eine TM konstruieren, die **SAT!** entscheidet, betrachten aber als Vorstufe hierfür zunächst eine einfachere Sprache.

Wir definieren uns $\mathcal{G}_{\text{AL0}}$ als die obige Grammatik, in der die Regel $S \rightarrow AZ$ fehlt. Offensichtlich enthält $L(\mathcal{G}_{\text{AL0}})$ genau die AL-Formeln, die keine Aussagenvariablen enthalten.

Weiter definieren wir

$$\text{SAT}_0 = \{F \mid F \in L(\mathcal{G}_{\text{AL0}}) \text{ und } F \text{ erfüllbar}\}$$

und konstruieren eine DTM, die SAT_0 entscheidet.

Bsp. 8.1: Sei $\mathcal{M}_{\text{SAT}_0}$ eine DTM, deren Verhalten wir wie folgt informell beschreiben.

- Grundlegende Idee: Laufe von links nach rechts über die Eingabe und ersetze alle Operationen auf Konstantensymbolen direkt durch das Ergebnis (z.B. $1 \wedge 0 \rightsquigarrow 0$). Wiederhole dies, bis 0 oder 1 auf dem Band steht.
- Problem: Das Resultat dieser Operationen hat weniger Zeichen als die Eingabe.
- Lösung: Erweitere das Bandalphabet um ein Zeichen $\blacksquare$, um entstandene Leerstellen zu markieren (z.B. $1 \wedge 0 \rightsquigarrow 0\blacksquare\blacksquare$). Alternative: Verschiebe den Bandinhalt entsprechend.

Frage: Was ist die bestmögliche Zeitkomplexität, die wir für $\mathcal{M}_{\text{SAT}_0}$ geben können?

Antwort: Wir müssen im schlimmsten Fall für jeden Operator einmal über die Eingabe laufen. Die Zeitkomplexität ist also „garantiert nicht viel schlimmer als n^2 “, wobei n die Größe der Eingabe ist. Wir können die Zeitkomplexität aber nicht exakt angeben, da die Beschreibung von $\mathcal{M}_{\text{SAT}_0}$ zu unpräzise ist. (Läuft der Kopf gelegentlich links und rechts über die Eingabe hinaus um das Wortende zu bestimmen? Läuft der Kopf beim Ersetzen zurück?)

Ein Formalismus zum Beschreiben von Aussagen wie „garantiert nicht viel schlimmer als ...“ ist das (große) $\mathcal{O}$ -Symbol der Bachmann–Landau-Notation. Wir gehen davon aus, dass die Leser des Skripts mit diesem Formalismus vertraut sind, wollen die Definition von (groß) $\mathcal{O}$ hier aber nochmal wiederholen.

Def. 8.5: Sei $g : \mathbb{N} \rightarrow \mathbb{N}$.

$$\mathcal{O}(g(n)) = \{f : \mathbb{N} \rightarrow \mathbb{N} \mid \exists n_0, k \in \mathbb{N} : \forall n \geq n_0 : f(n) \leq k \cdot g(n)\} \quad \oplus$$

Die Klasse $\mathcal{O}(g(n))$ enthält also genau die Funktionen, die für „große Argumente“ durch ein Vielfaches von $g(n)$ beschränkt sind.

Über die Schwierigkeit von SAT_0 können wir nun also sagen: Es gibt ein $f(n) \in \mathcal{O}(n^2)$, sodass $\text{SAT}_0 \in \text{DTIME}(f(n))$.

Bsp. 8.2: Sei $\mathcal{M}_{\text{SAT}}$ eine NTM, deren Verhalten wir wie folgt informell beschreiben.

- Phase 1: Laufe über die Eingabe, sammle alle Aussagenvariablen $A_{k_1}, \dots, A_{k_n}$, rate (wähle nichtdeterministisch) für jede davon eine Belegung b_{k_i} und schreibe

$$A_{k_1} = b_{k_1}, \quad \dots, \quad A_{k_n} = b_{k_n}$$

auf ein zweites Band. Dabei soll jede Variable höchstens einmal geschrieben werden, auch wenn sie mehrfach in der Eingabe vorkommt.

Zeitkomplexität: Ein Vergleich zweier Aussagenvariablen ist in $\mathcal{O}(n)$ möglich, pro Variablen müssen höchstens $\mathcal{O}(n)$ Vergleiche gemacht werden und in der Eingabe sind höchstens $\mathcal{O}(n)$ Variablen enthalten. Die Zeitkomplexität von Phase 1 kann also durch $\mathcal{O}(n^3)$ beschränkt werden.

- Phase 2: Laufe über die Eingabe und ersetze alle Aussagenvariablen durch die geratene Belegung.

Zeitkomplexität: Durch $\mathcal{O}(n^3)$ beschränkt, nahezu gleiche Argumentation wie oben.

- Phase 3: Wende die in Bsp. 8.2 beschriebene DTM $\mathcal{M}_{\text{SAT}_0}$ auf den Bandinhalt an.

Zeitkomplexität: Durch $\mathcal{O}(n)^2$ beschränkt.

Die Zeitkomplexität von $\mathcal{M}_{\text{SAT}}$ ist also durch $\mathcal{O}(n^3)$ beschränkt.

Über die Schwierigkeit von **SAT!** können wir somit sagen: Es gibt ein $f(n) \in \mathcal{O}(n^3)$, sodass **SAT!** $\in \text{NTIME}(f(n))$.

Vorlesung:
01.02.2019

Frage: Angenommen, wir möchten ein schnelles Entscheidungsverfahren für **SAT!** implementieren. Hilft uns dann die Konstruktionsidee von $\mathcal{M}_{\text{SAT}}$?

Antwort: Eher nein. Die Konstruktion basiert darauf, dass wir nichtdeterministisch eine Belegung „raten“. Es ist unklar, wie wir diesen Nichtdeterminismus auf einem (deterministischen) Computer effizient implementieren können. Eine Möglichkeit wäre analog zum Beweis von Satz 7.22 alle nichtdeterministischen Wahlmöglichkeiten zu simulieren. Da wir $\mathcal{M}_{\text{SAT}}$ aber pro Aussagenvariable nichtdeterministisch zwischen 0 und 1 wählen können, wäre das Verfolgen aller Wahlmöglichkeiten nur mit exponentiellem Aufwand möglich.

Frage: Gibt es ein „schnelles“ Entscheidungsverfahren für **SAT!** oder ist das Problem so schwierig, dass es kein solches Verfahren geben kann?

Antwort: Die Informatik kann bisher (Stand Januar 2019) auf beide Frageteile keine Antwort liefern. Wir stellen aber im Folgenden einen Formalismus vor, mit dem sich diese Frage präziser formulieren lässt und wir für verwandte Fragen hilfreiche Antworten liefern können.

Um Missverständnisse in Terminologie und Notation zu vermeiden, wiederholen wir hier nochmal die aus der Schule bekannte Definition eines Polynoms.

Def. 8.6: Ein Polynom ist eine Funktion $p : \mathbb{N} \rightarrow \mathbb{N}$ mit $\exists k \in \mathbb{N} \ a_0, \dots, a_k \in \mathbb{N}$ und $p(n) = \sum_i^k a_i n^i$ $\oplus$

Bemerkung: Die Funktion $f(n) = 2^n$ ist kein Polynom, da n hier ein Exponent ist.

Def. 8.7:

$$\begin{aligned} \mathbf{P!} &= \bigcup_{p \text{ Polynom}} \text{DTIME}(p(n)) \\ \mathbf{NP!} &= \bigcup_{p \text{ Polynom}} \text{NTIME}(p(n)) \end{aligned} \quad \oplus$$

Bemerkung: Die Laufzeitanalysen der TMs aus Bsp. 8.1 und Bsp. 8.2 zeigen, dass $\text{SAT}_0 \in P$ und $\mathbf{SAT!} \in \mathbf{NP!}$.

Offensichtlich gilt $P \subseteq \mathbf{NP!}$, da die DTM ein Spezialfall der NTM ist. Die Frage, ob $P = \mathbf{NP!}$ oder $P \neq \mathbf{NP!}$ gilt, wurde bereits 1971 von Stephen Cook gestellt und ist das vermutlich bekannteste ungelöste Problem der Informatik.

Das „ $\mathbf{P!}$ vs. $\mathbf{NP!}$ “-Problem ist auch deshalb interessant, weil es eine Vielzahl praxisrelevanter Problemstellungen gibt (Travelling Salesman, Erfüllbarkeit der Aussagenlogik, ...), die in $\mathbf{NP!}$ liegen, aber für die bisher nur (deterministische) Algorithmen mit mindestens exponentieller Laufzeit gefunden wurden.

Angenommen, wir wollen in unserem Unternehmen ein neues Problem lösen (z.B. automatisches Verteilen von miteinander kommunizierenden Benutzern auf verschiedene Server, sodass die Netzwerklast am geringsten ist). Angenommen, wir wissen bereits,

- dass wir das Problem mit einer NTM in polynomieller Zeit lösen könnten,
- dass für unsere Eingabegrößen eine polynomielle Laufzeit noch akzeptabel, aber eine exponentielle Laufzeit nicht mehr akzeptabel wäre und
- dass es einen approximativen Algorithmus gibt, der polynomielle Laufzeit hat, aber nicht immer optimale Resultate liefert.

Wie gehen wir nun weiter vor? Suchen wir nach einem (exakten) Algorithmus mit polynomieller Laufzeit oder geben wir auf und implementieren den approximativen Algorithmus?

Wir stellen im Folgenden einen Formalismus vor, mit dem wir Aussagen der folgenden Form machen können: „Ich weiß zwar nicht, ob meine neues Problem in **P!** liegt oder nicht. Ich weiß aber, dass mein Problem so schwierig ist, dass das Finden eines polynomiellen Algorithmus **P! = NP!** implizieren würde.“

Def. 8.8 (Polynomielle Reduktion): Seien $U, V \subseteq \Sigma^*$ Sprachen. U ist auf V *polynomiell reduzierbar*, falls eine totale, berechenbare Funktion $f : \Sigma^* \rightarrow \Sigma^*$ existiert, sodass

- $\forall w \in \Sigma^* : w \in U \iff f(w) \in V$
- f wird von einer DTM berechnet, deren Zeitkomplexität ein Polynom ist.

gilt. Wir nennen f *Reduktionsfunktion* außerdem nennen wir die zugehörige binäre Relation auf Sprachen die *Reduktionsrelation* und schreiben $U \preceq_p V$ wenn U auf V polynomiell reduzierbar ist. $\oplus$

Eine aussagenlogische Formel F ist in *konjunktiver Normalform* (CNF)[†], wenn F eine Konjunktion von Disjunktionen von Literalen ist. Wir verwenden *CNF* als Notation für die Menge aller AL-Formeln in CNF. Analog schreiben wir *3CNF* für die Menge aller Formeln in CNF, bei denen jeder Konjunkt aus höchstens drei Disjunkten besteht.

Def. 8.9: Das Problem **3SAT!** ist wie folgt definiert.^{††}

Gegeben: Eine aussagenlogische Formel $F \in 3CNF$

Frage: Ist F erfüllbar?

$\oplus$

Alternativ könnten wir die Definition von **3SAT!** auch wie folgt aufschreiben.

$$\mathbf{3SAT!} = \{F \in L(\mathcal{G}_{\text{AL}}) \mid F \text{ ist erfüllbar und } F \text{ ist in } 3\text{CNF}\}$$

[†]Die Abkürzung CNF wurde in Unterabschnitt 3.2 für die Chomsky Normalform von kontextfreien Grammatiken benutzt, steht hier aber für die konjunktive Normalform von aussagenlogischen Formeln.

^{††} Typischerweise nennt man Sprachen, die im Kontext der Komplexitätstheorie definiert werden, *Probleme*. Es ist üblich, von einem konkreten Alphabet Σ und einer konkreten Codierung der Objekte (hier: Formeln) zu abstrahieren und das Problem als (Gegeben, Frage)-Paar zu formulieren. Es bleibt dem Leser überlassen, sich selbst geeignete Alphabete und Codierungen zu überlegen. Für den Fall von aussagenlogische Formeln haben wir dies in Def. 8.3 noch einmal gemacht, werden aber in Zukunft darauf verzichten.

Offensichtlich gilt $\mathbf{3SAT!} \preceq_p \mathbf{SAT!}$ (Reduktionsfunktion. Wir zeigen als Nächstes auch die umgekehrte Richtung.

Lemma 8.1: $\mathbf{SAT!} \preceq_p \mathbf{3SAT!}$

BEWEIS: Unser Ziel ist es nun, eine Transformation anzugeben, die sich in polynomieller Zeit berechnen lässt und jede aussagenlogische Formel F_{AL} in eine 3CNF-Formel F_{3CNF} überführt, sodass $F_{\text{AL}} \in \mathbf{SAT!} \iff F_{\text{3CNF}} \in \mathbf{3SAT!}$ gilt.

Sei F_{AL} eine beliebige Formel aus $L(\mathcal{G}_{\text{AL}})$.

1. Erzeuge aus F_{AL} eine äquivalente Formel F_{NNF} in Negationsnormalform[†]. Wir können jede Formel in Negationsnormalform bringen, indem wir mit Hilfe der De Morganschen Regeln die Negationen „nach innen“ ziehen.

$$\text{Beispiel: } \neg(\neg(A_1 \vee \neg A_3) \vee A_2) \rightsquigarrow ((A_1 \vee \neg A_3) \wedge \neg A_2)$$

Ideen zur Implementierung: Erstelle auf einem zusätzlichen Band eine Kopie der Formel. Lasse vor jedem nicht negierten Literal ein Feld Platz, um später ggf. ein Negationssymbol zu platzieren. Wende De Morgans Regel von außen nach innen an. Laufe für jede Anwendung einmal über die Formel.

2. Erzeuge aus F_{NNF} eine Formel $F_{\alpha\gamma}$ mit Biimplikationszeichen „ $\leftrightarrow$ “ mit Hilfe der folgenden induktiv definierten Abbildungen α und γ .^{††}

Idee: Die Abbildung γ liefert für jede $\wedge$ -Teilformel und jede $\vee$ -Teilformel eine neue Hilfsvariable. Diese Hilfsvariable hat in der resultierenden Formel den Wahrheitswert, den die entsprechende Teilformel in der Eingabeformel hätte. Die Abbildung α konstruiert für jede Teilformel die entsprechenden Bedingungen für die Hilfsvariable. Erzeugt γ keine Hilfsvariable, so erzeugt α die (nicht einschränkende) Bedingung 1.

$$\gamma(F) = \begin{cases} 0 & \text{falls } F = 0 \\ 1 & \text{falls } F = 1 \\ A & \text{falls } F = A \\ \neg F_1 & \text{falls } F = \neg F_1 \\ B_F & \text{falls } F = F_1 \wedge F_2 \\ B_F & \text{falls } F = F_1 \vee F_2 \end{cases}$$

[†]Eine Formel ist in Negationsnormalform, wenn der Negationsoperator immer nur direkt vor einer Variable vorkommt.

^{††}Die hier beschriebene Transformation ist auch als Tseytin-Transformation bekannt.

$$\alpha(F) = \begin{cases} 1 & \text{falls } F = 0 \\ 1 & \text{falls } F = 1 \\ 1 & \text{falls } F = A \\ 1 & \text{falls } F = \neg F_1 \\ (\gamma(F) \leftrightarrow \gamma(F_1) \wedge \gamma(F_2)) \wedge \alpha(F_1) \wedge \alpha(F_2) & \text{falls } F = F_1 \wedge F_2 \\ (\gamma(F) \leftrightarrow \gamma(F_1) \vee \gamma(F_2)) \wedge \alpha(F_1) \wedge \alpha(F_2) & \text{falls } F = F_1 \vee F_2 \end{cases}$$

Wir definieren das Resultat $F_{\alpha\gamma} := \gamma(F_{\text{NNF}}) \wedge \alpha(F_{\text{NNF}})$.

Beispiel:

$$\begin{aligned} \alpha((A_1 \vee \neg A_3) \wedge \neg A_2) &= (B_{((A_1 \vee \neg A_3) \wedge \neg A_2)} \leftrightarrow (B_{(A_1 \vee \neg A_3)} \wedge \neg A_2)) \\ &\quad \wedge (B_{(A_1 \vee \neg A_3)} \leftrightarrow (A_1 \vee \neg A_3)) \wedge 1 \wedge 1 \\ &\quad \wedge 1 \end{aligned}$$

Ideen zur Implementierung:

Wähle ein Bandalphabet, sodass das Zeichen B enthalten ist. Verwende außerdem eine zusätzliche Art von Klammern (z.B. eckige Klammern), um den Subskriptanteil der B-Variablen vom restlichen Bandinhalt zu unterscheiden. Laufe für jede $\{\wedge, \vee\}$ -Teilformel einmal über die Eingabe. Verwende ein zusätzliches Band zum Schreiben des Resultats. Verwende noch ein zusätzliches Band für die aktuell bearbeitete $\{\wedge, \vee\}$ -Teilformel, da diese immer zweimal benötigt wird (Variablenname und α). Bearbeite äußere Teilformeln vor inneren. Lösche Teile, die nicht mehr benötigt werden.

3. Ersetze die Formelteile mit Biimplikationszeichen in $F_{\alpha\gamma}$ wie folgt durch logisch äquivalente Formeln in 3CNF.

$$\begin{aligned} \bullet \quad F_1 \leftrightarrow (F_2 \wedge F_3) &\rightsquigarrow (F_1 \vee \neg F_2 \vee \neg F_3) \wedge (\neg F_1 \vee F_2) \wedge (\neg F_1 \vee F_3) \\ \bullet \quad F_1 \leftrightarrow (F_2 \vee F_3) &\rightsquigarrow (F_1 \vee \neg F_2) \wedge (F_1 \vee \neg F_3) \wedge (\neg F_1 \vee F_2 \vee F_3) \end{aligned}$$

Ideen zur Implementierung:

Verwende ein zusätzliches Band zum Schreiben des Resultats. Verwende noch ein zusätzliches Band für die Operanden der Biimplikation, da diese immer mehrfach benötigt werden.

4. Ersetze alle aussagenlogischen Variablen der Form B_F durch aussagenlogische Variablen der Form A_i .[†]

[†]Dieser Schritt ist nur nötig, damit das Resultat in dem von uns definierten Alphabet Σ_{AL} dargestellt werden kann. Alternativ hätten wir auch zu Beginn ein reichhaltigeres Alphabet wählen können.

Ideen zur Implementierung:

Finde zunächst den höchsten Index i_{max} von A_i -Variablen (speichere aktuelles Maximum auf zusätzlichem Band). Verwende anschließend $i_{max} + 1, i_{max} + 2, \dots$ für neue Variablen. Schreibe zunächst eine Übersetzungsvorschrift (z.B. $A_4 := B_{((A_1 \vee \neg A_3) \wedge \neg A_2)}$, $A_5 := B_{(A_1 \vee \neg A_3)}$) auf ein zusätzliches Band und ersetze erst dann.

Sei $f : \Sigma_{AL} \rightarrow \Sigma_{AL}$ eine Funktion, die alle Formeln aus $L(\mathcal{G}_{AL})$ entsprechend obiger Konstruktion abbildet und alle anderen Wörter unverändert lässt. Dann gilt:

- $F_{AL} \in \mathbf{SAT!} \iff f(F_{AL}) \in \mathbf{3SAT!}$ (hier ohne Beweis)
- f ist total und lässt sich in polynomieller Zeit berechnen (hier ohne Beweis). $\square$

Lemma 8.2: Falls $A \preceq_p B$ und $B \in \mathbf{P!}$ (bzw. $B \in \mathbf{NP!}$), dann gilt auch $A \in \mathbf{P!}$ (bzw. $A \in \mathbf{NP!}$).

BEWEIS: $B \in \mathbf{P!}$: Nach Annahme gibt es eine TM $\mathcal{M}$, die B in $p(n)$ Schritten akzeptiert. Es gibt außerdem eine TM $\mathcal{M}_f$, welche die Reduktion $A \preceq_p B$ implementiert. Die Laufzeit von $\mathcal{M}_f$ sei durch das Polynom q beschränkt.

Betrachte $\mathcal{M}' = \text{„erst } \mathcal{M}_f, \text{ dann } \mathcal{M} \text{ auf dem Ergebnis“}$. $\mathcal{M}'$ akzeptiert A .

Sei $w \in A$. $\mathcal{M}_f(w)$ liefert $f(w)$ in höchstens $q(|w|)$ Schritten mit $|f(w)| \leq q(|w|) + |w|$. $\mathcal{M}$ angesetzt auf $f(w)$ benötigt höchstens $p(|f(w)|) \leq p(q(|w|) + |w|)$ Schritte zum Akzeptieren.

Insgesamt gilt also $A \in \text{DTIME}(q(|w|) + |w| + p(q(|w|) + |w|)) \subseteq \mathbf{P!}$. $\square$

Def. 8.10:

- Eine Sprache U heißt **NP!-schwer**, falls $\forall L \in \mathbf{NP!}$ gilt: $L \preceq_p U$.
- Eine Sprache U heißt **NP!-vollständig**, falls U **NP!-schwer** ist und $U \in \mathbf{NP!}$ gilt. $\oplus$

TODO: Ausformulieren

- **NP!-schwer:** sehr starke Forderung, dass man alle(!) **NP!-Probleme** darauf reduzieren kann
- zunächst unklar, ob es überhaupt ein **NP!-schweres** Problem gibt
- bedeutet: wenn ich ein **NP!-vollst.** Problem gefunden habe, das ich effizient lösen kann, kann ich alle **NP!-Probleme** effizient lösen (wie folgender Satz zeigt).

Satz 8.3: Wenn eine Sprache A **NP!-vollständig** ist, dann gilt die folgende Äquivalenz.

$$A \in \mathbf{P!} \iff \mathbf{P!} = \mathbf{NP!}$$

BEWEIS:

„ $\Leftarrow$ “ trivial.

„ $\Rightarrow$ “ Es gilt $A \in \mathbf{P!} \subseteq \mathbf{NP!}$. Da A $\mathbf{NP!}$ -vollständig ist, gilt $\forall L \in \mathbf{NP!} : L \preceq_p A$. Dann folgt mit Lemma 8.2 auch $L \in \mathbf{P!}$. $\square$

Lemma 8.4: $\preceq_p$ ist reflexiv und transitiv.

BEWEIS: Übungsblatt 14, Aufgabe 1. $\square$

Bem.: Aus der Transitivität folgt: Sobald ein $\mathbf{NP!}$ -schweres Problem U bekannt ist, reicht es $U \preceq_p V$ zu zeigen, um zu beweisen, dass V ebenfalls $\mathbf{NP!}$ -schwer ist.

Satz 8.5 (Cook, Levin): **SAT!** ist $\mathbf{NP!}$ -vollständig.

Wir werden nur für **SAT!** die $\mathbf{NP!}$ -Schwierigkeit direkt beweisen und für alle anderen $\mathbf{NP!}$ -vollständigen Probleme die $\mathbf{NP!}$ -Schwierigkeit wie in obiger Bemerkung angedeutet mit Hilfe einer polynomiellen Reduktion zeigen. Da der Beweis von Satz 8.5 sehr umfangreich ist, wollen wir diesen etwas aufschieben und zunächst die $\mathbf{NP!}$ -Vollständigkeit weiterer Probleme zeigen.

Satz 8.6: **3SAT!** ist $\mathbf{NP!}$ -vollständig.

BEWEIS:

- **3SAT! $\in \mathbf{NP!}$**

Wir können dies auf zwei Arten zeigen. Entweder wir zeigen direkt, dass es eine NTM gibt, die **3SAT!** in polynomieller Zeit entscheidet, oder wir zeigen dies indirekt mit Hilfe einer polynomiellen Reduktion auf ein Problem in $\mathbf{NP!}$. Wir wählen letzteres Vorgehen, da zum Beweis nur nochmal erwähnt werden muss, dass **SAT!** $\in \mathbf{NP!}$ und (offensichtlich) **3SAT!** $\preceq_p$ **SAT!** gilt.

- **3SAT!** ist $\mathbf{NP!}$ -schwer

Folgt aus Lemma 8.1. $\square$

Wir gehen davon aus, dass die Leser des Skripts mit Graphen vertraut sind, machen aber zum Fixieren von Notation und Terminologie die folgende Definition.

Def. 8.11: Ein *gerichteter Graph* ist ein Paar $\mathcal{G} = (V, E)$, bei dem

- V eine Menge ist, deren Elemente wir *Knoten* nennen und

- $E \subseteq V \times V$ eine Menge von geordneten Paaren über V ist. Wir nennen diese geordneten Paare *Kanten*.

Ein *ungerichteter Graph* ist ein gerichteter Graph, bei dem E symmetrisch ist.[†] $\oplus$

Def. 8.12 (CLIQUE): Das Problem CLIQUE ist wie folgt definiert.

Gegeben: Ein ungerichteter Graph $\mathcal{G} = (V, E)$ und eine Zahl $k \in \mathbb{N}$.

Frage: Hat $\mathcal{G}$ eine k -Clique?

Eine k -Clique ist eine k -elementige Menge von Knoten, die paarweise durch Kanten verbunden sind, d.h., eine Menge $C \subseteq V$, sodass $|C| = k$ und $\forall u, v \in C : u \neq v \rightarrow (u, v) \in E$.

$\oplus$

Satz 8.7: CLIQUE ist **NP!**-vollständig.

Bemerkung: Der folgende Beweis ist für ein Vorlesungsskript sehr knapp gehalten. Der Beweis enthält aber genau die Informationen, die wir in Übungsaufgaben und Klausur zum Erreichen der maximalen Punktzahl erwarten würde.

Beweise der **NP!**-Vollständigkeit folgen typischerweise dem hier verwendeten Schema. Der schwierige (da Kreativität erfordernde) Teil ist dabei, eine geeignete Konstruktion für die Reduktionsfunktion zu finden. Für den folgenden Beweis wird die Idee der Konstruktion in Abb. 27 mit Hilfe eines Beispiels illustriert.

In unseren Übungs- und Klausuraufgaben wird die Aufgabenstellung manchmal solch ein Beispiel enthalten. Damit soll ein Hinweis auf eine geeignete Konstruktion gegeben werden. Eine Besonderheit des folgenden Beweises ist, dass die Konstruktion aus zwei Schritten besteht (Erweiterung der Eingabeformel, Konstruktion des Graphen). Beide Schritte werden in Abb. 27 illustriert.

BEWEIS:

- Zeige $\text{CLIQUE} \in \text{NP!}$.

Verfahren: Wähle nichtdeterministisch eine k -elementige Knotenmenge C . Prüfe, ob C eine Clique ist. Dies ist in polynomieller Laufzeit möglich, da wir für jedes Knotenpaar höchstens einmal die Menge der Kanten durchlaufen müssen.

[†]Für die Zwecke dieser Vorlesung ist es komfortabel, den ungerichteten Graphen als Spezialfall des gerichteten Graphen zu definieren. Eine häufig verwendete Alternative ist, die Knotenmenge als Menge von zweielementigen Mengen zu definieren.

- Zeige, dass CLIQUE **NP!**-schwer ist (mit Hilfe der Reduktion **3SAT!** $\preceq_p$ CLIQUE).

Ziel: Konstruiere für eine gegebene 3CNF-Formel F einen Graphen $\mathcal{G} = (V, E)$, sodass F genau dann erfüllbar ist, wenn $\mathcal{G}$ eine k -Clique hat.

Unsere Konstruktion besteht aus zwei Schritten. Gegeben sei eine 3CNF-Formel F mit m Konjunkten.

- Schritt 1.

Erweitere F , sodass jeder Konjunkt aus genau drei Disjunkten besteht. Wähle hierfür eine Äquivalenzumformung, die einfach nur existierende Disjunkte wiederholt. Die resultierende Formel F' hat also die folgende Form.

$$F' = \bigwedge_{i=1}^m (z_{i,1} \vee z_{i,2} \vee z_{i,3}), \text{ wobei } z_{i,j} \in \{A_1, \dots, A_n\} \cup \{\neg A_1, \dots, \neg A_n\}$$

- Schritt 2. Definiere zu einer Formel F mit m Konjunkten den Graphen $\mathcal{G} = (V, E)$ und k wie folgt:

$$\begin{aligned} V &= \{(i, j) \mid 1 \leq i \leq m, j \in \{1, 2, 3\}\} \\ E &= \{((i, j), (p, q)) \mid i \neq p, z_{i,j} \neq \neg z_{p,q}\} \\ k &= m \end{aligned}$$

Es gibt offensichtlich eine totale Reduktionsfunktion, welche diese Konstruktion in polynomieller Zeit berechnet. Nun gilt:

$$\begin{aligned} F \text{ ist erfüllbar} &\iff \text{Es gibt eine Folge } z_{1,j_1}, \dots, z_{m,j_m}, \text{ sodass } F \text{ unter der} \\ &\quad \text{Belegung, die jedem Folgenglied 1 zuordnet, erfüllt ist.} \\ &\iff \text{Es gibt eine Folge } z_{1,j_1}, \dots, z_{m,j_m}, \text{ sodass in jedem Konjunkt} \\ &\quad \text{ein } z_{i,j_i} \text{ vorkommt und } \forall i \neq p : z_{i,j_i} \neq \neg z_{p,j_p} \text{ gilt.} \\ &\iff \mathcal{G} \text{ hat eine Menge von Knoten } \{(1, j_1), \dots, (m, j_m)\}, \\ &\quad \text{die paarweise durch Kanten verbunden sind.} \\ &\iff \mathcal{G} \text{ hat eine } k\text{-Clique für } k = m. \end{aligned}$$

Damit haben wir **3SAT!** $\preceq_p$ CLIQUE gezeigt. □

Die Formel

$$(X \vee \neg Y) \wedge (Y \vee \neg X) \wedge (X \vee Y)$$

ist in 3CNF. Die folgende Formel ist äquivalent und jeder Konjunkt besteht aus drei Disjunkten.

$$\underbrace{(X \vee \neg Y \vee \neg Y)}_1 \wedge \underbrace{(Y \vee \neg X \vee \neg X)}_2 \wedge \underbrace{(X \vee X \vee Y)}_3$$

Die Belegung β definiert durch $\beta(X) = 1, \beta(Y) = 1$ ist eine erfüllende Belegung für diese Formel, da z.B. der erste Disjunkt im ersten Konjunkt, der erste Disjunkt im zweiten Konjunkt und der dritte Disjunkt im dritten Konjunkt auf 1 gesetzt sind. Im folgenden Graphen bilden die Knoten $(1, 1)$, $(2, 1)$ und $(3, 3)$ eine 3-Clique.

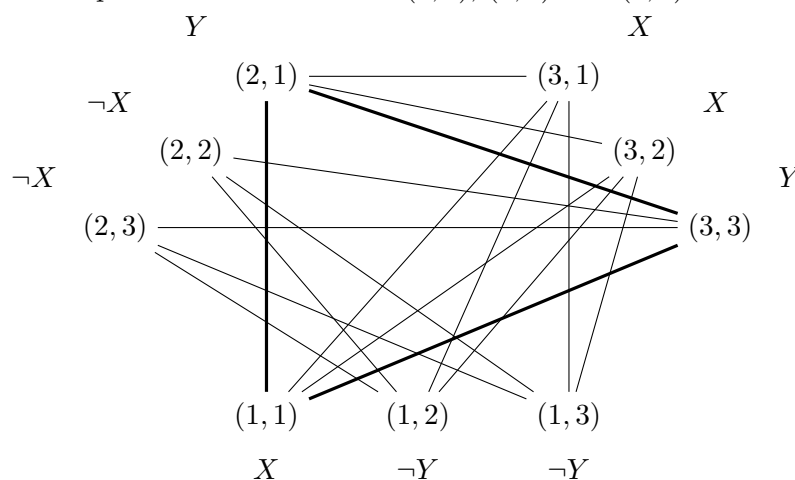


Abb. 27: Beispiel zu $3SAT! \preceq_p CLIQUE$

An dieser Stelle wollen wir nun den Beweis von Satz 8.5 nachholen. Wir benötigen dafür zunächst das folgende Lemma.

Vorlesung:
06.02.2019

Lemma 8.8: Für jedes $k \in \mathbb{N}$ existiert eine aussagenlogische Formel $G(A_1, \dots, A_k)$, sodass $|G(A_1, \dots, A_k)| \in \mathcal{O}(k^2)$ und

$$G(A_1, \dots, A_k) = 1 \text{ gdw. } \exists j \in \{1, \dots, k\} : A_j = 1 \text{ und } \forall i \in \{1, \dots, k\} : i \neq j \rightarrow A_i = 0$$

BEWEIS:

$$G(A_1, \dots, A_k) = \bigvee_{i=1}^k x_i \wedge \bigwedge_{i \neq j} \neg(A_i \wedge A_j)$$

□

BEWEIS (von Satz 8.5):

1. **SAT! $\in$ NP!**

Rate nichtdeterministisch eine Belegung β . Werte anschließend $F[\beta]$ in polynomialer Zeit aus. (Siehe Bsp. 8.2)

2. **SAT! ist NP!-schwer.**

Zeige: $\forall L \in \mathbf{NP!} : L \preceq_p \mathbf{SAT!}$

Sei $L \in \mathbf{NP!}$, d.h., es gibt ein Polynom p und eine NTM $\mathcal{M}$ mit $L = L(\mathcal{M})$ mit Zeitkomplexität $p(n)$ für $n = |w|$.

Sei $w = x_1 \dots x_n \in \Sigma^*$ eine beliebige Eingabe für $\mathcal{M}$ der Länge n .

Ziel: Definiere F , sodass F erfüllbar $\iff \mathcal{M}$ akzeptiert w .

Seien Q die Zustandsmenge von $\mathcal{M}$ mit $\{q_1, \dots, q_k\} = Q$ und Γ das Bandalphabet von $\mathcal{M}$ mit $\{a_1, \dots, a_l\} = \Gamma$.

Seien $t \in \{0, 1, \dots, p(n)\}$, $i \in \{-p(n), \dots, -1, 0, 1, \dots, p(n)\}$, $q \in Q$ und $a \in \Gamma$.

Definiere damit folgende aussagenlogische Variablen zur Verwendung in F .

- $\text{state}(t, q) = 1$ gdw. $\mathcal{M}$ nach t Schritten im Zustand q ist
- $\text{pos}(t, i) = 1$ gdw. der Kopf von $\mathcal{M}$ nach t Schritten auf Position i steht.
- $\text{tape}(t, i, a) = 1$ gdw. nach t Schritten an Position i ein a steht.

Wir setzen $F = R \wedge A \wedge T_1 \wedge T_2 \wedge E$, wobei die Teilformeln folgendermaßen definiert sind.

1. Randbedingungen

$$\begin{aligned} R = & \bigwedge_t G(\text{state}(t, q_1), \dots, \text{state}(t, q_k)) \\ & \wedge \bigwedge_t G(\text{pos}(t, -p(n)), \dots, \text{pos}(t, 0), \dots, \text{pos}(t, p(n))) \\ & \wedge \bigwedge_{t,i} G(\text{tape}(t, i, a_1), \dots, \text{tape}(t, i, a_l)) \end{aligned}$$

2. Anfangskonfiguration

$$\begin{aligned} A = & \text{state}(0, q_1) \wedge \text{pos}(0, 1) \\ & \wedge \text{tape}(0, 1, x_1) \wedge \dots \wedge \text{tape}(0, n, x_n) \\ & \wedge \bigwedge_{i < 1 \vee i > n} \text{tape}(0, i, \sqcup) \end{aligned}$$

3. Transitionsschritte

$$\begin{aligned}
T_1 &= \bigwedge_{\substack{t < p(n), \\ i, q}} \text{state}(t, q) \wedge \text{pos}(t, i) \wedge \text{tape}(t, i, a) \\
&\rightarrow \bigvee_{1 \leq m \leq |\delta(q, a)|} \text{state}(t+1, q'_m) \wedge \text{pos}(t+1, i+d_m) \wedge \text{tape}(t+1, i, a'_m) \\
&\quad \text{für jede Transition } (q'_m, a'_m, d_m) \in \delta(q, a) \text{ mit } d_m \in \{-1, 0, 1\} \\
T_2 &= \bigwedge_{\substack{t < p(n), \\ i, q}} \neg \text{pos}(t, i) \wedge \text{tape}(t, i, a) \rightarrow \text{tape}(t+1, i, a)
\end{aligned}$$

4. Endkonfiguration

$$E = \bigvee_{q \in F} \text{state}(p(n), q)$$

$|F|$ ist polynomiell beschränkt in $|\mathcal{M}| + |w|$, also gilt $L \preceq_p \mathbf{SAT!}^\dagger$.

Nach Konstruktion gilt: F erfüllbar $\iff \mathcal{M}$ akzeptiert w .

Damit haben wir „**SAT!** ist **NP!**-vollständig“ gezeigt. □

[†] $|\mathcal{M}|$ dürfen wir als zu L gehörende Konstante betrachten.

9 Einordnung von Sprachen in Chomsky-Hierarchie und Abschlusseigenschaften

Wir haben bereits gesehen, dass jede Laufzeitschranke auch eine Beschränkung der Platzkomplexität impliziert. Wir werden nun sehen, dass zumindest für haltende DTMs auch die umgekehrte Richtung gilt.

Lemma 9.1: Sei $\mathcal{M}$ eine TM mit Platzkomplexität $s(n)$. Dann durchläuft jede Berechnung höchstens $s(n) \cdot |Q| \cdot |\Gamma|^{s(n)}$ verschiedene Konfigurationen.

BEWEIS: Es gibt

- höchstens $s(n)$ mögliche Positionen, an denen sich der Kopf befinden kann,
- höchstens $|Q|$ mögliche Zustände und
- jede Zelle des besuchten Bands (davon gibt es höchstens $s(n)$ Stück) kann mit höchstens $|\Gamma|$ verschiedenen Zeichen beschriftet sein. $\square$

Bemerkung: Für jede DTM mit Platzkomplexität $s(n)$ hält also jede Berechnung entweder

- nie oder
- nach höchstens $s(n) \cdot |Q| \cdot |\Gamma|^{s(n)}$ Schritten.

Grund: Der zweite Fall folgt aus Lemma 9.1. Ansonsten wird eine Konfiguration zweimal erreicht und die DTM ist in einer „Endlosschleife“.

Satz 9.2: Jede Sprache $L \in \text{NSPACE}(n)$ ist entscheidbar.

BEWEIS: Wegen Lemma 9.1 genügt es, jede Berechnung bis zu $s(n) \cdot |Q| \cdot |\Gamma|^{s(n)}$ Schritten zu simulieren, um herauszufinden, ob ein Wort der Länge n akzeptiert wird. $\square$

Satz 9.3: $\text{CH1} = \text{NSPACE}(n)$

BEWEIS:

„ $\Rightarrow$ “: Gegeben: Typ-1-Grammatik $\mathcal{G} = (\Sigma, N, P, S)$

Gesucht: NTM $\mathcal{M}$ mit $L(\mathcal{M}) = L(\mathcal{G})$

Wähle das Bandalphabet $\Gamma = \Sigma \cup N \cup \{\sqcup\}$ und folgendes Verhalten:

1. $\mathcal{M}$ rät nichtdeterministisch eine Position auf dem Band und eine Produktion $\alpha \rightarrow \beta$. Falls β gefunden wird, ersetze durch α , weiter bei 1.

2. Falls der Bandinhalt nur noch S ist, so hält $\mathcal{M}$ und akzeptiert.

Dieses Verfahren terminiert.

„ $\Leftarrow$ “: Gegeben: NTM $\mathcal{M}$

Gesucht: Typ-1-Grammatik $\mathcal{G}$ mit $L(\mathcal{G}) = L(\mathcal{M})$

Wir wollen den Beweis hier nicht formal ausformulieren und beschreiben nur die zugrundeliegenden Ideen.

- Idee 1: Ahme die Berechnungsschritte von $\mathcal{M}$ mit Ableitungsschritten der Grammatik nach. Die manipulierten Satzformen repräsentieren dabei die Konfigurationen von $\mathcal{M}$. Wir wählen als (vorläufige) Variablenmenge $N' = \Gamma \cup (Q \times \Gamma)$. Eine Konfiguration $uqav$ mit $u, v \in \Gamma^*$, $a \in \Gamma$ wird durch die Satzform $u(q, a)v$ repräsentiert. Die Regeln sind wie folgt definiert.

$$\begin{aligned} P' = & \{(q, a) \rightarrow (q', a') \mid (q', a', N) \in \delta(q, a)\} \\ & \cup \{(q, a)b \rightarrow a'(q', b) \mid b \in \Gamma, (q', a', R) \in \delta(q, a)\} \\ & \cup \{(q, a) \rightarrow (q', a') \mid (q', a', L) \in \delta(q, a)\} \end{aligned}$$

Somit gilt $uqav \vdash^* u'q'a'v'$, also dass $\mathcal{M}$ eine Konfiguration in eine andere überführen kann, genau dann, wenn wir mit Regeln aus P' die Ableitung $u(q, a)v \vdash_{\mathcal{G}}^* u'(q', a')v'$ machen können.

- Idee 2: Die erste Idee hat noch das folgende Problem und muss deshalb ergänzt werden: Während eine TM mit der Eingabe startet und diese in eine Antwort der Form „akzeptiert“/„akzeptiert nicht“ transformiert, arbeitet eine Grammatik umgekehrt. Sie startet mit dem Startsymbol und am Ende erhalten wir das resultierende Wort.

Wir unterteilen die Satzformen der Grammatik in zwei Spuren, eine obere Spur und eine untere Spur. Die untere Spur enthält das Eingabewort, die obere Spur simuliert das Band der TM. Formal erreichen wir dies, indem wir als Variablenmenge das kartesische Produkt aus vorläufiger Variablenmenge und Alphabet nehmen: $N = \{S\} \dot{\cup} N' \times \Sigma$

$$a_1 \cdots a_n \longrightarrow \begin{pmatrix} a_1 \\ a_1 \end{pmatrix} \begin{pmatrix} a_2 \\ a_2 \end{pmatrix} \begin{pmatrix} (q, a) \\ a_3 \end{pmatrix} \begin{pmatrix} a_4 \\ a_4 \end{pmatrix} \begin{pmatrix} a_n \\ a_n \end{pmatrix} \quad \begin{array}{l} \text{Spur 1} \\ \text{Spur 2} \end{array}$$

Eine Ableitung besteht dann aus drei Phasen.

- * Phase 1: Wir wählen nichtdeterministisch das Wort, das wir erzeugen wollen (untere Spur), und bringen das Band der TM (obere Spur) in die entsprechende Startkonfiguration.

Entsprechende Regeln:

$$\begin{aligned} S &\rightarrow \begin{pmatrix} (q_0, a) \\ a \end{pmatrix} & \forall a \in \Sigma \\ S &\rightarrow S \begin{pmatrix} a \\ a \end{pmatrix} & \forall a \in \Sigma \end{aligned}$$

- * Phase 2: Wir simulieren auf der oberen Spur die Berechnungen der TM, bis eine Haltekonfiguration erreicht ist.

Entsprechende Regeln:

$$\begin{aligned} \begin{pmatrix} \alpha \\ a \end{pmatrix} &\rightarrow \begin{pmatrix} \beta \\ a \end{pmatrix} & \forall \alpha \rightarrow \beta \in P' \\ & & \alpha, \beta \in \Delta \\ \begin{pmatrix} \alpha_1 \\ a_1 \end{pmatrix} \begin{pmatrix} \alpha_2 \\ a_2 \end{pmatrix} &\rightarrow \begin{pmatrix} \beta_1 \\ a_1 \end{pmatrix} \begin{pmatrix} \beta_2 \\ a_2 \end{pmatrix} & \forall \alpha_1 \alpha_2 \rightarrow \beta_1 \beta_2 \in P' \\ & & \alpha_i, \beta_i \in \Delta \end{aligned}$$

- * Phase 3: Wenn die TM in einem akzeptierenden Zustand ist, ersetzen wir alle „Zweispurzeichen“ durch das entsprechende Zeichen der unteren Spur und erhalten das Eingabewort.

Entsprechende Regeln:

$$\begin{aligned} \begin{pmatrix} x \\ a \end{pmatrix} &\rightarrow a & x \in \Gamma \\ & & a \in \Sigma \\ \begin{pmatrix} (q', x) \\ a \end{pmatrix} &\rightarrow a & x \in \Gamma, q' \in F, \delta(q', x) = \emptyset \\ & & a \in \Sigma \end{aligned} \quad \square$$

Aus Satz 9.2 und Satz 9.2 folgt direkt:

Korollar 9.4: Jede Sprache $L \in \text{CH1}$ ist entscheidbar. $\oplus$

Bemerkung: Alternativ hätten wir die Aussage von Korollar 9.4 auch ohne Satz 9.2 und Satz 9.2 wie folgt beweisen können.

Seien $\mathcal{G} = (\Sigma, N, P, S)$ eine Typ-1-Grammatik (kontextsensitive Grammatik) für L , $w \in \Sigma^*$ und n die Länge von w . Da alle Regeln expansiv (d.h., $\alpha \rightarrow \beta \in P \implies |\alpha| \leq |\beta|$) sind, müssen wir nicht alle (unendlich viele!) Ableitungen betrachten. Es genügt, wenn wir Ableitungen

$$S \vdash \alpha_1 \vdash \dots \vdash \alpha_m,$$

betrachten, bei denen alle Satzformen α_j höchstens Länge n haben (also $|\alpha_j| \leq n$) und paarweise verschieden sind.

Satz 9.5: Die Menge der entscheidbaren Sprachen ist verschieden von CH1.

Vorlesung:
08.02.2019

BEWEIS: Wir konstruieren zunächst analog zu Abschnitt 6.5.1 eine Codierung von Grammatiken durch Wörter $w \in \{0, 1\}^*$. Anschließend definieren wir $\mathcal{G}_w$ als die Grammatik, die durch Wort w codiert wird. Dabei wählen wir die Grammatik $(\{S\}, \{0, 1\}, \{\}, S)$ für den Fall, dass w nicht Code einer Grammatik ist.[†]

Wir betrachten nun eine Matrix, in deren Spalten alle Wörter $w_1, w_2, \dots$ und in deren Zeilen alle Grammatiken $\mathcal{G}_{w_1}, \mathcal{G}_{w_2}, \dots$ aufgelistet sind. Ein Matrixeintrag $(\mathcal{G}_{w_i}, w_j)$ enthält ein Y, falls $w_j \in L(\mathcal{G}_{w_i})$, und sonst ein N.

Da das Wortproblem für CH1-Sprachen entscheidbar ist, ist insbesondere der Test „ $w_i \in L(\mathcal{G}_{w_i})$ “ für alle Diagonaleinträge berechenbar und somit auch die Sprache $L_{\text{evil}} = \{w \in \{0, 1\}^* \mid w \notin L(\mathcal{G}_w)\}$ entscheidbar.

Angenommen, jede entscheidbare Sprache sei CH1. Dann gibt es auch ein w_k , sodass $L(\mathcal{G}_{w_k}) = L_{\text{evil}}$. Nun gilt aber $w_k \in L_{\text{evil}} \iff w_k \notin L_{\text{evil}}$. Widerspruch! $\square$

	w_0	$\dots$	w_i	$\dots$	w_k	$\dots$	w_j	$\dots$
$\mathcal{G}_{w_0}$	Y	$\dots$	Y	$\dots$	Y	$\dots$	N	$\dots$
$\vdots$								
$\mathcal{G}_{w_i}$	N	$\dots$	N	$\dots$	Y	$\dots$	Y	$\dots$
$\vdots$								
$\mathcal{G}_{w_k}$	N	$\dots$	Y	$\dots$	?	$\dots$	Y	$\dots$
$\vdots$								
$\mathcal{G}_{w_j}$	Y	$\dots$	Y	$\dots$	N	$\dots$	N	$\dots$
$\vdots$								

Abb. 28: Illustration des Diagonalarguments beim Beweis, dass nicht jede entscheidbare Sprache CH1 ist

Satz 9.6: Die Typ-1-Sprachen sind abgeschlossen unter $\cup$, $\cap$, $\cdot$, $*$ und Komplement.

BEWEIS:

- „ $\cup$ “ und „ $\cap$ “: Betrachte eine NTM. Seien L_1 und L_2 zwei Typ-1-Sprachen und $\mathcal{M}_1$ und $\mathcal{M}_2$ zwei zugehörige NTMs.

[†]Diese Wahl ist wie bei der Definition von $\mathcal{M}_w$ willkürlich.

- Verwende als Bandalphabet 2-Tupel, um eine 2-Spur-TM zu simulieren.
- Verwende die untere Spur für eine „Sicherungskopie“ der Eingabe.
- Simuliere auf der oberen Spur $\mathcal{M}_1$ und speichere das Ergebnis im Zustand.
- Simuliere auf der unteren Spur $\mathcal{M}_2$ und werte das Gesamtergebnis aus.

- „ $\cdot$ “ und „ $*$ “:

Konstruiere eine Typ-1-Grammatik analog zu Satz 5.5.

- Komplement:

„2. LBA-Problem[†]“ bis 1987, dann gelöst durch Immerman und Szelepcsényi. $\square$

Bemerkung: „1. LBA-Problem (1964)“: Ist $\text{NSPACE}(n) = \text{DSpace}(n)$? Bisher ungelöst.

Satz 9.7: CH0 ist die Menge der semi-entscheidbaren Sprachen.

BEWEIS: “ $\Rightarrow$ “ Verwende die Konstruktion einer NTM $\mathcal{M}$ wie in Satz 9.3, aber ohne Platzbeschränkung.

“ $\Leftarrow$ “ Konstruktion analog zu Satz 9.3 + Startsymbol S'

$$S' \rightarrow \begin{pmatrix} \sqcup \\ \varepsilon \end{pmatrix} S' \begin{pmatrix} \sqcup \\ \varepsilon \end{pmatrix} \quad \text{Schaffe Platz für Berechnung von } \mathcal{M}$$

$$S' \rightarrow S$$

Erweitere N

$$= \{S', S\} \cup \Delta \times (\Sigma \cup \{\varepsilon\})$$

Neue Löschregeln:

$$\begin{pmatrix} x \\ \varepsilon \end{pmatrix} \rightarrow \varepsilon \quad \forall x \in \Gamma$$

$\vdash$ die einzigen Regeln, die Typ-1-Bedingung verletzen. $\square$

Satz 9.8: Die entscheidbaren Sprachen sind eine echte Teilmenge von CH0.

BEWEIS:

- „ $\subseteq$ “: Folgt trivialerweise aus Def. 7.7, Def. 6.4 und Def. 7.4.

[†]LBA = Linear Bounded Automaton – 1964 Kuroda

- „ $\neq$ “: Das spezielle Halteproblem ist nicht entscheidbar (Satz 7.8), aber semi-entscheidbar (Lemma 7.10). $\square$

Satz 9.9: $CH0 \neq \mathcal{P}(\Sigma^*)$

BEWEIS: Nach Korollar 7.12 ist $\overline{K}$ nicht semi-entscheidbar. $\square$

Satz 9.10: Die Typ-0 Sprachen sind unter $\cup$, $\cap$, $\cdot$, $*$ abgeschlossen, sind aber nicht unter Komplementbildung abgeschlossen.

BEWEIS:

- „ $\cup$ “ und „ $\cap$ “:

Siehe Satz 7.21.

- „ $\cdot$ “ und „ $*$ “:

Konstruiere Grammatik analog zu Satz 5.5.

- Komplement:

Das spezielle Halteproblem K ist semi-entscheidbar (Lemma 7.10), aber sein Komplement $\overline{K}$ ist nicht semi-entscheidbar (Korollar 7.12). $\square$

		$\cap$	$\cup$	$-$	$\cdot$	$*$
regulär	CH3	ja	ja	ja	ja	ja
kontextfrei	CH2	nein	ja	nein	ja	ja
kontextsensitiv	CH1	ja	ja	ja [♣]	ja	ja
entscheidbar	-	ja	ja	ja	ja	ja
semi-entscheidbar	CH0	ja	ja	nein	ja	ja

Abb. 29: Abschlusseigenschaften der Sprachklassen aus der Chomsky-Hierarchie sowie der Klasse der Entscheidbaren Sprachen

Liste der Definitionen

1.1	Def. (Alphabet Σ)	3
1.2	Def. (Wort w über Σ)	3
1.3	Def. (Konkatenation von Wörtern)	4
1.4	Def.	4
1.5	Def. (Sprache über Σ)	4
1.6	Def. (Konkatenation und Potenzierung von Sprachen)	5
1.7	Def. (Kleene-Abschluss, Kleene-Stern, Kleene-Plus)	5
2.1	Def. (DEA)	9
2.2	Def. (Induktive Erweiterung von δ auf Wörter)	9
2.3	Def. (Die durch einen DEA akzeptierte Sprache)	10
2.4	Def. (Produktautomat für Schnitt)	10
2.5	Def.	12
2.6	Def. (Äquivalenz von DEA-Zuständen)	13
2.7	Def.	14
2.8	Def. (Äquivalenzklassenautomat)	15
2.9	Def. (Rechtskongruente Äquivalenzrelation)	16
2.10	Def.	17
2.11	Def.	18
2.12	Def. (NEA)	22
2.13	Def. (Lauf eines Automaten)	22
2.14	Def. (NEA zu DEA)	22
2.15	Def. (Potenzmengenautomat)	23
2.16	Def. (ε -NEA)	28
2.17	Def.	28
2.18	Def.	29
2.19	Def.	29
2.20	Def. (ε -freier Automat)	30
2.21	Def.	31
2.22	Def. (Abgeschlossenheit von $\mathcal{L}$)	32
2.23	Def. ($\text{RE}(\Sigma)$)	34
2.24	Def. (Semantik eines regulären Ausdrucks)	34
2.25	Def.	36
3.1	Def.	40
3.2	Def. (Ableitungsrelation, Ableitung, Sprache einer Grammatik)	40
3.3	Def. (Chomsky-Hierarchie)	42
3.4	Def. (Ableitungsbaum)	44
3.5	Def. (Eindeutigkeit von CFG und CFL)	45
3.6	Def.	45

3.7	Def.	46
3.8	Def.	49
3.9	Def. (CFG in CNF)	52
3.10	Def.	56
4.1	Def. (NPDA)	59
4.2	Def. (Menge der Konfigurationen eines NPDA)	60
5.1	Def.	70
5.2	Def. (DPDA)	70
5.3	Def.	71
6.1	Def. (Turing-Maschine (TM))	75
6.2	Def. (Konfiguration)	76
6.3	Def. (Rechenschrittrelation)	77
6.4	Def. (Akzeptanz)	77
6.5	Def. (Halten)	77
6.6	Def. (berechnete Funktion)	78
6.7	Def. (Flussdiagramm)	79
6.8	Def. (Von Flussdiagramm definierte TM)	80
6.9	Def. (normierte TM)	86
7.1	Def. (berechenbare Funktion)	90
7.2	Def. (Abzählbar, Überabzählbar)	91
7.3	Def. (Durch Binärstring codierte TM)	92
7.4	Def. (Entscheidbarkeit)	96
7.5	Def. (Charakteristische Funktion)	97
7.6	Def. (Spezielles Halteproblem)	98
7.7	Def. (Semi-Entscheidbarkeit)	99
7.8	Def. (Reduktion)	100
7.9	Def. (Halteproblem)	101
7.10	Def. (Halteproblem auf leerem Band H_ϵ)	101
7.11	Def. (Universelles Halteproblem H_{vw})	103
7.12	Def. (Berechnung)	104
8.1	Def.	107
8.2	Def. (NTIME Klasse)	107
8.3	Def.	108
8.4	Def.	109
8.5	Def.	109
8.6	Def. (Polynom)	111
8.7	Def.	111
8.8	Def. (Polynomielle Reduktion)	112
8.9	Def. (3SAT)	112
8.10	Def. (NP! -schwer und NP! -vollständig)	115
8.11	Def.	116

8.12	Def. (CLIQUE)	117
------	---------------	-----

Liste der Sätze

2.1	Satz	11
2.2	Lemma ($\equiv$ ist Äquivalenzrelation)	14
2.3	Lemma	15
2.4	Satz (Äquivalenzklassenautomat ist wohldefiniert)	16
2.5	Satz (Myhill und Nerode)	19
2.5	Korollar	21
2.6	Satz (Rabin und Scott)	23
2.7	Lemma (Pumping Lemma)	25
2.8	Satz	29
2.9	Lemma	31
2.10	Lemma	32
2.11	Lemma	32
2.12	Satz (Kleene)	35
2.13	Lemma (Ardens Lemma)	37
2.14	Korollar	38
3.1	Lemma	44
3.2	Lemma (SEP)	45
3.3	Lemma (BIN)	46
3.4	Satz	47
3.5	Korollar	48
3.6	Lemma (DEL)	48
3.7	Lemma (UNIT)	49
3.8	Satz	52
3.9	Lemma	52
3.10	Satz (Wortproblem für CFL entscheidbar)	53
3.11	Korollar	55
3.12	Satz (Entscheidbarkeit des Leerheitsproblems für kontextfreie Sprachen)	56
3.13	Satz (Typ-3-Sprache ist regulär)	56
3.14	Lemma	57
4.1	Lemma	61
4.2	Lemma (Mehr Keller – mehr Möglichkeiten.)	61
4.3	Lemma	63
4.4	Lemma	63
4.5	Satz	66
5.1	Satz (Pumping Lemma für CFL)	67
5.2	Lemma	67

5.3	Lemma	68
5.4	Lemma	69
5.5	Satz	69
5.6	Satz	69
5.7	Satz	70
5.8	Lemma	70
5.9	Lemma (DPDA, der gesamte Eingabe verarbeitet)	70
5.10	Satz (Abgeschlossenheit der deterministischen CFL)	72
5.11	Satz	72
6.1	Lemma (normierte TM)	86
6.2	Satz (Universelle TM)	89
7.1	Lemma (Standardnummerierung stdnum)	92
7.2	Lemma (bin2tm)	93
7.3	Lemma (func2power)	93
7.4	Satz (Cantor)	93
7.5	Satz (nicht berechenbare Funktionen)	93
7.6	Satz (nicht entscheidbare Sprache)	96
7.7	Satz (Berechenbarkeit und Entscheidbarkeit)	97
7.8	Satz (spezielles Halteproblem ist unentscheidbar)	98
7.9	Korollar (Komplement des speziellen Halteproblems ist unentscheidbar)	99
7.10	Lemma (spezielles Halteproblems ist semi-entscheidbar)	100
7.11	Satz ($L, \bar{L}$ semi-entscheidbar $\Rightarrow L$ entscheidbar)	100
7.12	Korollar (Komplement des speziellen Halteproblems ist semi-entscheidbar)	100
7.13	Lemma (Reduktion)	101
7.14	Satz (H ist unentscheidbar)	101
7.15	Satz (H ist semi-entscheidbar)	101
7.16	Satz (H_ϵ ist unentscheidbar)	101
7.17	Satz	102
7.18	Satz (Satz von Rice)	102
7.19	Satz (Universelles Halteproblem $H_{\forall w}$)	103
7.20	Satz	103
7.21	Satz	104
7.22	Satz	105
8.1	Lemma	113
8.2	Lemma	115
8.3	Satz	115
8.4	Lemma ($\preceq_p$ ist reflexiv und transitiv)	116
8.5	Satz (Cook, Levin)	116
8.6	Satz (3SAT ist NP! -vollständig)	116
8.7	Satz (CLIQUE ist NP! -vollständig)	117
8.8	Lemma	119

9.1	Lemma	122
9.2	Satz	122
9.3	Satz	122
9.4	Korollar	124
9.5	Satz	125
9.6	Satz	125
9.7	Satz	126
9.8	Satz	126
9.9	Satz	127
9.10	Satz (Abgeschlossenheit von Typ-0 Sprachen)	127

Abbildungsverzeichnis

1	Endliches Band	7
2	Schematische Darstellung eines DEA, mit $\tilde{\delta}(q^{\text{init}}, u) = \tilde{\delta}(q^{\text{init}}, v) = q_{42}$. . .	17
3	Nichtdeterministischer Automat für L_n	21
4	DEA „modulo 3“	37
5	Algorithmus UNIT	50
6	CYK Algorithmus	54
7	Zustandsdiagramm eines Kellerautomaten	58
8	Zustandsdiagramm der Kellerautomaten aus Abb. 7. Statt alle Transitionen explizit zu zeichnen verwenden wir die Variablen a und Z um eine Menge von Transitionen durch eine einzige Kante zu repräsentieren. . . .	60
9	Illustration des Argumentes aus dem Beweis von Satz 5.1	68
10	Turingband	73
11	Informelle graphische Darstellung einer Turingmaschinenkonfiguration . .	76
12	Durch Flussdiagramm beschriebene TM	81
13	Mehrspurmaschine	82
14	Simulation einer Mehrbandmaschine durch eine Mehrspurmaschine	83
15	Startkonfiguration einer Mehrbandmaschine codiert auf Mehrspurmaschine	84
16	Standardnummerierung für einige Beispielwerte.	92
17	Abbildung func2power für einige Beispielwerte	93
18	Konkatenierte Abbildung aus dem Widerspruchsbeweis zu Satz 7.5	94
19	Illustration des Diagonalarguments beim Beweis des Satzes von Cantor . .	95
20	Illustration des Diagonalarguments beim Rätsel des Barbier von Waldkirch	96
21	Konkatenierte Abbildung aus dem Widerspruchsbeweis zu Satz 7.6	97
22	Definition der TM $\mathcal{M}_{\text{evil}}$ aus dem Beweis von Satz 7.8.	98
23	Illustration des Diagonalarguments beim Beweis der Unentscheidbarkeit des speziellen Halteproblems	99
24	Illustration einer Reduktion $U \preceq V$ von einer Sprache U auf eine Sprache V	100

25	Semi-Entscheidbarkeit der in diesem Kapitel vorgestellten unentscheidbaren Sprachen	104
26	Inklusionsrelationen zwischen Komplexitätsklassen	108
27	Beispiel zu 3SAT ! $\preceq_p$ CLIQUE	119
28	Illustration des Diagonalarguments beim Beweis, dass nicht jede entscheidbare Sprache CH1 ist	125
29	Abschlusseigenschaften der Sprachklassen aus der Chomsky-Hierarchie sowie der Klasse der Entscheidbaren Sprachen	127

Abkürzungsverzeichnis

AL	Aussagenlogik
CFL	Menge der kontextfreien Sprachen
CFG	Menge der kontextfreien Grammatiken
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

Vorlesung: 19.10.2018	3
Vorlesung: 20.10.17	7
Vorlesung: 24.10.18	10
Vorlesung: 26.10.18	14
Vorlesung: 31.10.18	18
Vorlesung: 02.11.18	20
Vorlesung: 7.11.18	24
Vorlesung: 9.11.18	26
Vorlesung: 14.11.18	29
Vorlesung: 16.11.18	33
Vorlesung: 21.11.18	37
Vorlesung: 23.11.18	38
Vorlesung: 28.11.18	41
Vorlesung: 30.11.2018	43
Vorlesung: 5.12.2018	47
Vorlesung: 7.12.2018	49
Vorlesung: 12.12.2018	55
Vorlesung: 12.12.2018	60
Vorlesung: 09.01.2019	73
Vorlesung: 11.01.2019	80
Vorlesung: 16.01.2019	84
Vorlesung: 18.01.2019	91
Vorlesung: 23.01.2019	96

Vorlesung: 25.01.2019	100
Vorlesung: 30.01.2019	103
Vorlesung: 01.02.2019	110
Vorlesung: 06.02.2019	119
Vorlesung: 08.02.2019	125