Automaten und Berechenbarkeit

Modul	Automaten und Berechenbarkeit
Dozent	Dr. Jörg Vogel
Semester	Wintersemester 2019/2020

Dieses Dokument (Stand 18. Februar 2022) stellt eine studentische Mitschrift während der Veranstaltungen zu oben aufgeführtem Modul dar. Obwohl es nach bestem Wissen und Gewissen angefertigt worden ist, können Fehler enthalten sein, welche Sie gerne als Issue im Git-Projekt vermerken können. Seit 2021 wird dieses Skript durch den FSR Informatik betreut und aktualisiert. Änderungswünsche und Anregungen an fsrinfo@uni-jena.de.

Inhaltsverzeichnis

1 Einführung 3

1.1 Grundlegende Fragestellungen der Theoretischen Informatik 3

Automaten 3

Berechenbarkeit 5

Berechnungsprobleme versus Entscheidungsprobleme 6

Was hat Theoretische Informatik mit Wirklichkeit zu tun? 7

Was bedeutet effizient? 8

1.2 Alphabete, Wörter und Sprachen 9

Operationen für Wörter 11

Relationen für Wörter 11

Potenzen von Wörtern 12

Binäre Nummern und Binärdarstellung 12

Kanonische Ordnung und dyadische Darstellung . . . 13

Formale Sprachen 15

2 Reguläre Sprachen 16

2.1 Deterministische Endliche Automaten 16

Konfigurationen eines Automaten 18

Akzeptierte Sprachen 20

Zustandsklassen von endlichen Automaten 21

2.2 Mengenoperationen für reguläre Sprachen 24

2.3 Nichtdeterministische endliche Automaten 26

2.4 Reguläre Operationen und reguläre Ausdrücke 31

2.5 Das Pumping-Lemma für reguläre Sprachen 34

Beweise für Nicht-Regularität 36

Klammersprachen 38

„Rückrichtung“ des Pumping-Lemmas 38

2.6 Satz von Myhill und Nerode 40

3 Die Chomsky-Hierarchie 45

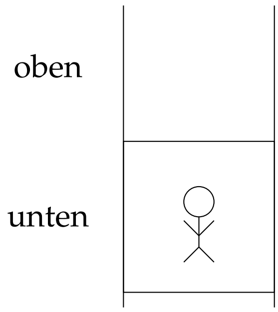
3.1 Formale Grammatiken . . . 45
3.2 Chomsky-Grammatiken vom Typ 1 . . . 49
3.3 Kontextsensitive Grammatiken . . . 52
3.4 Kontextfreie Grammatiken . . . 55
3.5 Grammatiken von Chomsky-Typ 3 . . . 68
Rechtslineare Grammatiken . . . 68
Linkslineare Grammatiken . . . 71
4 Berechenbarkeit . . . 73
4.1 Turingmaschinen . . . 73
Einführung in Turingmaschinen . . . 73
Techniken zur Konstruktion von Turingmaschinen . 78
4.2 Typen von Turingmaschinen . . . 81
Turingmaschinen mit Halbband . . . 81
Linear beschränkte Automaten . . . 81
Turingmaschinen mit mehreren Spuren . . . 82
Turingmaschinen mit mehreren Bändern . . . 84
Weitere Varianten . . . 86
Turingmaschine als Erkenner bzw. als Akzeptoren . . . 87
Darstellung von Turingmaschinen . . . 89
4.3 Entscheidbarkeit und Aufzählbarkeit . . . 89
4.4 Gödelisierung von Turingmaschinen . . . 102
Codierung von Standard-Turingmaschinen . . . 102
Das Halteproblem . . . 104
4.5 Reduzierbarkeit . . . 108
5 Einschübe und Exkurse . . . 112
5.1 Ein Wörterbuch der Algorithmen . . . 112
5.2 Literaturhinweise . . . 114

1 Einführung

1.1 Grundlegende Fragestellungen der Theoretischen Informatik

Automaten

Beispiel 1 Fahrstuhl (nicht formal)



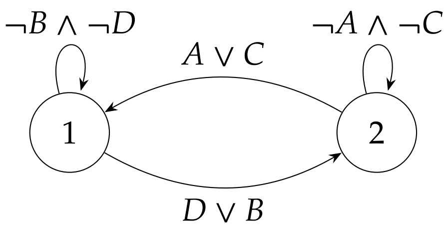
Zwei Zustände:

1. Fahrstuhl ist oben
2. Fahrstuhl ist unten

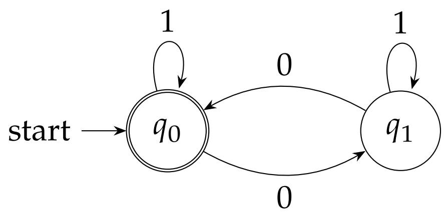
Vier Eingaben:

- im Fahrstuhl will jemand nach oben
- im Fahrstuhl will jemand nach unten
- oben wartet jemand
- unten wartet jemand

Darstellung mit einem Transitionsdiagramm:



Beispiel 2 Eingaben sind Wörter über einem Alphabet. Ein Standard-Alphabet $\Sigma = \{0, 1\}$.



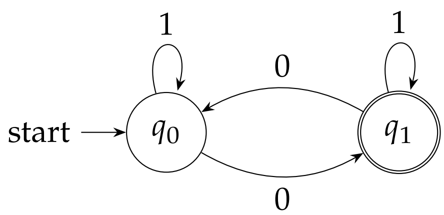
Wir wählen beispielhaft $w = 0110100$. Dieser Automat akzeptiert diese Eingabe.

Frage: Welche Eingaben akzeptiert dieser Automat? Sprache $L = \{w \in \{0,1\}^* \mid |w|_0 \equiv 0 \pmod 2\}$.

Definition 1 Es sei Σ ein *Alphabet* (eine endliche, nicht leere Menge). $a \in \Sigma$ sei ein *Buchstabe* aus dem Alphabet, $w \in \Sigma^*$ sei ein *Wort* über diesem Alphabet.

$|w|_a$ bezeichnet die Anzahl der Vorkommen des Buchstaben a in w. $|w| = \sum_{a \in \Sigma} |w|_a$ bezeichnet die *Länge* des Wortes w.

Beispiel 3 Eine Variante von Beispiel 2:



Sprache $L' = \{w \in \{0,1\}^* \mid |w|_0 \equiv 1 \pmod 2\}$. Dabei ist $L \cup L' = \{0,1\}^*$.

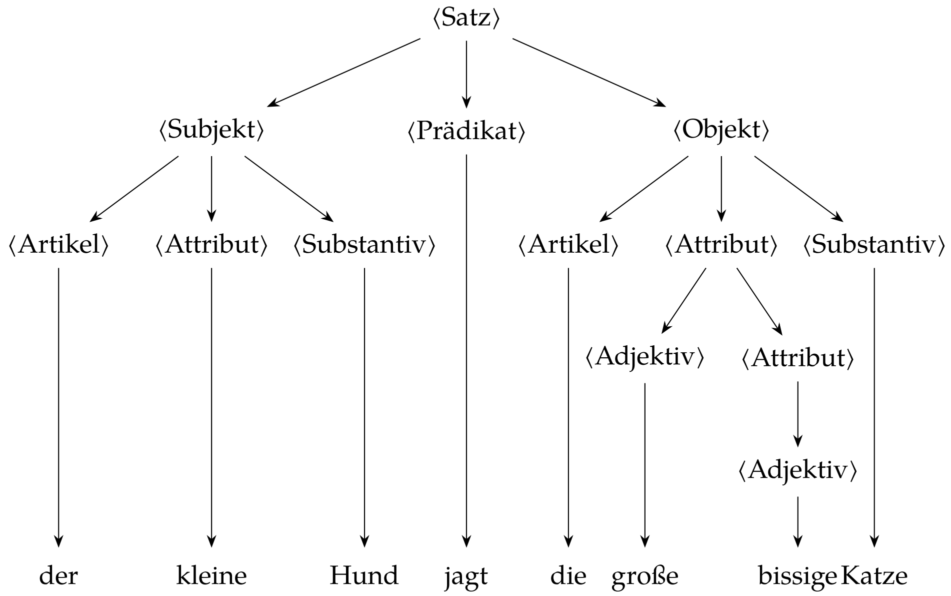
Frage: Wie können (formale) Sprachen erzeugt werden? Sprachen werden durch Grammatiken erzeugt.

Beispiel 4

⟨Satz⟩	→ ⟨Subjekt⟩⟨Prädikat⟩⟨Objekt⟩
⟨Subjekt⟩	→ ⟨Artikel⟩⟨Attribut⟩⟨Substantiv⟩
⟨Artikel⟩	→ λ | der | die | das
⟨Attribut⟩	→ λ | ⟨Adjektiv⟩ | ⟨Adjektiv⟩⟨Attribut⟩
⟨Adjektiv⟩	→ kleine | große | bissige
⟨Substantiv⟩	→ Hund | Katze
⟨Prädikat⟩	→ jagt
⟨Objekt⟩	→ ⟨Artikel⟩⟨Attribut⟩⟨Substantiv⟩

Ein Beispiel eines Ableitungsbaumes zu dieser Grammatik ist in Abbildung 1 abgebildet. Dies ist ein Beispiel eines Satzes (eines Wortes), der durch diese Grammatik erzeugt wird.

Abbildung 1 Beispielhafte Ableitung eines Satzes aus der Grammatik aus Beispiel 4



Diese Grammatik erzeugt unendlich viele Sätze der deutschen Sprache (da ⟨Attribut⟩ eine Rekursion ermöglicht).

Wir betrachten zwei Sorten von Symbolen:

- in Klammern, z. B. ⟨Satz⟩: Metasymbol, Variable, *Nichtterminal*
- nicht in Klammern, z. B. der: Zeichen, Konstante, *Terminal*

Eine Grammatik benötigt

i. Nichtterminale (werden ersetzt)
ii. Terminale (bleiben unverändert)
iii. Startsymbol aus der Menge der Nichtterminale
iv. Regeln (endlich viele)

Berechenbarkeit

Beispiel 5 Berechnungsproblem: Ist jede beliebige (Zahlen-)Funktion durch einen Algorithmus berechenbar? Lassen sich spezielle Funktionen nur mit besonderen Programmiersprachen berechnen?

Ganzzahlige Division durch 7. Gegeben ist $n \in \mathbb{N}$ in Dezimaldarstellung. Aufgabe: Bestimme $\lfloor \frac{n}{7} \rfloor$.

Auf diese Weise wird eine Funktion f berechnet: $f(n) = \lfloor \frac{n}{7} \rfloor$. Beispiele: $f(14) = 2, f(8) = 1, f(5) = 0$. Das Resultat ist eine natürliche Zahl. f ist offensichtlich berechenbar.

Beispiel 6 Zu Frage 2: Teilbarkeit durch 7. Gegeben ist $n \in \mathbb{N}$ in Dezimaldarstellung. Aufgabe: Ist 7 ein Teiler von n $(7 \mid n)$?

Zwei mögliche Ausgaben:

- ja: falls dies wahr ist
- nein: falls dies falsch ist

Das Resultat ist ein Wahrheitswert. Auch hier wird eine Funktion $g(n)$ berechnet.

$$g(n) = \begin{cases} 1 & \text{falls } 7 \mid n \\ 0 & \text{sonst} \end{cases}$$

Berechnungsprobleme versus Entscheidungsprobleme

Beispiel 7 *Zerl*: Gegeben sei $n \in \mathbb{N}$ in Dezimaldarstellung. Aufgabe: Bestimme eine Liste aller Primfaktoren von n.

Beispiel: $n = 360 = 2 \cdot 180 = 2 \cdot 2 \cdot 90 = 2 \cdot 2 \cdot 2 \cdot 45 = 2 \cdot 2 \cdot 2 \cdot 3 \cdot 15 = 2 \cdot 2 \cdot 2 \cdot 3 \cdot 3 \cdot 5$

Es gibt einen Algorithmus, der die Liste der Primfaktoren erzeugt.

Dogma 1 Es gibt keinen Algorithmus, der für beliebig große natürliche Zahlen n die Liste der Primfaktoren von n erzeugt und der effizient ist.

Bemerkung 1 Auf Dogma 1 beruht die Sicherheit des Krypto-Systems RSA.

Beispiel 8 Anderes Problem (*Prim*): Gegeben sei $n \in \mathbb{N}$ in Dezimaldarstellung. Frage: Ist n eine Primzahl?

Es gibt einen Algorithmus, der diese Frage beantwortet.

Test:

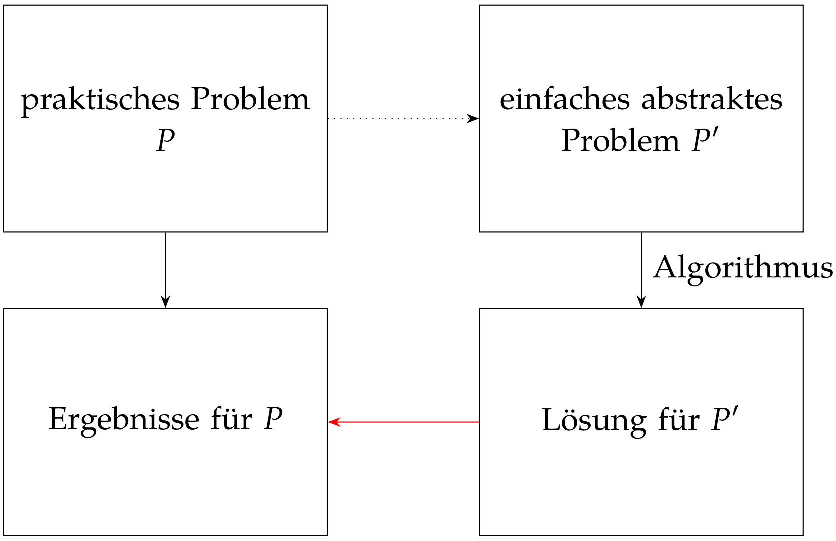
- Falls $n = 0$ oder $n = 1$, gib falsch aus.
- Sonst: Setze einen Zähler $i = 2$. Teste, ob $i \mid n$.
  - Falls ja, gib falsch aus.
  - Sonst, setze $i = i + 1$ solange $i \leq \sqrt{n}$ und teste erneut.
  - Falls $i = \sqrt{n}$, gib wahr aus.

Das Entscheidungsproblem ist lösbar und es gibt sogar einen effizienten Algorithmus für das Problem.

Bemerkung 2 Welcher Zusammenhang besteht zwischen dem Berechnungsproblem *Zerl* und dem Entscheidungsproblem *Prim*?

Eine Lösung von *Zerl* beantwortet *Prim* (*Prim* ist nicht schwerer als *Zerl*). Aber nicht jede Antwort auf *Prim* liefert eine Lösung für *Zerl*, mit anderen Worten: *Zerl* ist mindestens so schwer wie *Prim*.

Was hat Theoretische Informatik mit Wirklichkeit zu tun?



Beispiel 9 Transportproblem TP: Gegeben seien n LKW (Größe, maximale Ladung, Kosten, ...) und m Pakete (Größe, Gewicht, Ausgangspunkt, Zielort, Deadline, ...). Aufgabe: Bestimme einen optimalen Transportplan.

Beobachtung: Die formale Beschreibung von TP ist aufwändig.

Beispiel 10 Ein theoretisches Problem *Partition*: Gegeben sind m natürliche Zahlen $a_1, a_2, \ldots, a_m$. Frage: Gibt es eine Indexmenge $I \subseteq \{1, 2, \ldots, m\}$ mit

$$\sum_{i \in I} a_i = \sum_{i \in \overline{I}} a_i\,?$$

Beobachtung: *Partition* ist einfach zu beschreiben.

Bemerkung 3 Welcher Zusammenhang existiert zwischen beiden Problemen? Eine Lösung von TP hat eine Lösung von *Partition* zur Folge.

Annahme: Wir haben einen Algorithmus für TP. Dann konstruieren wir einen Algorithmus für *Partition*, in den wir den Algorithmus für TP als Unterprogramm verwenden.

Gegeben ist eine Eingabe für *Partition*: $a_1, a_2, \ldots, a_m$. Wir übersetzen diese Eingabe in eine Eingabe für *TP* auf folgende Weise:

- 2 LKW mit maximaler Ladung $\frac{A}{2}$. Dabei ist $A = a_1 + a_2 + \ldots + a_m$.
- m Pakete mit den Gewichten $a_1, a_2, \ldots, a_m$. Alle haben denselben Ausgangs- und auch Zielort.

Frage: Lässt sich dieser Transport mit den 2 LKW ausführen? Eine Lösung für *TP* liefert eine Lösung für *Partition* (*Partition* ist nicht schwerer als *TP*), aber *Partition* ist *NP-vollständig*.

Dogma 2 Für NP-vollständige Probleme gibt es keine effizienten Lösungen.

Bemerkung 4 Konsequenz aus Dogma 2: Für *TP* gibt es erst recht keine effiziente Lösung.

Was bedeutet effizient?

$$\text{Eingabe} \longrightarrow \boxed{\text{Algorithmus/Programm}} \longrightarrow \text{Ausgabe}$$

Datum/
Information:
Wort $w \in \Sigma^*$
(Länge $|w| = n$)

Wie messen wir den Aufwand für einen Algorithmus/Programm? Zentrale Größen sind Laufzeit (Zeit) und Speicherbedarf (Raum).

Wie messen wir die Zeit? Das ist die Anzahl der Schritte des Programms bzw. die Anzahl der Takte des Algorithmus in Abhängigkeit von der Länge der Eingabe.

- *Part* hat einen Aufwand exponentiell in der Eingabelänge.
- *SAT* hat einen Aufwand exponentiell in der Eingabelänge.

Das bedeutet „praktisch nicht machbar“.

Bemerkung 5 *Effizient* ist das, was praktisch machbar ist. Das ist das, was einen zeitlichen Aufwand hat, der polynomiell in der Länge der Eingabe ist. Wir schreiben das polynom(n), z. B. $O(n^2)$ für quadratisch.

Definition 2

$$P = \{L \mid L \text{ ist ein Entscheidungsproblem, für das ein Lösungsalgorithmus existiert, der polynomiellen Aufwand hat.}\}$$

Das bedeutet, es gibt eine natürliche Zahl k und die Zeit ist beschränkt durch $\mathcal{O}(n^k)$.

Bemerkung 6

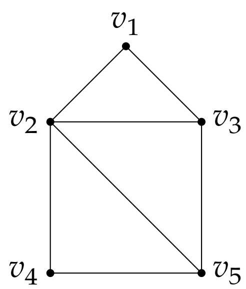
$$NP = \{L \mid L \text{ ist ein Entscheidungsproblem, für das ein Lösungsalgorithmus existiert, der polynomiellen Aufwand hat und für eine beliebige Eingabe-Instanz von } L \text{ entscheidet, ob diese Instanz eine Lösung ist oder nicht.}\}$$

Bemerkung 7 Das P-NP-Problem beinhaltet die Frage, ob $P = NP$ gilt. Wir wissen, dass $P \subseteq NP$.

1.2 Alphabete, Wörter und Sprachen

Die Darstellung von Daten bzw. Informationen – als Eingabe für Algorithmen – passiert durch „Wörter“.

Beispiel 11 Darstellung eines Graphen durch seine Adjazenzmatrix.



	v_1	v_2	v_3	v_4	v_5
v_1	0	1	1	0	0
v_2	1	0	1	1	1
v_3	1	1	0	0	1
v_4	0	1	0	0	1
v_5	0	1	1	1	0

Codewort: #01100#10111#11001#01001#01110# ist ein Wort über dem Alphabet $\Sigma_{\text{graph}} = \{0, 1, \#\}$

Definition 3 Ein *Alphabet* Σ ist eine endliche nicht-leere Menge.

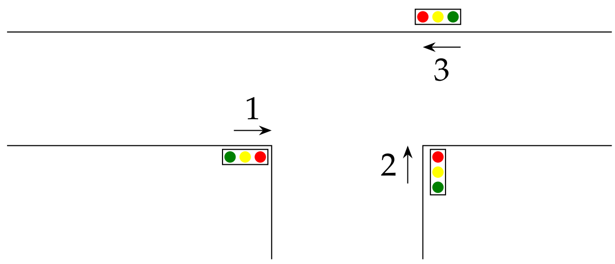
Beispiel 12

- $\{a, b, A, B\}$
- $\{x, y, z, ?, ?\}$
- $\{2, 4, 5, 8, \ldots\}$ – kein Alphabet
- $\Sigma_{\text{griech}} = \{\alpha, \beta, \gamma, \ldots, \omega\}$ (24 Buchstaben)
- $\Sigma_{\text{lat}} = \{a, b, c, \ldots, z\}$ (26 Buchstaben)
- $\Sigma_{\text{Boole}} = \{0, 1\}$
- $\Sigma_{\text{dezi}} = \{0, 1, \ldots, 9\}$ (10 Buchstaben)
- $\emptyset$ – kein Alphabet
- $\Sigma_{\text{unär}} = \{1\}$

Beispiel 13 Darstellung von aussagenlogischen Formeln wie $\neg(A_1 \vee \neg A_2) \implies A_3$

Codierung als $\neg(A_1 \vee \neg A_{10}) \implies A_{11}$. $\Sigma_{\text{logik}} = \{A, 0, 1, \wedge, \vee, \neg, \implies, \iff, (,)\}$

Beispiel 14 Darstellung einer Ampelschaltung



Eingaben: Aus Richtung i

- kommt ein Auto: 1
- kommt kein Auto: 0

eine mögliche Eingabe: $(0, 0, 0)$, alle möglichen Eingaben: $\Sigma_{\text{Ampel}} = \{(0, 0, 0), (0, 0, 1), (0, 1, 0), (1, 0, 0), (0, 1, 1), (1, 0, 1), (1, 1, 0), (1, 1, 1)\}$ (8 Buchstaben)

im Unterschied zu $\Sigma' = \{0, 1, (,)\} \cup \{,\}$ für das „universelle" Alphabet $\Sigma_{\text{Tastatur}} = \{a, b, \ldots, z\} \cup \{A, B, \ldots, Z\} \cup \{\text{ä}, \text{Ä}, \ldots\} \cup \{0, 1, \ldots, 9\} \cup \{\text{␣}\} \cup \ldots$.

Definition 4 Ein *Wort* w über einem Alphabet Σ ist eine endliche Folge von Buchstaben aus Σ. Σ^* ist die Menge aller endlichen Folgen von Buchstaben aus Σ.

Bemerkung 8 $\Sigma = \{a_1, a_2, \ldots, a_m\}$ sind m Buchstaben in Σ. $w = a_{i_1} a_{i_2} \ldots a_{i_n}$ ist eine Folge von n Buchstaben ($|w| = n$).

Sonderfall Länge 0: λ – das leere Wort

Bemerkung 9 Σ^* ist die Menge aller (endlichen) Wörter über Σ. Formaler per Rekursion:

$$\begin{aligned}\Sigma^0 &= \{\lambda\}\\ \Sigma^1 &= \Sigma\\ \Sigma^{n+1} &= \Sigma^n\Sigma = \{wa \mid w \in \Sigma^n, a \in \Sigma\}\\ \Sigma^* &= \bigcup_{n\in\mathbb{N}} \Sigma^n\end{aligned}$$

Σ^n ist die Menge aller Wörter der Länge n über Σ.

Bemerkung 10 Was sind Wörter über Σ_{Tastatur}? Wörter (natürliche Sprache), Zahlen, Sätze Texte, Romane, ...

Der Begriff „Wort" in der Sprache der Informatik ist ein anderer als in unserer Umgangssprache.

Operationen für Wörter

Hintereinanderschreiben: Für zwei Wörter $u = x_1x_2...x_n$ und $v = y_1y_2...y_k$ (mit $x_i, y_i \in \Sigma = \{a_1, ..., a_m\}$) ist $u \circ v = x_1x_2...x_ny_1y_2...y_k$ die *Konkatenation*.

$\circ$ ist eine zweistellige, assoziative Operation, d. h. $(u \circ v) \circ w = u \circ (v \circ w)$ mit $w = z_1z_2...z_l$, und $u \circ \lambda = u = \lambda \circ u$, d. h. λ ist das neutrale Element bezüglich $\circ$. Das bedeutet, die Struktur $(\Sigma^*, \circ, \lambda)$ ist ein Monoid und heißt *freie Worthalbgruppe* über Σ.

Wir wissen, die Struktur $(\mathbb{N}, +, 0)$ ist auch ein Monoid. Die Abbildung $|\,| : \Sigma^* \to \mathbb{N}$, die jedem Wort $w \in \Sigma^*$ seine *Länge* $|w|$ zuordnet, ist ein Homomorphismus von Σ^* auf $\mathbb{N}$, denn es gilt $|u \circ v| = |u| + |v|$.

Spiegelwort: Für ein Wort $u = x_1x_2...x_{n-1}x_n$ mit $x_i \in \Sigma = \{a_1, a_2, ..., a_m\}$. $\text{Sp}(u) = x_nx_{n-1}...x_2x_1 = u^R$. Sp ist eine einstellige Operation, für die gilt:

$$\begin{aligned}\text{Sp}(\lambda) &= \lambda\\ \text{Sp}(a) &= a && (a \in \Sigma)\\ \text{Sp}(wa) &= a\text{Sp}(w) && (a \in \Sigma, w \in \Sigma^n)\end{aligned}$$

Relationen für Wörter

Definition 5 Seien $u, w \in \Sigma^*$. u ist *Teilwort* von w (in Zeichen: $u \sqsubseteq w$) genau dann, wenn es zwei Wörter l und r gibt, sodass $w = lur$.

Beispiel 15 $w = baabaaab \in \{a,b\}^*$. Alle Teilwörter der Länge 3: *baa*, *aab*, *aba*, *baa*, *aaa*, *aab*

Bemerkung 11 Eigenschaften von $\sqsubseteq$:

- reflexiv
- anti-symmetrisch
- transitiv

$\sqsubseteq$ ist eine Halbordnungsrelation. Damit ist die Struktur $(\Sigma^*, \sqsubseteq)$ eine halbgeordnete Menge.

Bemerkung 12 Sei $u \sqsubseteq w$ und $w = lur$. l oder r können das leere Wort sein.

- Falls $l = \lambda$, dann gilt $w = ur$. u ist *Präfix* (Anfangswort) von w.
- Falls $r = \lambda$, dann gilt $w = lu$. u ist *Suffix* (Endwort) von w.
- Falls $l = \lambda$ und $r = \lambda$, dann ist $u = w$.

Potenzen von Wörtern

Definition 6 Sei $w = x_1, x_2, ..., x_n \in \Sigma^*$ ein Wort der Länge n über Σ.

$$\begin{aligned} w^0 &= \lambda \\ w^1 &= w \\ w^{k+1} &= w^k \circ w \end{aligned}$$

Die *k-te Potenz* von w ist das Wort w k-mal hintereinander geschrieben.

Bemerkung 13 Dies erlaubt, Wörter kürzer darzustellen:

$$\begin{aligned} w &= baabaaabab \\ &= (baa)^2(ab)^2 = (ba^2)^2(ab)^2 \\ &= b^1a^2b^1a^2a^1b^1a^1b^1 \end{aligned}$$

Damit haben wir folgende Schreibweise für $\Sigma = \{a\}$: $\Sigma^* = \{\lambda, a, a^2, a^3, ..., a^n, ...\}$.

Binäre Nummern und Binärdarstellung

Definition 7 $\Sigma_{\text{binär}} = \{0,1\}$. $w \in \Sigma^*_{\text{binär}}$ sei ein beliebiges Wort mit der Form $w = x_1x_2...x_n$. Die *binäre Nummer* ist definiert als

$$\mathrm{Nr}_{\mathrm{bin}}(w) = \sum_{i=1}^{n} x_i \cdot 2^{n-i}.$$

Definition 8 Die *Binärdarstellung* von n ist das kürzeste Wort w aus $\{0,1\}^*$ mit $\mathrm{Nr}_{\mathrm{bin}}(w) = n$. Binärdarstellungen sind solche Wörter aus $\{0,1\}^*$, die mit 1 beginnen.

Beispiel 16 $w = 001110$. In unserem Beispiel mit $n = 6$ ist $\mathrm{Nr}_{\mathrm{bin}}(001110) = 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 14$. Zum Beispiel $\mathrm{bin}(14) = (1110)_2$.

Bemerkung 14 Es gibt genauso viele Wörter, die mit 1 beginen wie solche, die mit 0 beginnen. Für jede natürliche Zahl n gibt es unendlich viele Wörter w mit $\mathrm{Nr}_{\mathrm{bin}}(w) = n$ (das ist das Gegenteil von eindeutiger Zuordnung).

Zum Beispiel gilt für alle $w \in \{0, 00, 000, ..., 0^k, ...\}$ $\mathrm{Nr}_{\mathrm{bin}}(w) = 0$. Wünschenswert ist die Identifizierung zwischen Wörtern und natürlichen Zahlen.

Kanonische Ordnung und dyadische Darstellung

Beispiel 17 Es sei $\Sigma = \{a_1, a_2, ..., a_m\}$ ein beliebiges Alphabet mit der natürlichen Ordnung $a_1 < a_2 < ... < a_m$. Warum lässt sich die übliche alphabetische Ordnung für Wörter aus Σ^* nicht nutzen?

Zählt man die Wörter einfach auf, erhält man $a_1 < a_1a_1 < a_1a_1a_1 < ...$. Auf diese Weise würden nur Wörter aus a_1 gebildet und a_2 und andere Buchstaben nicht erreicht.

Definition 9 Seien $u, v \in \Sigma^*$. Die *kanonische Ordnung* (quasilexikografische Ordnung) ist definiert durch $u < v$ genau dann, wenn

1. $|u| < |v|$ oder
2. $|u| = |v|$ und es gibt einen Präfix p, sodass gilt $u = pa_iu_R$ und $v = pa_jv_R$ und $a_i < a_j$.

Bemerkung 15 Zur Schreibweise der Konkatenation:

$$u \circ v = x_1...x_my_1...y_k = uv$$

Beispiel 18 Nummerierungen von Wörtern über einem Alphabet:

$$
\begin{array}{lllcccccl}
\Sigma^*_{\text{binär}} & = \{0,1\}^* & = \{ & \lambda, & 0, & 1, & 00, & 01, & \\
\multicolumn{2}{c}{\text{Platznummern}} & & 0 & 1 & 2 & 3 & 4 & \\
 & & & 10, & 11, & 000, & 001, & \dots & \} \\
 & & & 5 & 6 & 7 & 8 & \dots & \\
\Sigma^* & = \{a,b\}^* & = \{ & \lambda, & a, & b, & aa, & ab, & \\
 & a < b & & ba, & bb, & aaa, & aab, & \dots & \} \\
\Sigma^*_{\text{dya}} & = \{1,2\}^* & = \{ & \lambda, & 1, & 2, & 11, & 12, & \\
 & 1 < 2 & & & 1 \cdot 2^0 & 2 \cdot 2^0 & 1 \cdot 2^1 & 1 \cdot 2^2 & \\
 & & & & & & +1 \cdot 2^0 & +2 \cdot 2^0 & \\
 & & & 21, & 22, & 111, & 112, & \dots & \} \\
 & & & 2 \cdot 2^1 & 2 \cdot 2^1 & 1 \cdot 2^2 & 1 \cdot 2^2 & & \\
 & & & +1 \cdot 2^0 & +2 \cdot 2^0 & +1 \cdot 2^1 & +1 \cdot 2^1 & & \\
 & & & & & +1 \cdot 2^0 & +2 \cdot 2^0 & &
\end{array}
$$

Bemerkung 16 Fakt: Jede positive natürliche Zahl k besitzt eine Darstellung der Form

$$
\begin{aligned}
k &= y_1 \cdot 2^{n-1} + y_2 \cdot 2^{n-2} + \dots + y_{n-1} \cdot 2^1 + y_n \cdot 2^0 \\
&= \sum_{i=1}^{n} y_i \cdot 2^{n-i}
\end{aligned}
$$

mit $y_1, \dots, y_n \in \{1,2\}$. Dann heißt $(y_1 y_2 \dots y_{n-1} y_n)_{\text{dya}}$ *dyadische Darstellung* von k. Die dyadische Darstellung von 0 ist das leere Wort.

Damit haben wir eine Zuordnung Platznummer : $\{1,2\}^* \twoheadrightarrow \mathbb{N}$, die eineindeutig ist. Wir haben eine zweite Zuordnung dya : $\mathbb{N} \twoheadrightarrow \{1,2\}^*$, die eineindeutig ist und die Umkehrabbildung von Platznummer. Wir haben damit eine Bijektion (Identifizierung) zwischen Wörtern und natürlichen Zahlen. Deshalb benutzen wir künftig bevorzugt die dyadische Darstellung.

Beispiel 19 Es ist für positive natürliche Zahlen $k = (y_1 y_2 \dots y_n)_{\text{dya}}$ und $n \geq 1$ zu zeigen

$$
k = \sum_{i=1}^{n} y_i \cdot 2^{n-i}. \tag{1}
$$

Der Beweis erfolgt per vollständiger Induktion über k.

- Induktionsanfang: trivial
- Induktionsvoraussetzung: Gleichung (1) gilt für k.
- Induktionsbehauptung: Gleichung (1) gilt für $k+1$

Beweis per Fallunterscheidung:

1. Fall: Alle Ziffern von k sind $y_1 = y_2 = ... = 2$. Dann gilt $k + 1 = (\underbrace{111...1}_{n+1})_{\text{dya}} = \sum_{i=1}^{k+1} 1 \cdot 2^{n+1-i}$.
2. Fall: Es gibt eine 1 in der dyadischen Darstellung von k. $y_l = 1$, wobei l der größte Index mit dieser Eigenschaft ist. Also ist $k = (y_1 y_2 ... y_{l-1} y_l y_{l+1} ... y_n)$, also $k = (y_1 ... y_{l-1} 12...2)$. Dann ist $(k+1)_{\text{dya}} = (y_1 y_2 ... y_{l-1} 21...1)$.

Die Fallunterscheidung ist vollständig. □

Bemerkung 17 $\Sigma = \{1, 2, 3\}$. Fakt: Es gibt eine eindeutig bestimmte Darstellung für $k > 0$: $k = \sum_{i=1}^{n} z_i \cdot 3^{n-i}$ mit $z_1, ..., z_n \in \{1, 2, 3\}$. $(z_1 z_2 ... z_n)_{\text{tria}}$ heißt *triadische Darstellung*.

$\Sigma = \{1, 2, ..., m\}$. Fakt: Es gibt eine eindeutig bestimmte Darstellung für $k > 0$: $\sum_{i=1}^{n} \alpha_i \cdot m^{n-i}$ mit $\alpha_1, ..., \alpha_n \in \{1, ..., m\}$. $(\alpha_1 \alpha_2 ... \alpha_n)_{\text{m-ad.}}$ heißt *m-adische Darstellung*.

Formale Sprachen

Definition 10 Eine Menge L ist eine *formale Sprache* L über Σ genau dann, wenn $L \subseteq \Sigma^*$. (Jede Ansammlung von Wörtern über Σ ist eine formale Sprache.)

Beispiel 20 $\Sigma = \{0, 1\}$. $L_{\text{binär}} = \{w \in \{0, 1\}^* \mid \exists u \in \{0, 1\}^* : w = 1u\} \subsetneq \{0, 1\}^*$

Beispiel 21 $\Sigma = \{1, 2\}$. $L_{\text{dya}} = \Sigma^*$.

Bemerkung 18

- Formale Sprachen sind Mengen. Deshalb sind alle Mengenoperationen für formale Sprachen definiert: $L_1 \cup L_2, L_1 \cap L_2, L_1 \setminus L_2, \overline{L} = \Sigma^* \setminus L, L_1 \times L_2, ...$
- Formale Sprachen sind Mengen von Wörtern. Deshalb sind die Wortoperationen definiert:
  - $L_1 \circ L_2 = \{w \in \Sigma^* \mid \exists u \in \Sigma^* : \exists v \in \Sigma^* : w = u \circ v \land u \in L_1 \land v \in L_2\}$,
  - $\text{Sp}(L) = \{w \in \Sigma^* \mid \exists u \in \Sigma^* : w = \text{Sp}(u) \land w \in L\} = \{w \in \Sigma^* \mid \text{Sp}(w) \in L\}$
  - Potenzen von Sprachen: $L^0 = \{\lambda\}$, $L^1 = L$, $L^{n+1} = L^n \circ L$, $L^* = \bigcup_{n \in \mathbb{N}} L^n$

Beispiel 22 Es ist zu zeigen, dass $L_1 \circ (L_2 \cup L_3) = L_1 \circ L_2 \cup L_1 \circ L_3$.

$$\begin{aligned}
w \in L_1 \circ (L_2 \cup L_3) &\iff \exists u : \exists v : u \in L_1 \land v \in L_2 \cup L_3 \land uv = w \\
&\iff \exists u : \exists v : u \in L_1 \land \\
&\qquad\qquad (v \in L_2 \lor v \in L_3) \land uv = w \\
&\iff \exists u : \exists v : (u \in L_1 \land v \in L_2 \lor \\
&\qquad\qquad u \in L_1 \land v \in L_3) \land uv = w \\
&\iff \exists u : \exists v : (uv \in L_1 \circ L_2 \lor \\
&\qquad\qquad uv \in L_1 \circ L_3) \land uv = w \\
&\iff w \in L_1 \circ L_2 \lor w \in L_1 \circ L_3 \\
&\iff w \in L_1 \circ L_2 \cup L_1 \circ L_3
\end{aligned}$$

□

2 Reguläre Sprachen

2.1 Deterministische Endliche Automaten

Definition 11 Ein *endlicher Automat* A ist ein 5-Tupel $A = (Q, \Sigma, \delta, q_o, F)$, wobei gilt

Q ist eine endliche Menge (Zustandsmenge),
Σ ist eine endliche Menge (Eingabealphabet),
δ ist eine Abbildung $\delta : Q \times \Sigma \mapsto Q$ (Überführungsfunktion),
$q_0 \in Q$ ist der Startzustand und
$F \subseteq Q$ ist die Menge der Finalzustände.

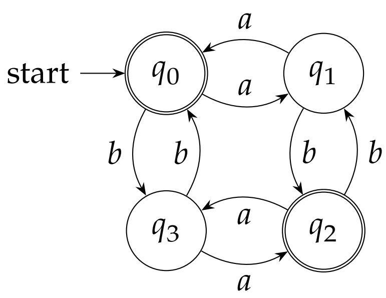
Bemerkung 19 Die Überführungsfunktion δ ist total definiert.

Beispiel 23 $A = (Q, \Sigma, \delta, q_0, F)$ mit

- $Q = \{q_0, q_1, q_2, q_3\}$,
- $\Sigma = \{a, b\}$,
- $q_0 = q_0$ und
- $F = \{q_0, q_2\}$.

δ	a	b
q_0	q_1	q_3
q_1	q_0	q_2
q_2	q_3	q_1
q_3	q_2	q_0

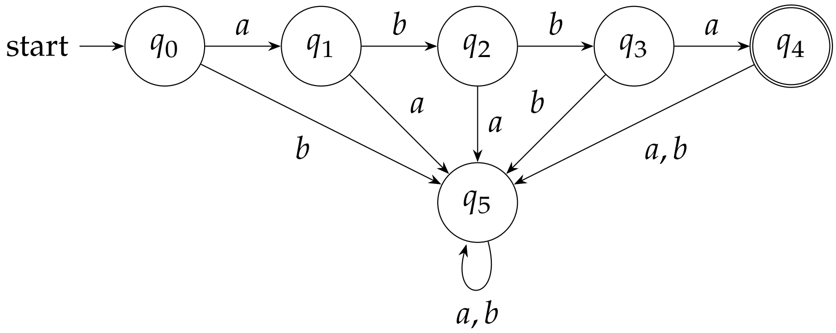
Darstellung als Diagramm: Zustände sind Knoten. Überführung wird durch Pfeile dargestellt.



Beispieleingaben:

- *aaba*: Wanderung durch den Automaten, die in q_2 endet. Da $q_2 \in F$, wird diese Eingabe akzeptiert.
- *aa*: Wanderung nach q_0, wird akzeptiert
- *abba*: wird akzeptiert
- *aba*: wird nicht akzeptiert

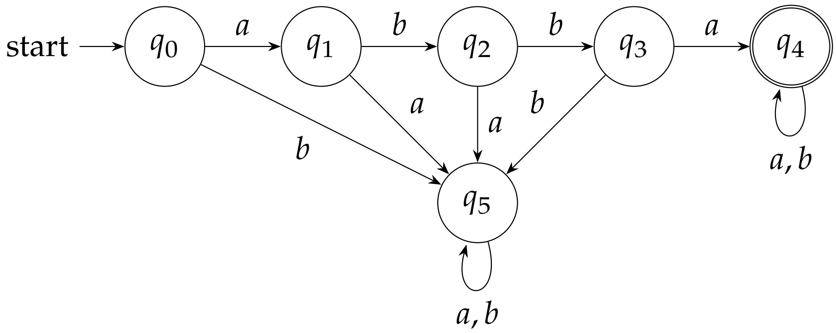
Beispiel 24 Wir konstruieren einen endlichen Automaten, der die Sprache $\{abba\}$ akzeptiert[1].



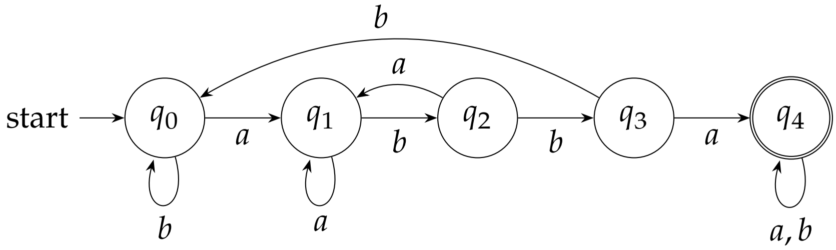
δ ist total definiert.

Beispiel 25 Wir konstruieren einen endlichen Automaten, der die Sprache $\{abbaw \mid w \in \Sigma^*\}$ mit $\Sigma = \{a, b\}$ akzeptiert. Der Automat akzeptiert alle Wörter aus Σ^* mit dem Präfix *abba*.

[1] Der Automat akzeptiert ein einziges Wort.

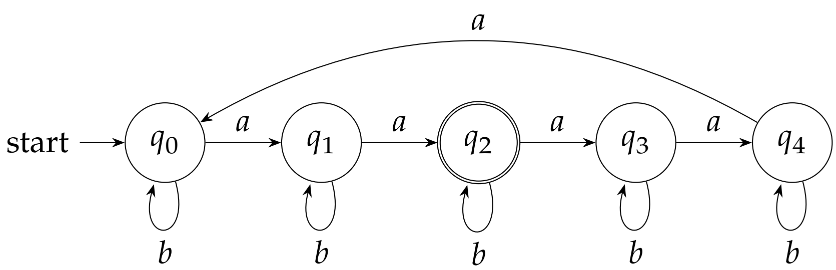


Beispiel 26 Wir konstruieren einen endlichen Automaten, der die Sprache $\{uabbav \mid u, v \in \{a, b\}^*\}$ akzeptiert. Der Automat akzeptiert alle Wörter aus Σ^* mit dem Teilwort *abba*.



Bemerkung 20 Endliche Automaten können Muster erkennen, z. B. das Muster „*abba*“. Automaten können auch kongruent modulo m zählen.

Beispiel 27 Wir konstruieren einen endlichen Automaten, der die Sprache $\{w \in \{a, b\}^* \mid |w|_a \equiv_5 2\}$ akzeptiert.



Konfigurationen eines Automaten

Wieso können Automaten rechnen? Formale Definition der Berechnung eines Automaten A bei Eingabe w.

Definition 12 Gegeben sei ein endlicher Automat $A = (Q, \Sigma, \delta, q_0, F)$ und außerdem eine Eingabe $w \in \Sigma^*$. Eine *Konfiguration* K von A bei Eingabe w ist ein Paar $K = (p, u)$, wobei $p \in Q$ (aktueller Zustand von A) und $u \in \Sigma^*$[2]

[2] u ist der aktuelle noch zu lesende „Rest“ der Eingabe w. u ist der Suffix von w.

und ist damit eine vollständige Beschreibung der Momentansituation von A mit w.

Definition 13 Gegeben sei ein endlicher Automat $A = (Q, \Sigma, \delta, q_0, F)$ und der Eingabe $w \in \Sigma^*$. Wir definieren eine binäre Relation $\vdash$ über der Menge der Konfigurationen, d. h. $\vdash \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$, auf folgende Weise: Es sei $K = (p, u)$ und $K' = (q, v)$. Falls $\exists x \in \Sigma : u = xv$ und $\delta(p, x) = q$, dann ist $K \vdash K'$ die *unmittelbare Nachfolgekonfiguration* von K.

Bemerkung 21 Dieser Übergang $K \vdash K'$ entspricht einem Berechnungsschritt (einem Takt) im Automaten A bei Eingabe w.

Besondere Konfigurationen:

- *Startkonfiguration* von A bei Eingabe w: Start-$K_A(w) = (q_0, w)$
- *akzeptierende Finalkonfiguration*: $K = (q, \lambda)$ mit $q \in F$
- *ablehnende Finalkonfiguration*: $K' = (p, \lambda)$ mit $p \in Q \setminus F$

Definition 14 Gegeben sei ein endlicher Automat $A = (Q, \Sigma, \delta, q_0, F)$ und eine Eingabe $w \in \Sigma^*$. Ein *Takt* der Berechnung von A bei w ist ein Element der *Übergangsrelation* $\vdash \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma)^*$ der Form $(p, u) \vdash (q, v)$, wobei $u = xv$ für ein passendes $x \in \Sigma$ und $\delta(p, u) = q$. (q, v) ist *unmittelbare Nachfolgekonfiguration* von (p, u).

Bemerkung 22 Wir betrachten $\vdash^*$, die reflexive und transitive Hülle von $\vdash$.

Erinnerung: Es sei $R \subseteq M \times M$ eine binäre Relation. $\mathrm{Ref}(R)$ ist die kleinste Relation, die R umfasst und die reflexiv ist, d. h.

$$\mathrm{Ref}(R) = \bigcap_{\substack{R \subseteq R' \\ R' \text{ ist reflexiv}}} R' = R \cup \mathrm{Id}_M.$$

Die Transitive Hülle $\mathrm{Trans}(R)$ ist die kleinste Relation, die R umfasst und transitiv ist, d. h.

$$\mathrm{Trans}(R) = \bigcap_{\substack{R \subseteq R'' \\ R'' \text{ ist transitiv}}} R'' = R \cup R \circ R \cup R \circ R \circ R \cup \ldots = \bigcup_n R^n.$$

Die symmetrische Hülle $\mathrm{Sym}(R)$ ist die kleinste Relation, die R umfasst und symmetrisch ist, d. h.

$$\mathrm{Sym}(R) = \bigcap_{\substack{R \subseteq R''' \\ R''' \text{ ist symmetrisch}}} R''' = R \cup R^{-1}.$$

Bemerkung 23 Frage: Wann gilt $K \vdash^* K'$? In diesem Fall heißt K' *Nachfolgekonfiguration* von K.

$K \vdash^* K'$ genau dann, wenn es eine Folge $K_0, K_1, K_2, ..., K_t$ mit den folgenden Eigenschaften gibt:

- $K = K_0$
- $K_{i-1} \vdash K_i$ für $i = 1, ..., t$
- $K_t = K'$.

Bemerkung 24 Die neue Frage: Wann gilt Start-$K_A \vdash^* (p, \lambda)$? Es sei $w = w_1 w_2 ... w_n$ ein Wort der Länge n (w_i ist der i-te Buchstabe von w).

Start-$K_A \vdash^* (p, \lambda)$ genau dann, wenn es eine Folge $K_0, K_1, ..., K_n$ mit den folgenden Eigenschaften gibt:

- Start-$K_A(w_1 w_2 ... w_n) = K_0$
- $K_{i-1} \vdash K_i$ für $i = 1, ..., n$
- $K_n = (p, \lambda)$.

Bemerkung 25 Im Fall Start-$K_A \vdash^* (p, \lambda)$ ist $B_A(w)$ die *Berechnung* von A bei Eingabe $w = w_1 ... w_n$ als genau diese Folge $(K_0, K_1, ..., K_n)$ definiert.

- Die Berechnung ist *akzeptierend* genau dann, wenn $p \in F$.
- Die Berechnung ist *ablehnend* genau dann, wenn $p \in Q \setminus F$.

Für jedes $w \in \Sigma^*$ ist $B_A(w)$ determiniert und terminierend.

Akzeptierte Sprachen

Bemerkung 26 Sei A ein Automat. Die Sprache, die von A akzeptiert wird, ist

$$\begin{aligned} L(A) &= \{w \in \Sigma^* \mid B_A(w) \text{ ist akzeptierend}\} \\ &= \{w \in \Sigma^* \mid (q_0, w) \vdash^* (p, \lambda) \wedge p \in F\} \\ &= \{w \in \Sigma^* \mid \delta^*(q_0, w_1 w_2 ... w_n) \in F\}. \end{aligned}$$

Dabei ist δ^* die *erweiterte Überführungsfunktion*, die induktiv (Wortinduktion) wie folgt definiert ist (für $q \in Q, u \in \Sigma^*, x \in \Sigma$):

$$\begin{aligned} \delta^*(q, \lambda) &= q \\ \delta^*(q, ux) &= \delta(\delta^*(q, u), x) \end{aligned}$$

Effekt:

$$
\begin{array}{ll}
1. & \delta^*(q,x) = \delta^*(q,\lambda x) = \delta(\delta^*(q,\lambda),x) = \delta(q,x) \\
2. & \delta^*(q,yx) = \delta(\delta^*(q,y),x) = \delta(\delta(q,y),x) \\
3. & \delta^*(q,zyx) = \delta(\delta^*(q,zy),x) = \delta(\delta(\delta(q,z),y),x) \\
\vdots & \qquad\qquad\qquad\qquad \vdots \\
n. & \delta^*(q,x_1x_2...x_n) = \underbrace{\delta(\delta(...}_{n\text{-mal}}\delta(q,x_1),x_2),...,x_n)
\end{array}
$$

Damit gilt für die erweiterte Überführungsfunktion $\delta^* : Q \times \Sigma^* \mapsto Q$ (n-fache Anwendung der Überführungsfunktion für Eingaben der Länge n).

Das funktioniert ebenso für folgende alternative Definition:

$$
\begin{aligned}
\delta^*(q,\lambda) &= q \\
\delta^*(q,xu) &= \delta^*(\delta(q,x),u)
\end{aligned}
$$

Bemerkung 27 $\mathcal{REG}$ bezeichnet die Klasse aller Sprachen, die von endlichen Automaten akzeptiert werden.

$$\mathcal{REG} = \{L \mid \exists \text{ endlicher Automat } A : L = L(A)\}$$

$\mathcal{REG}$ heißt die Klasse der *regulären Sprachen*.

Definition 15 Eine Sprache $L \subseteq \Sigma^*$ ist *regulär* genau dann, wenn es einen endlichen Automaten A gibt mit $L(A) = L$.

Bemerkung 28 Bezeichnungen für endliche Automaten:

- EA (endlicher Automat)
- FA (finite automaton)
- DEA (deterministischer endlicher Automat)
- DFA (deterministic finite automaton)

Zustandsklassen von endlichen Automaten

Bemerkung 29 Es sei $A = (Q, \Sigma, \delta, q_0, F)$ ein endlicher Automat. Wir definieren eine *Zustandsfunktion* $f_A : \Sigma^* \mapsto Q$ auf folgende Weise:

$$
\begin{aligned}
f_A(w) = q &\iff (q_0, w) \vdash^* (q, \lambda) \\
&\iff \delta^*(q_0, w) = q
\end{aligned}
$$

f_A bewirkt eine Zerlegung von Σ^*. Dazu definieren wir die *Zustandsklassen*

$$[q] = \{w \in \Sigma^* \mid f_A(w) = q\}$$

Es gilt

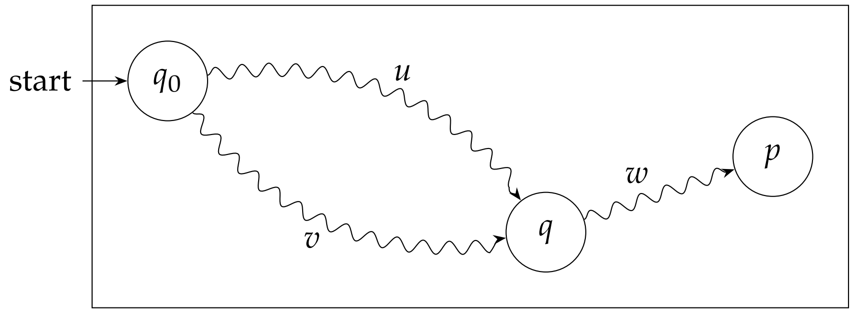
- Für $p \neq q$ ist $[p] \cap [q] = \emptyset$[3].
- $\bigcup_{q \in Q} [q] = \Sigma^*$.
- $\bigcup_{q \in F} [q] = L(A)$.

Also ist die Aufteilung in Zustandsklassen tatsächlich eine Zerlegung.

Definition 16 Die *A-Äquivalenz* ist für alle $u, v \in \Sigma^*$ wie folgt definiert:

$$u \sim_A v \iff \exists q \in Q : u \in [q] \land v \in [q]$$

Was bedeutet $u \sim_A v$?



Alle Informationen über u bzw. v „stecken" in q. Für jedes beliebige Wort $w \in \Sigma^*$ kann A nicht unterscheiden zwischen uw und vw.

Lemma 1 Es sei $A = (Q, \Sigma, \delta, q_0, F)$ ein endlicher Automat. Dann gilt für die A-Äquivalenz

$$u \sim_A v \implies \forall w \in \Sigma^* : uw \sim_A vw.$$

Insbesondere gilt

$$u \sim_A v \implies \forall w \in \Sigma^* : uw \in L(A) \iff vw \in L(A)$$

Beispiel 28 Wir betrachten in Anlehnung an Beispiel 24

- A_1 mit $L(A_1) = \{abba\}$ mit $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$. $[q_0] = \{\lambda\}$, $[q_1] = \{a\}, [q_2] = \{ab\}, [q_3] = \{abb\}, [q_4] = \{abba\}, [q_5] = \Sigma^* \setminus \{\lambda, a, ab, abb, abba\}$
- Wir vergleichen zwei beliebige Wörter aus $[q_0]$ bis $[q_4]$, z. B. $u = a$ und $v = ab$. Dann gilt für $w = bba$: $uw = abba \in L_1$ und $vw = abbba \notin L_1$.
- Das heißt, jeder endliche Automat, der L_1 akzeptiert, muss zwischen a und ab unterscheiden können.

[3] Eine Eingabe, die in p endet, kann nicht in q enden.

- Das heißt, jeder endliche Automat, der L_1 akzeptiert, muss zwischen den Wörtern $\lambda, a, ab, abb, abba$ und b unterscheiden können.
- Das heißt, diese Wörter müssen in verschiedenen Zustandsklassen liegen.
- Das heißt, jeder endliche Automat, der L_1 akzeptiert, muss mindestens 6 Zustände haben.
- Das heißt, A_1 ist minimal.

Beispiel 29 Wir betrachten in Anlehnung an Beispiel 26

- A_3 mit $L(A_3) = \Sigma^* \circ \{abba\} \circ \Sigma^*$ und $Q = \{p_0, p_1, p_2, p_3, p_4\}$
- $[p_0] = \{\lambda, b, bb, bbb, \ldots\}$ (nichts mit *abba* gemein)
- $[p_1] = \{a, aa, ba, \ldots\}$ (Suffix dieser Wörter ist erster Buchstabe in *abba*)
- $[p_2] = \{ab, aab, bab, \ldots\}$ (Suffix dieser Wörter ist zweibuchstabiger Präfix von *abba*)

Definition 17 Wir definieren, unabhängig von endlichen Automaten, die *L-Äquivalenz*. Es sei $L \subseteq \Sigma^*$ eine formale Sprache. Für je zwei Wörter $u, v \in \Sigma^*$ gilt

$$u \sim_L v \iff \forall w \in \Sigma^* : uw \in L \Leftrightarrow vw \in L$$

Bemerkung 30 Zu zeigen ist $\sim_L$ ist eine Äquivalenzrelation.

- reflexiv:

$$\forall u \in L : u \sim_L u \equiv \forall u \in L : \forall w \in \Sigma : uw \in L \iff uw \in L$$

- symmetrisch:

$$\begin{aligned}
\forall u, v \in L : u \sim_L v \implies v \sim_L u \\
&\equiv \forall u, v \in L : (\forall w \in \Sigma^* : uw \in L \iff vw \in L) \\
&\quad \implies (\forall w \in \Sigma^* : vw \in L \iff uw \in L) \\
&\equiv \forall u, v \in L : (\forall w \in \Sigma^* : uw \in L \iff vw \in L) \\
&\quad \implies (\forall w \in \Sigma^* : uw \in L \iff vw \in L)
\end{aligned}$$

  Prämisse und Konklusion stimmen überein.
- transitiv: $\forall u, v, r \in L : u \sim_L v \land v \sim_L r \implies u \sim_L r \equiv \forall u, v, r \in L : (\forall w \in \Sigma^* : uw \in L \iff vw \in L) \land (\forall w \in \Sigma^* : vw \in L \iff rw \in L) \implies \forall w \in \Sigma^* : uw \in L \iff rw \in L$. Da $\forall w \in \Sigma^* : uw \in L \iff vw \in L$ gilt, lässt sich die zweite Bedingung

als $\forall w \in \Sigma^* : uw \in L \iff rw \in L$ schreiben. Damit stimmen Prämisse und Konklusion überein. $\square$

2.2 Mengenoperationen für reguläre Sprachen

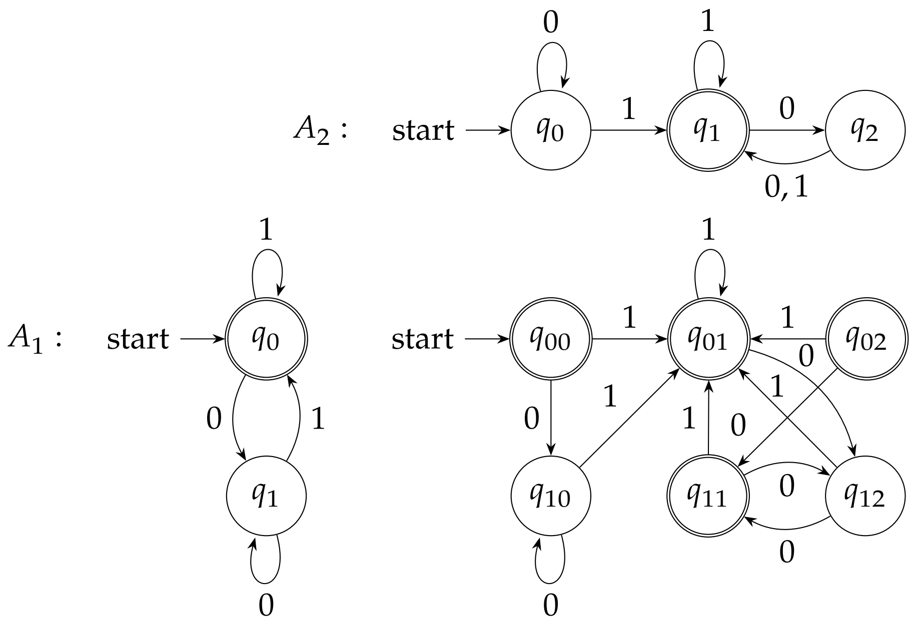
Bemerkung 31 $\mathcal{REG} = \{L \mid \exists A \in \mathrm{EA} : L(A) = L\}$. Klar ist: Wenn $L_1, L_2 \subseteq \Sigma^*$ formale Sprachen sind, dann sind auch $\overline{L_1}, L_1 \cup L_2, L_1 \cap L_2, \ldots$ formale Sprachen. Das heißt, die formalen Sprachen sind abgeschlossen bezüglich Komplement, Vereinigung und Durchschnitt[4].

Bemerkung 32 Die Frage ist: Wenn $L_1, L_2 \subseteq \Sigma^*$ reguläre Sprachen sind, sind dann auch $\overline{L_1}, L_1 \cup L_2, L_1 \cap L_2, \ldots$ reguläre Sprachen?

Einfach ist die Antwort für das Komplement: Es sei L eine reguläre Sprache. Dann existiert ein endlicher Automat $A = (Q, \Sigma, \delta, q_0, F)$ mit $L(A) = L$. Dann akzeptiert der endliche Automat $A^{\mathrm{Co}} = (Q, \Sigma, \delta, q_0, Q \setminus F)$ gerade die Sprache $\overline{L} = \Sigma^* \setminus L = L(A^{\mathrm{Co}})$.

Beispiel 30 Der Produktautomat zweier Automaten ist in Abbildung 2 dargestellt.

Abbildung 2 Produktautomat der Vereinigung zweier Automaten (für den Durchschnitt ist nur q_{01} Finalzustand)



Produktautomat

Wir betrachten mögliche Eingaben

[4] Das heißt, sie sind abgeschlossen bezüglich der Booleschen Operationen.

- $w_1 = 0011$: $w_1 \in L(A_1), w_1 \in L(A_2)$
- $w_2 = 010$: $w_2 \notin L(A_1), w_2 \notin L(A_2)$
- $w_3 = \lambda$: $w_3 \in L(A_1), w_3 \notin L(A_2)$
- $w_4 =?$: $w_4 \notin L(A_1), w_4 \in L(A_2)$

Definition 18 Seien $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ und $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ zwei endliche Automaten. Der *Produktautomat* $(A_1 \times A_2)^V$ bzw. $(A_1 \times A_2)^D$ ist wie folgt definiert:

- Zustandsmenge: $Q = Q_1 \times Q_2$
- Alphabet: Σ
- Überführungsfunktion: $\delta : Q \times \Sigma \mapsto Q$ mit $\delta((q, p), x) = (\delta_1(q, x), \delta_2(p, x))$
- Startzustand: $q_0 = (q_{01}, q_{02})$
- Finalzustände (Vereinigung): $F = F_1 \times Q_2 \cup Q_1 \times F_2$
- Finalzustände (Durchschnitt): $F = F_1 \times F_2$.

Satz 1 Wenn L_1 eine reguläre Sprache ist und L_2 eine reguläre Sprache ist, dann ist $L_1 \cup L_2$ auch eine reguläre Sprache.

Beweis in zwei Teilen:

1. Teil: Konstruktion des Produktautomaten. Das ist getan: $A = (A_1 \times A_2)^V = (Q_1 \times Q_2, \Sigma, \delta, q_0, F_1 \times Q_2 \cup Q_1 \times F_2)$.
2. Teil: Korrektheit des Produktautomaten. Dieser Nachweis ist noch nicht erbracht: $w \in L(A) \iff w \in L_1 \cup L_2$. Nachweis der Korrektheit:

$$
\begin{aligned}
w \in L(A) &\iff \delta^*(q_0, w) \in F \\
&\iff \delta^*((q_{01}, q_{02}), w) \in F_1 \times Q_2 \cup Q_1 \times F_2 \\
&\iff (\delta_1^*(q_{01}, w), \delta_2^*(q_{02}, w)) \in F_1 \times Q_2 \cup Q_1 \times F_2 \qquad (2) \\
&\iff (\delta_1^*(q_{01}, w), \delta_2^*(q_{02}, w)) \in F_1 \times Q_2 \\
&\qquad \text{oder } (\delta_1^*(q_{01}, w), \delta_2^*(q_{02}, w)) \in Q_1 \times F_2 \\
&\iff \delta_1^*(q_{01}, w) \in F_1 \text{ oder } \delta_2^*(q_{02}, w) \in F_2 \\
&\iff w \in L(A_1) = L_1 \text{ oder } w \in L(A_2) = L_2 \\
&\iff w \in L_1 \cup L_2
\end{aligned}
$$

Wir wissen: $\delta((q_1, q_2), x) = (\delta_1(q_2, x), \delta_2(q_2, x))$. Es ist noch zu zeigen, dass $\delta^*((q_1, q_2), w) = (\delta_1^*(q_1, w), \delta_2^*(q_2, w))$ (verwendet in Gleichung (2)). Das lässt sich per Wortinduktion zeigen.

Bemerkung 33 *Wortinduktion*: Es sei Σ ein Alphabet und E irgendeine Eigenschaft der Wörter aus Σ^*[5].

Das Prinzip der Wortinduktion:

- Induktionsanfang: $E(\lambda)$
- Induktionsschritt: $\forall x \in \Sigma : E(w) \implies E(wx)$

Dann gilt $\forall w \in \Sigma^* : E(w)$.

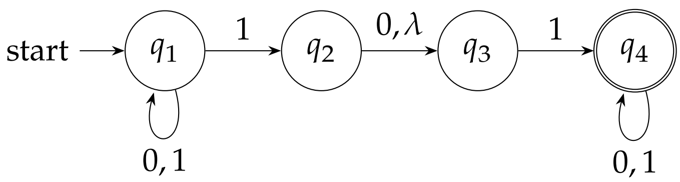
Satz 2 Die Klasse $\mathcal{REG}$ der regulären Sprachen ist abgeschlossen bezüglich der Booleschen Operationen.

Bemerkung 34 Ist die Klasse $\mathcal{REG}$ auch bezüglich der Sprachoperationen, insbesondere bzgl. der Konkatenation, abgeschlossen?

Es sei $L_1 = L(A_1)$ und $L_2 = L(A_2)$. Ist $L_1 \circ L_2 = \{w \in \Sigma^* \mid \exists u \in L_1 : \exists v \in L_2 : w = uv\} \in \mathcal{REG}$? Der Produktautomat ist nicht geeignet, $L_1 L_2$ zu akzeptieren.

2.3 Nichtdeterministische endliche Automaten

Beispiel 31 Wir betrachten den folgenden Automaten:



Besonderheiten:

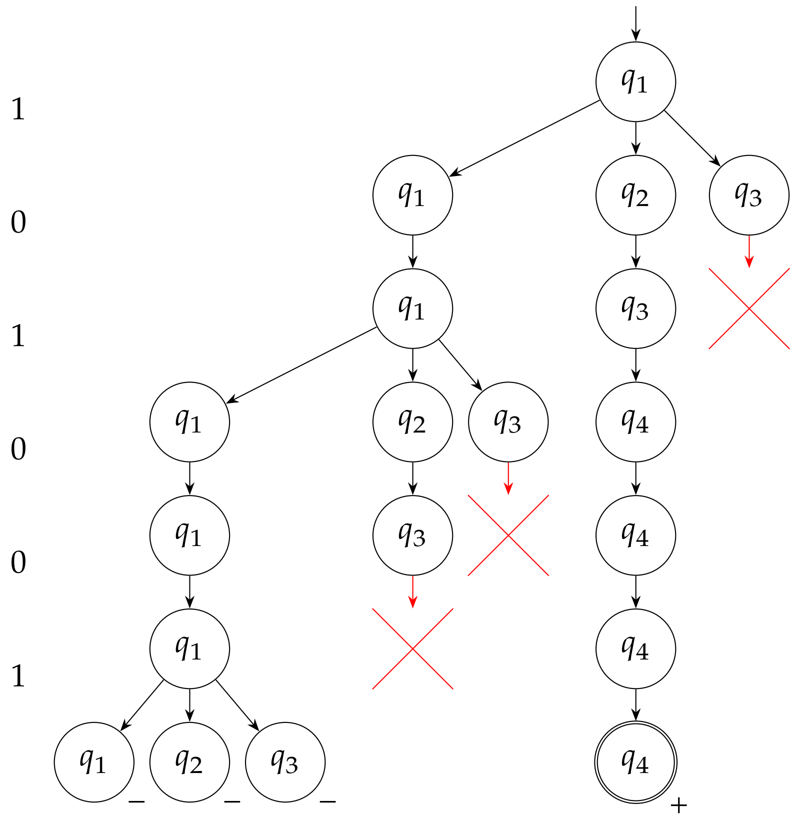
- q_1 mit q bleibe in q_1 oder gehe zu q_2
- q_2 hat keinen Übergang für 1, aber hat einen λ-Übergang
- q_3 hat keinen Übergang für 0

Eine beispielhafte Berechnung ist in Abbildung 3 abgebildet.

δ	**0**	**1**	$\boldsymbol{\lambda}$
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

[5] Beispiel: für $w \in \Sigma^*$ gilt $E(w) \iff w \in L(A) \iff w \in L_1 \cup L_2$.

Abbildung 3 Berechnung des Automaten aus Beispiel 31 bei Eingabe 101001



Definition 19 Ein *nichtdeterministischer endlicher Automat* (NEA, NFA) N ist ein 5-Tupel der Form $N = (Q, \Sigma, \delta, q_0, F)$, wobei

Q endliche Menge (Zustandsmenge),
Σ Alphabet (Eingabealphabet)
q_0 $\in Q$ (Startzustand)
F $\subseteq Q$ Finalzustände
δ Überführungsfunktion: $\delta : Q \times \Sigma_\lambda \to \mathcal{P}(Q)$ mit $\Sigma_\lambda = \Sigma \cup \{\lambda\}$. Es gilt $\delta(X, y) = \bigcup_{q \in X} \delta(q, y)$.

Bemerkung 35 Konfiguration von N: (q, u) mit dem aktuellen Zustand q und dem „Rest" der Eingabe u. Dies unterscheidet sich nicht von deterministischen endlichen Automaten.

Die Übergangsrelation: $\vdash \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$ ist nicht mehr rechtseindeutig. Es kann mehrere oder keine unmittelbare Nachfolgerkonfiguration geben.

Berechnung ist eine Menge von Konfigurationen, die als Baum dargestellt werden kann. Pfade im Baum heißen Berechnungspfade. Die

Berechnung ist akzeptierend genau dann, wenn es einen akzeptierenden Berechnungspfad gibt, d. h. $L(N) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \cap F \neq \emptyset\}$.

Definition 20 Die erweiterte Überführungsfunktion δ^* ist:

- erster Teil: ohne λ-Übergänge:

$$\begin{aligned}\delta^*(q,\lambda) &= \{q\}\\ \delta^*(q,wx) &= \delta(\delta^*(q,w),x)\end{aligned}$$

- zweiter Teil: mit λ-Übergängen: Für $P \subseteq Q$ sei $E(P) = \{q \in Q \mid q \in P \vee q$ ist in einem oder mehreren λ-Übergängen aus P erreichbar$\}$.

$$\begin{aligned}\delta^*(q,\lambda) &= E(q)\\ \delta^*(q,wx) &= E(\delta(E(\delta^*(E(q),w)),x))\end{aligned}$$

Bemerkung 36 $\mathcal{L}(\text{NEA})$ ist die Klasse der von NEA akzeptierten Sprachen. Jeder gewöhnliche endliche Automat A ist per Definition ein nichtdeterministischer Automat. Das heißt, $\mathcal{REG} \subseteq \mathcal{L}(\text{NEA})$.

Wir können zeigen: Wenn $L_1 \in \mathcal{L}(\text{NEA})$ und $L_2 \in \mathcal{L}(\text{NEA})$, dann ist $L_1 \circ L_2 \in \mathcal{L}(\text{NEA})$.

Bemerkung 37 Sei N ein NEA und $w \in \Sigma^*$. $w \in L(N) \iff \delta^*(q_0, w) \cap F \neq \emptyset$. Dabei ist $N = (Q, \Sigma, \delta, q_0, F)$ mit dem „Programm“ $\delta : Q \times \Sigma_\lambda \to \mathcal{P}(Q)$. Die Überführungsfunktion hat die Fortsetzung $\delta : \mathcal{P}(Q) \times \Sigma_\lambda \to \mathcal{P}(Q)$ durch $\delta(P, x) = \bigcup_{p \in P} \delta(p, x)$.

Satz 3 Die Klasse $\mathcal{L}(\text{NEA})$ ist abgeschlossen bezüglich Konkatenation, d. h. wenn L_1 durch einen NEA N_1 akzeptiert wird und L_2 durch einen NEA N_2 akzeptiert wird, dann gibt es einen NEA N, der $L_1 \circ L_2$ akzeptiert.

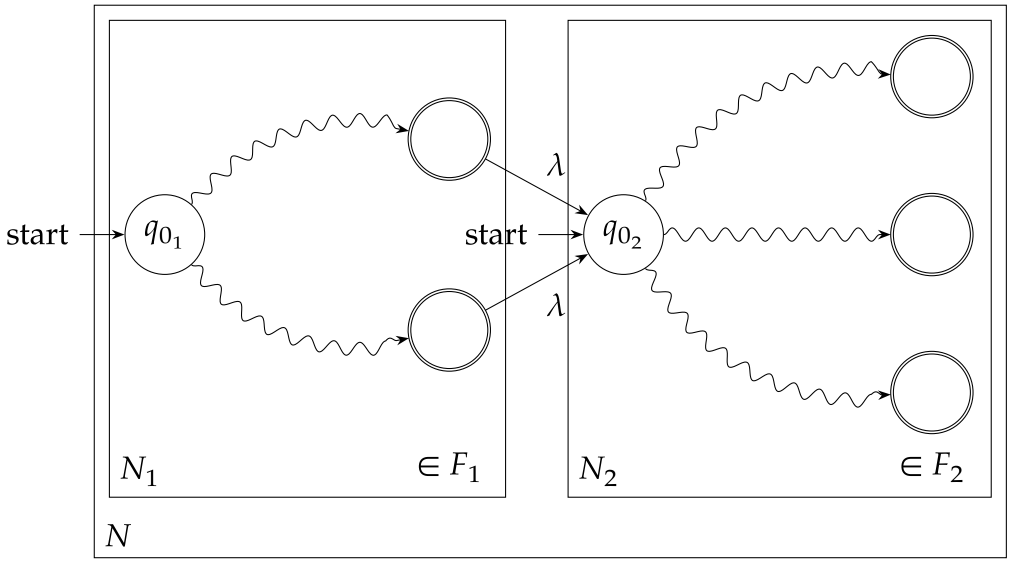
Beweis: Sei $N_i = (Q_i, \Sigma, \delta_i, q_{0_i}, F_i)$ mit $L(N_i) = L_i$ für $i = 1, 2$. Wir definieren $N = (Q, \Sigma, \delta, q_0, F)$ auf folgende Weise:

- $Q = Q_1 \cup Q_2$ (dabei ist $Q_1 \cap Q_2 = \emptyset$),
- $q_0 = q_{0_1}$ und
- $F = F_2$.

Wesentlich ist

$$\delta(q,x) = \begin{cases} \delta_1(q,x) & \text{für } q \in Q_1 \setminus F_1, x \in \Sigma_\lambda \\ \delta_1(q,x) & \text{für } q \in F_1, x \neq \lambda \\ \delta_1(q,x) \cup \{q_{0_2}\} & \text{für } q \in F_1, x = \lambda \\ \delta_2(q,x) & \text{für } q \in Q_2, x \in \Sigma_\lambda \end{cases} .$$

Abbildung 4 Diagramm zur Konkatenation von NEA (Satz 3)



Wir beweisen die Korrektheit von N, d. h. $L(N) = L_1 \circ L_2$. Sei

$$\begin{aligned} w \in L_1 \circ L_2 &\iff \exists u \in L_1 : \exists v \in L_2 : w = uv \\ &\iff \exists u \in L(N_1) : \exists v \in L(N_2) : w = uv \\ &\iff \exists u : \exists q' \in F_1 : q' \in \delta_1^*(q_1, u) \wedge \\ &\qquad \exists v : \exists q'' \in F_2 : q'' \in \delta_2^*(q_2, v) \wedge w = uv \\ &\iff \exists u : q_2 \in \delta^*(q_1, u) \wedge \\ &\qquad \exists v : \exists q'' \in F_2 : q'' \in \delta_2^*(q_2, v) \wedge w = uv \\ &\iff \exists q'' \in F_2 : q'' \in \delta^*(q_1, uv) \\ &\iff uv = w \in L(N) \end{aligned}$$

□

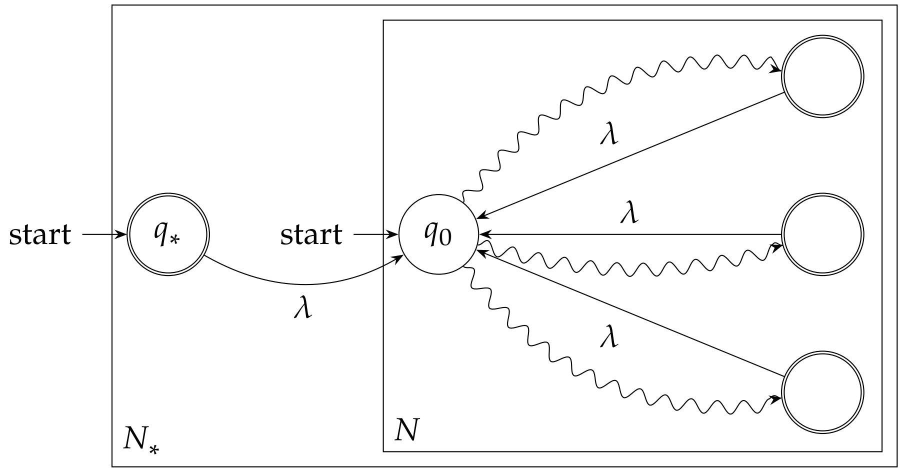
Satz 4 Die Klasse $\mathcal{L}$(NEA) ist abgeschlossen bzgl. der Kleene-Hülle, d. h. wenn L durch einen NEA akzeptiert wird, dann gibt es einen NEA N_*, der $L^* = \bigcup_{n\in\mathbb{N}} L^n$ (mit $L^0 = \{\lambda\}$) akzeptiert.

Beweis: Sei $N = (Q, \Sigma, \delta, q_0, F)$ mit $L(N) = L$. Wir definieren $N_* = (Q_*, \Sigma, \delta_*, q_*, F_*)$ mit

- $Q_* = Q \cup \{q_*\}$ (wobei $q_* \notin Q$),
- q_* als Startzustand und
- $F_* = F \cup \{q_*\}$.

$$\delta_*(q,x) = \begin{cases} \delta(q,x) & \text{für } q \in Q \setminus F, x \in \Sigma_\lambda \\ \delta(q,x) & \text{für } q \in F, x \neq \lambda \\ \delta(q,x) \cup \{q_0\} & \text{für } q \in F, x = \lambda \\ \{q_0\} & \text{für } q = q_*, x = \lambda \end{cases}$$

Abbildung 5 Diagramm zur Kleene-Hülle von NEA (Satz 4)



$w \in L^* \iff \exists n_0 : w \in L^{n_0}$.

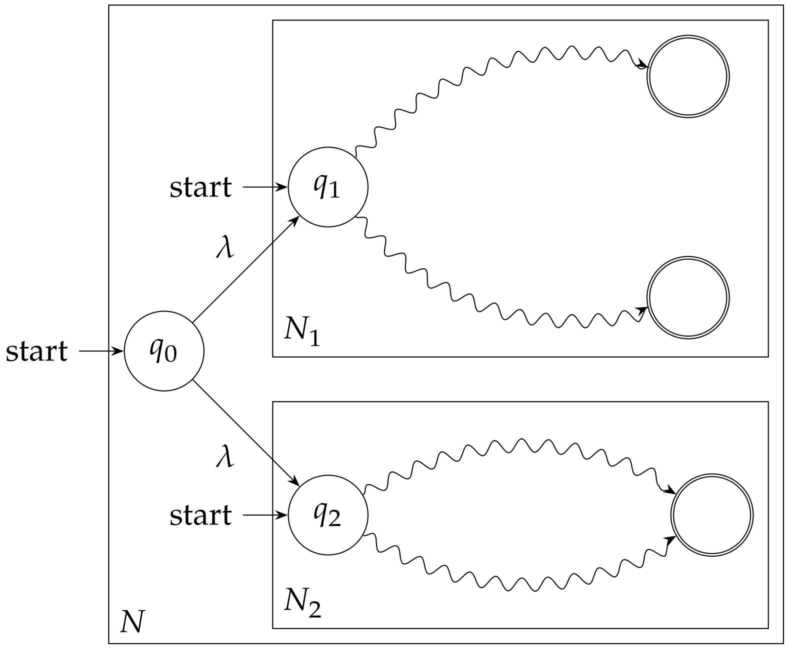
- Für $n_0 = 0$ ist $w \in L(N_*)$.
- Für $n_0 = 1$ ist $w \in L(N_*)$.
- Für $n_0 = 2$ ist $w \in L(N_*)$.
- usw. (vollständige Induktion)

Satz 5 Die Klasse $\mathcal{L}(\text{NEA})$ ist abgeschlossen bezüglich der Vereinigung, d. h. wenn L_1 durch einen NEA N_1 akzeptiert wird und L_2 durch einen NEA N_2 akzeptiert wird, dann gibt es einen NEA N, der $L_1 \cup L_2$ akzeptiert.

Beweis: Sei $N_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$ mit $L(N_i) = L_i$ für $i = 1, 2$. Wir definieren $Q = Q_1 \cup Q_2 \cup \{q_0\}$, den Startzustand q_0, $F = F_1 \cup F_2$.

$$\delta(q,x) = \begin{cases} \{q_1, q_2\} & \text{für } q = q_0, x = \lambda \\ \delta_1(q,x) & \text{für } q \in Q_1, x \in \Sigma_\lambda \\ \delta_2(q,x) & \text{für } q \in Q_2, x \in \Sigma_\lambda \end{cases}$$

Abbildung 6 Diagramm zur Vereinigung von NEA (Satz 5)



Bemerkung 38 Die drei Operationen Konkatenation, Kleene-Hülle und Vereinigung werden als *reguläre Operationen* bezeichnet. Die Sätze 3, 4, 5 bedeuten zusammengefasst: Die Klasse $\mathcal{L}(\text{NEA})$ ist abgeschlossen bzgl. der regulären Operationen.

Bemerkung 39 Wir wissen: Die Klasse $\mathcal{REG}$ ist abgeschlossen bzgl. der Booleschen Operationen.

2.4 Reguläre Operationen und reguläre Ausdrücke

Definition 21 *Syntax der regulären Ausdrücke*: Sei Σ ein Alphabet. Wir definieren die Syntax induktiv.

Induktionsanfang:

- x ist ein regulärer Ausdruck für $x \in \Sigma$.
- λ ist ein regulärer Ausdruck.
- $\emptyset$ ist ein regulärer Ausdruck.

Induktionsschritt: Wenn r, r_1 und r_2 bereits reguläre Ausdrücke sind, dann auch

- $(r_1 \circ r_2)$ $(r_1 r_2)$ $(r_1 \cdot r_2)$
- (r^*)
- $(r_1 \cup r_2)$ $(r_1 \mid r_2)$ $(r_1 + r_2)$

Beispiel 32 $\Sigma = \{a, b, c\}$. $(((a \circ b) \cup c)^*) = (a \circ b \cup c)^*$ ist ein regulärer Ausdruck.

Bemerkung 40 Klammereinsparungsregeln: * bindet stärker als $\circ$ bindet stärker als $\cup$.

Definition 22 *Semantik der regulären Ausdrücke*: Gemäß der Syntax induktiv:

Induktionsanfang:

- $L(x) = \{x\}$
- $L(\lambda) = \{\lambda\}$
- $L(\emptyset) = \emptyset$

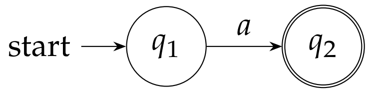
Induktionsschritt: Wenn $L(r)$, $L(r_1)$ und $L(r_2)$ bereits als formale Sprachen definiert sind, dann auch

- $L(r_1 \cup r_2) = L(r_1) \cup L(r_2)$
- $L(r^*) = (L(r))^*$
- $L(r_1 \circ r_2) = L(r_1) \circ L(r_2)$

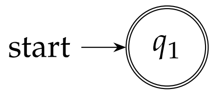
Bemerkung 41 Reguläre Ausdrücke beschreiben Sprachen. $\mathcal{L}(\text{RegA})$ ist die Klasse der Sprache, die durch reguläre Ausdrücke charakterisiert wird.

Bemerkung 42

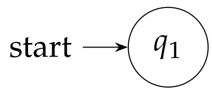
- Für $a \in \Sigma$ ist a ein regulärer Ausdruck. $L(a) = \{a\}$. Wir betrachten folgenden NEA mit $L(N) = \{a\}$.



- λ ist ein regulärer Ausdruck mit $L(\lambda) = \{\lambda\}$. Wir betrachten folgenden NEA mit $L(N') = \{\lambda\}$.



- $\emptyset$ ist ein regulärer Ausdruck mit $L(\emptyset) = \emptyset$. Wir betrachten folgenden NEA mit $L(N'') = \emptyset$.



Mit dieser Beobachtung und den Sätzen 3, 4 und 5 gilt die Folgerung $\mathcal{L}(\text{RegA}) \subseteq \mathcal{L}(\text{NEA})$[6]. Insbesondere gilt für alle Alphabete Σ:

$$\mathcal{L}(\text{RegA})(\Sigma) \subseteq \mathcal{L}(\text{NEA})(\Sigma).$$

Satz 6 Hauptsatz: $\mathcal{L}(\text{NEA}) \subseteq \mathcal{REG}$, d. h. für jede Sprache $L \subseteq \Sigma^*$, die durch einen NEA N akzeptiert wird, gibt es einen deterministischen endlichen Automaten A, der L akzeptiert.

[6] Wir wissen auch, dass $\mathcal{REG} \subseteq \mathcal{L}(\text{NEA})$.

Beweis: Sei $N = (Q, \Sigma, \delta, q_0, F)$ ein NEA mit $L(N) = L$. Wir konstruieren den *Potenzmengen-Automat* $A = (Q', \Sigma, \delta', q_0', F')$ auf folgende Weise:

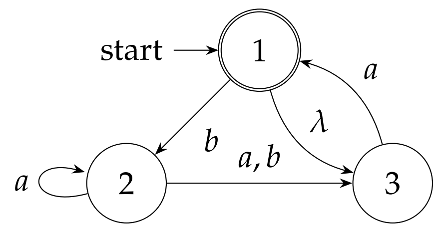
- Die Zustandsmenge $Q' = \mathcal{P}(Q)$.
- Im Allgemeinen kann N λ-Übergänge haben. Für $P \subseteq Q$ ist

$$E(P) = \{q \in Q \mid q \text{ ist durch keinen, einen oder mehrere } \lambda\text{-Übergänge aus einem Zustand aus } P \text{ erreichbar}\}$$

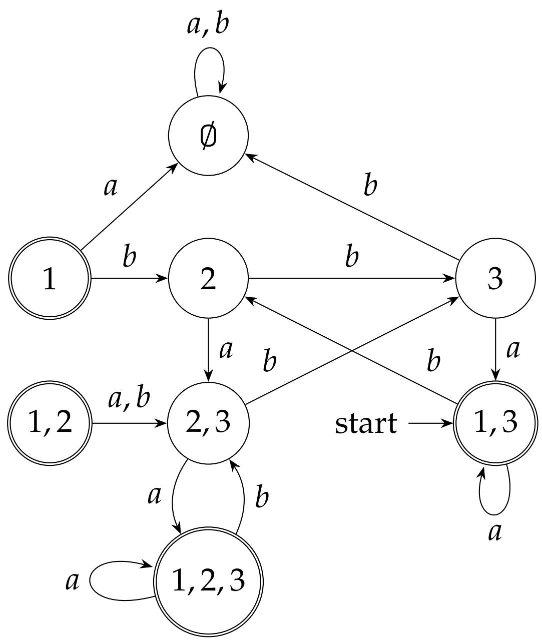
- Der Startzustand $q_0' = E(q_0)$.
- Die Finalzustände $F' = \{P \subseteq Q \mid P \cap F \neq \emptyset\}$.
- Die Überführungsfunktion $\delta'(R, a) = \bigcup_{r \in R} E(\delta(r, a))$ mit $R \subseteq Q$.

Korrektheit von A: Sei $w = w_1 w_2 ... w_n \in \Sigma^*$ mit $w \in L(N)$, also gilt $\delta^*(q_0, w) \cap F \neq \emptyset$. Also gibt es einen akzeptierenden Berechnungspfad, d. h. es gibt eine Folge $P_1, P_2, ..., P_l \subseteq Q$ mit $l \geq n$ und $w = w_1 w_2 ... w_n = x_1 x_2 x_3 ... x_l$ mit $x_j \in \Sigma_\lambda$, sodass gilt $\delta(q_0, x_1) = P_1$, $\delta(P_1, x_2) = P_2$, ..., $\delta(P_{l-1}, x_l) = P_l$ und $P_l \cap F \neq \emptyset$. Damit ist $P_l \in F'$ und $(\delta')^*(E(q_0), w) = P_l$. Also $w \in L(A)$. □

Beispiel 33 Wir betrachten den NEA N:



Dazu gehört der folgende äquivalente DEA:



2.5 Das Pumping-Lemma für reguläre Sprachen

Satz 7 $\mathcal{REG} = \mathcal{L}(\text{RegA})$

Bemerkung 43 Reguläre Sprachen sind

- Sprachen mit einem endlichen Muster
- Sprachen, die modulo m zählen können
- typische STOTTOTTOTTERER-Sprachen

Reguläre Sprachen sind keine

- Sprachen mit einem „globalen" Muster, z. B. Spiegelwörter, Doppelwörter etc.
- Sprachen, die zählen können, z. B. $|w|_a = |w|_b$ etc.

Satz 8 Das *Pumping-Lemma* für reguläre Sprachen (auch „uvw-Theorem"): Wenn $L \subseteq \Sigma^*$ eine reguläre Sprache ist, dann existiert eine natürliche Zahl $n_L \geq 1$, sodass für alle Wörter $z \in L$ mit $|z| \geq n_L$ existiert eine Zerlegung $z = uvw$ mit folgenden Eigenschaften:

1. $|uv| \leq n_L$
2. $|v| \geq 1$ $(v \neq \lambda)$
3. Für alle i ist $uv^iw \in L$.

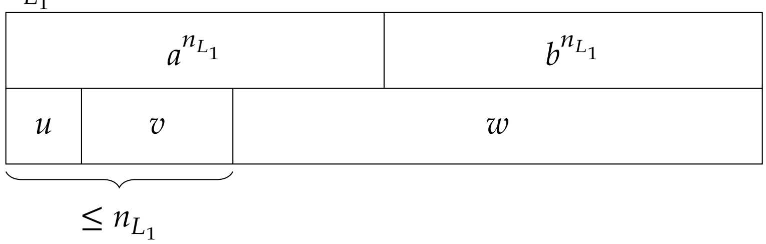
Bemerkung 44 Das Pumping-Lemma ist ein wirksames Instrument zum Nachweis der *Nicht*-Regularität einer Sprache.

Beispiel 34 $L_1 = \{a^n b^n \mid n \in \mathbb{N}\}$.

Indirekter Beweis der Nichtregularität: Angenommen $L_1 \in \mathcal{REG}$. Dann existiert eine natürliche Zahl n_{L_1}, sodass für alle Wörter z mit $|z| \geq n_{L_1}$ eine Zerlegung $z = uvw$ mit den Eigenschaften 1, 2, 3 existiert.

Wir setzen $z = a^{n_{L_1}} b^{n_{L_1}} \in L_1$ mit $|z| = 2 \cdot n_{L_1} \geq n_{L_1}$. Also existiert eine Zerlegung $z = uvw$ mit

1. $|uv| \leq n_{L_1}$

   Das heißt, $uv \in \{a\}^*$.
2. Insbesondere gilt $v \in \{a\}^*$, d. h. $v = a^m$ mit $m \geq 1$.

Damit ist wegen 3: $z_0 = uv^0w = uw \in L$. Aber $z_0 = a^{n_{L_1}-m} b^{n_{L_1}} \notin L$. □

Beispiel 35 $L_2 = \{w \in \{a, b\}^* \mid |w|_a = |w|_b\}$.

Behauptung: $L_2 \notin \mathcal{REG}$. Beweis mit dem Pumping-Lemma. Dazu können Sie den obigen Beweis nahezu wörtlich abschreiben.

Bemerkung 45 Beweis zum Pumping-Lemma (Satz 8): Sei L eine reguläre Sprache. Dann gibt es einen deterministischen endlichen Automaten $A = (Q, \Sigma, \delta, q_1, F)$, der L akzeptiert. Wir definieren $n_L = |Q|$.

Sei $z = z_1 z_2 \ldots z_l \in L(A)$ mit $l \geq n_L$. Dann gibt es eine akzeptierende Berechnung $B_A(z) = (K_0, K_1, \ldots, K_l)$ von A mit $K_0 = \text{Start-}K_A(z)$, $K_{i-1} \vdash K_i$ für $i = 1, \ldots, l$ und K_l ist akzeptierende Finalkonfiguration.

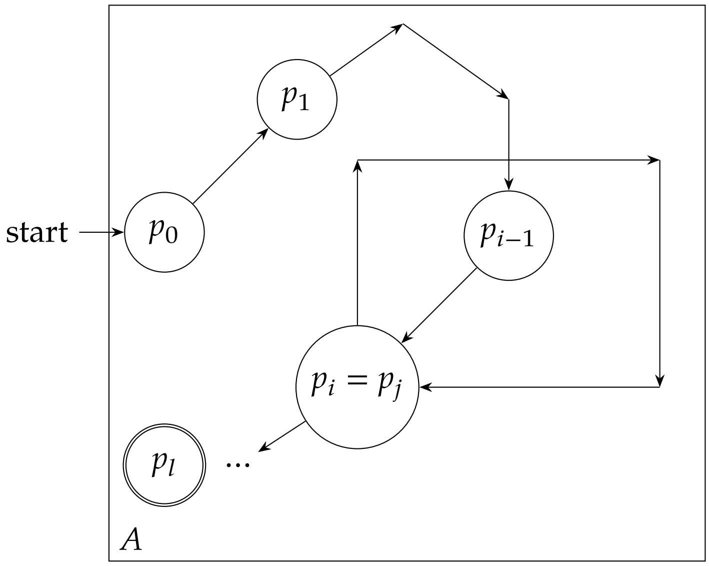
Es sei weiter $(p_0, p_1, \ldots, p_l)$ die zugehörige Folge der Zustände. Für die Länge dieser Folge gilt $l + 1 \geq n_L + 1$. Also gibt es ein Paar (i, j), für das gilt:

- $p_j = p_i$ und
- (i, j) ist das erste Paar mit dieser Eigenschaft und $i < j$,

das heißt $\underbrace{p_0, p_1, \ldots, p_{i-1}, p_i, p_{i+1}, \ldots, p_{j-1}}_{\leq n_L}, p_j, p_{j+1}, \ldots, p_l$.

Die Berechnung ist in Abbildung 7 im Diagramm dargestellt.

Abbildung 7 Diagramm zur Berechnung im Beweis des Pumping-Lemmas (Bemerkung 45)



Wir definieren $u = z_1 z_2 ... z_i$, $v = z_{i+1} ... z_{j-1} z_j$ und $w = z_{j+1} ... z_l$. Hierfür gilt:

1. $|uv| \leq n$, da $|z_1 ... z_j| = |uv|$ und die Länge der Folge $(p_0, ..., p_{j-1})$ durch n_L beschränkt.
2. $|v| \geq 1$, da $j \neq i$ und $i < j$.

Es bleibt 3. $uv^i w \in L$ für alle i zu zeigen.

Für $i = 0$ gilt $u \sim_A uv$ für $q = p_i = p_j$. Denn $x \sim_A y \iff (q_1, x) \vdash^* (q, \lambda) \wedge (q_1, y) \vdash^* (q, \lambda)$, d. h. es existiert $q \in Q$ mit $x \in [q]$ und $y \in [q]$.

$u \sim_A uv$ bedeutet: Für alle Wörter $y \in \Sigma^*$ gilt $uy \sim_A uvy$, insbesondere $uy \in L(A) \iff uvy \in L(A)$. Das heißt, für $y = w$, dass $uw \in L(A) \iff uvw \in L(A)$, also $z \in L(A) \implies uv^0 w \in L(A)$.

Weiter für $y = vw$: $uvw \in L(A) \iff uvvw \in L(A)$. Da $uvw = z \in L(A)$, ist $uv^2 w \in L(A)$. Analog für $y = v^2 w$: $uv^2 w \in L(A) \iff uvv^2 w \in L(A)$. Da $uv^2 w \in L(A)$, gilt $uv^3 w \in L(A)$. □

Beweise für Nicht-Regularität

Beispiel 36 Spiegelwörter: $L_{\text{pal}} = \{w \in \{a, b\}^* \mid w = \mathrm{Sp}(w)\}$. Behauptung: $L_{\text{pal}} \notin \mathcal{REG}$.

Indirekter Beweis: Angenommen $L_{\text{pal}} \in \mathcal{REG}$, dann gibt es eine Pumping-Zahl n_*, sodass für alle Wörter $z \in L_{\text{pal}}$ mit $|z| \geq n$ eine Zerlegung $z = uvw$ mit den Eigenschaften 1, 2 und 3 existiert.

Wir definieren $z = a^{n_*}b^{n_*}b^{n_*}a^{n_*}$. Es gilt $z \in L_{\text{pal}}$.

<table>
<tr><td colspan="3">a^{n_*}</td><td>b^{n_*}</td><td>b^{n_*}</td><td>a^{n_*}</td></tr>
<tr><td>u</td><td>v</td><td colspan="4">w</td></tr>
</table>

Für eine Zerlegung $z = uvw$ mit

1. $|uv| \leq n_*$ gilt $uv \in \{a\}^*$ und
2. $|v| \geq 1$ gilt $v \in \{a\}^*$.

Damit die zweite Eigenschaft erfüllt ist, sei $v = a^m$ mit $m \geq 1$. Wir betrachten $z_0 = uv^0w = uw = a^{n_*-m}b^{n_*}b^{n_*}a^{n_*}$. Einerseits sollte $z_0 \in L_{\text{pal}}$, aber tatsächlich ist $z_0 \notin L_{\text{pal}}$. Widerspruch. $\square$

Beispiel 37 Wir betrachten $L'_{\text{pal}} = \{w \in \{a,b\}^* \mid \exists u \in \{a,b\}^* : w = u\text{Sp}(u)\}$.

- $aabaa \in L_{\text{pal}}$, aberr $aabaa \notin L'_{\text{pal}}$.
- $L'_{\text{pal}} \subseteq L_{\text{pal}}$.
- $L_{\text{pal}} \cap \left(\bigcup_{n\in\mathbb{N}} \Sigma^{2n}\right) = L'_{\text{pal}}$.

Es gilt $L'_{\text{pal}} \notin \mathcal{REG}$.

Beispiel 38 Spiegelwörter über einem 1-buchstabigen Alphabet: $L_{\text{pal-1}} = \{w \in \{a\}^* \mid w = \text{Sp}(w)\} = \{a\}^* = \{a^0, a^1, a^2, \ldots\}$. $L_{\text{pal-1}} \in \mathcal{REG}$.

$L'_{\text{pal-1}} = \{w \in \{a\}^* \mid \exists u \in \{a\}^* : w = u\text{Sp}(u)\} = \{\lambda, a^2, a^4, \ldots\}$. $L'_{\text{pal-1}} \in \mathcal{REG}$.

Beispiel 39 Doppelwörter: $L_{\text{dup}} = \{ww \mid w \in \{a,b\}^*\}$. Behauptung: $L_{\text{dup}} \notin \mathcal{REG}$.

Aber: $L_{\text{dup-1}} = \{ww \mid w \in \{a\}^*\} = \{a^0, a^2, \ldots\}$. $L_{\text{dup-1}} \in \mathcal{REG}$.

Bemerkung 46 Sind alle 1-buchstabigen Sprachen regulär? Nein.

Beispiel: $L_{\text{quad}} = \{a^m \mid m = n \text{ ist Quadratzahl}\} = \{a^{n^2} \mid n \in \mathbb{N}\}$. Behauptung: $L_{\text{quad}} \notin \mathcal{REG}$.

Indirekter Beweis mit Pumping-Lemma: Angenommen $L_{\text{quad}} \in \mathcal{REG}$. Dann gibt es eine Pumping-Zahl n_0, sodass für alle Wörter $z \in L_{\text{quad}}$ mit $|z| \geq n_0$ eine Zerlegung $z = uvw$ mit den Eigenschaften 1, 2, 3 besitzen.

Wir definieren $z = a^{n_0^2} \in L_{\text{quad}}$. Es gilt wegen 1 $|uv| \leq n_0$ und wegen 2 $|v| \geq 1$ mit $|v| = m$, d. h. $v = a^m$ mit $m \geq 1$. Dann gilt für $z_2 = uv^2w = uvwv = zv$ mit $|z_2| = |z|+|v| = n_0^2+m \leq n_0^2+n_0 < n_0^2+2n_0+1 = (n_0+1)^2$. Damit gilt $n_0^2 < |z_2| < (n_0 + 1)^2$. Das heißt, $z_2 \notin L_{\text{quad}}$. Andererseits wegen 3 sollte $z_2 \in L_{\text{quad}}$. □

Bemerkung 47 *Kontraposition des Pumping-Lemmas*: Für alle natürlichen Zahlen $n \geq 1$ gibt es ein Wort $z \in L$ mit $|z| \geq n$, sodass es für alle Zerlegungen $z = uvw$ mit den Eigenschaften 1 und 2 ein $i \in \mathbb{N}$ mit $z_i = uv^iw \notin L$ gibt.

Wenn die Kontraposition des Pumping-Lemmas gilt, dann ist $L \notin \mathcal{REG}$.

Klammersprachen

Arithmetische Terme wie $((a+b)\cdot(c+d))^2\cdot(e+f)$ werden formal induktiv definiert. Die Klammersprache umfasst die Klammern der korrekt definierten Terme. Im Beispiel $(()())()$, $(((()())())())()$ usw.

Definition 23 Die *Dyck-Sprache* beschreibt korrekte Klammerwörter.

$$D = \{w \in \{(,)\}^* \mid \text{für } w = w_1w_2...w_n \text{ gilt: } |w|_(= |w|_) \text{ und}$$
$$\text{für alle Präfixe } w_1w_2...w_i \text{ gilt } |w_1...w_i|_(\geq |w_1...w_i|_)\}$$

Bemerkung 48 Beweis der Nichtregularität von D mit dem Pumping-Lemma: Für ein vorgegebenes n bestimmen wir $z = (^n)^n$. Für eine vorgegebene Zerlegung $z = uvw$ mit den Eigenschaften 1 und 2,

1. $|uv| \leq n$: $uv \in \{(\}^*$ und
2. $|v| = m \geq 1$: $v = (^m$,

gilt für $i = 0$: $z_0 = uw = (^{n-m})^n \notin L$. □

„Rückrichtung" des Pumping-Lemmas

Wir betrachten über $\Sigma = \{a, b, c\}$ die folgende Sprache:

$$L = \{a^ib^jc^k \mid i, j, k \in \mathbb{N} \text{ mit } i = 0 \text{ oder } j = k\}.$$

L sieht aus wie nicht regulär. Aber L erfüllt die Forderungen des Pumping-Lemmas: Es gibt eine natürliche Zahl $n = 1$, sodass für alle Wörter $z \in L$ mit $|z| \geq 1$ gilt: Es gibt eine Zerlegung $z = uvw$ mit $u = \lambda$, d. h. $z = vw$, sodass für alle Zahlen $i \in \mathbb{N}$ $z_i \in L$.

Um dies zu zeigen, machen wir eine Fallunterscheidung:

1. Fall: $v = a$, d. h. $i \neq 0$ und $w = a^r b^j c^j$.
2. Fall: $v = b$, d. h. $i = 0$ und $j \neq 0$ und $w = b^r c^k$.
3. Fall: $v = c$, d. h. $i = 0$ und $j = 0$ und $w = c^r$.

Diese Zerlegung erfüllt die drei Eigenschaften:

1. $|uv| = |v| = 1 \leq n = 1$, da $u = \lambda$.
2. $|v| = 1 \geq 1$.
3. in den drei Fällen:
   1. $z_i = uv^i w = v^i w = a^i a^r b^j c^j = a^s b^j c^j \in L$.
   2. $z_i = uv^i w = v^i w = b^i b^r c^k = b^t c^k \in L$.
   3. $z_i = uv^i w = v^i w = c^i c^r = c^t \in L$.

Bemerkung 49 Fazit: Das Pumping-Lemma ist ein Werkzeug zum Nachweis der *Nicht-Regularität* einer Sprache, aber nicht zum Nachweis der Regularität.

Beispiel 40 Gegeben sei eine injektive Funktion $f : \mathbb{N} \mapsto \mathbb{N}$[7] und $L_f = \{a^n \# a^m \mid n \in \mathbb{N}, f(n) = m\} = \{a^n \# a^{f(n)} \mid n \in \mathbb{N}\}$.

L_f ist nicht regulär. Der Beweis erfolgt mit der Kontraposition des Pumping-Lemmas. Wenn L regulär ist, dann gibt es eine Pumping-Zahl n_0. Wir wählen $z = a^{n_0} \# a^{f(n_0)} \in L_f$. Für alle Zerlegungen $z = uvw$ gilt:

1. $|uv| \leq n_0$
2. $|v| \geq 1$

Daraus folgt, dass $uv \in \{a\}^*$ und $v = a^k$ mit $1 \leq k \leq n_0$. Nun zeigen wir, dass ein $i \in \mathbb{N}$ existiert, für das gilt $uv^i w \notin L_f$. $uviw = a^{n+(i-1)\cdot k} \# a^{f(n)}$. Wir wählen $i = 2$. Dann ist $uv^2 w = a^{n+k} \# a^{f(n)} \in L \iff f(n+k) = f(n)$. Wegen der Linkseindeutigkeit gilt dies genau dann, wenn $n + k = n$. Das ist ein Widerspruch und damit die dritte Bedingung des Pumping-Lemmas verletzt.

Beispiel 41 $L(c,d) = \{v \in \{a\}^* \mid \exists n \in \mathbb{N} : |v| = c + d \cdot n\} = \{v \in \{a\}^* \mid |v|_a \equiv_d c\}$. $L(c) = \{v \in \{a\}^* \mid v = a^c\}$ und $L(d) = \{v \in \{a\}^* \mid |v|_a \equiv_d 0$. $L = L(c) \circ L(d)$. L ist regulär, da DEA modulo rechnen $(L(c))$ und endlich zählen $(L(d))$ können.

$L' = \{a^m a^n a^{m+n} \mid m, n \in \mathbb{N}\} = \{a^{2(m+n)} \mid m, n \in \mathbb{N}\} = \{a^{2k} \mid k \in \mathbb{N}\} = L(0, 2)$. L' ist regulär, da L regulär ist.

---

[7] $f \subseteq \mathbb{N} \times \mathbb{N}$ ist eine rechtseindeutige $(\forall x, y_1, y_2 \in \mathbb{N} : (x, y_1) \in f \wedge (x, y_2) \in f \implies y_1 = y_2)$, linkseindeutige $(\forall x_1, x_2, y \in \mathbb{N} : (x_1, y) \in f \wedge (x_2, y) \in f \implies x_1 = x_2)$ und linkstotale $(\forall x \in \mathbb{N} : \exists y \in \mathbb{N} : (x, y) \in f)$ binäre Relation.

2.6 Satz von Myhill und Nerode

Bemerkung 50 Wir kennen die A-Äquivalenz für einen endlichen Automaten A. Für $x, y \in \Sigma^*$ gilt $x \sim_A y \iff \exists q \in F : x \in [q] \wedge y \in [q]$ mit der Eigenschaft $x \sim_A y \implies \forall z \in \Sigma^* : xz \sim_A yz$.

Außerdem kennen wir die L-Äquivalenz für formale Sprachen $L \subseteq \Sigma^*$. Für $x, y \in \Sigma^*$ gilt $x \sim_L y \iff \forall z \in \Sigma^* : xz \in L \iff yz \in L$. Für jeden beliebigen endlichen Automaten A mit $L(A) = L$ gilt: Wenn $x \sim_A y$, dann $x \sim_{L(A)} y$.

Jede Klasse $[u]_L$ der L-Äquivalenz besteht aus einer oder mehreren Klassen $[q]_A$ der A-Äquivalenz. Mit anderen Worten: Zwei Zustände $p, q \in Q_A$ mit $\delta_A^*(p, z) = \delta^*(q, z)$ für alle $z \in \Sigma^*$ lassen sich „vereinen“.

Lemma 2 Für eine Sprache $L \subseteq \Sigma^*$ gilt

1. wenn $x \sim_L y$ und $x \in L$, dann ist $y \in L$.
2. wenn $x \sim_L y$ und $a \in \Sigma$, dann ist $xa \sim_L ya$.

Definition 24 Für eine Sprache L ist der *Index* (der Sprache; Nerode-Index) als Kardinalzahl der Faktormenge bezüglich der L-Äquivalenz definiert, d. h. $\mathrm{ind}(L) = \mathrm{card}(\Sigma^*/\sim_L)$.

Bemerkung 51

1. Fall: $\mathrm{card}(\Sigma^*/\sim_L)$ ist endlich, d. h. es gibt eine natürliche Zahl n mit $n = \mathrm{card}(\Sigma^*/\sim_L)$. Dann ist L eine Sprache mit Index n.
2. Fall: $\mathrm{card}(\Sigma^*/\sim_L)$ ist nicht endlich, d. h. $\mathrm{card}(\Sigma^*/\sim_L) = \aleph_0$. Dann ist L eine Sprache mit unendlichem Index.

Beispiel 42 $L = \{w \in \{0,1\}^* \mid |w| \geq 2 \text{ und vorletzter Buchstabe von } w \text{ ist } 0\}$. Wir suchen den Index von L.

Idee: Wir betrachten die Suffixe:

$[00]_L$	$\{00, 000, 110, 0000, ...\}$
$[01]_L$	$\{01, 001, 101, 0001, ...\}$
$[10]_L$	$\{0, 10, 010, 110, ...\}$
$[11]_L$	$\{\lambda, 1, 11, 111, ...\}$

Außerdem gilt: Je zwei dieser vier Wörter sind paarweise nicht äquivalent. Also ist $\mathrm{ind}(L) = 4$.

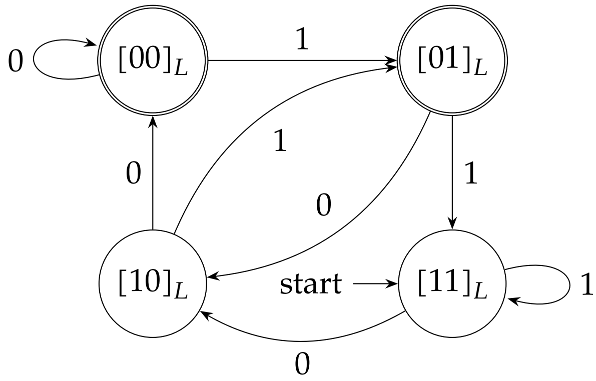
Beispiel 43 $L = \{w \in \{0,1\}^* \mid \exists n \in \mathbb{N} : w = 0^n 1^n\} = \{0^n 1^n \mid n \in \mathbb{N}\}$.

Es gilt $\neg(\lambda \sim_L 01)$ und $\neg(0 \sim_L 001)$ und $\neg(00 \sim_L 0001)$ usw. Also besitzt L unendlichen Index.

Satz 9 Satz von Myhill/Nerode:

- Teil 1: Wenn $L \in \mathcal{REG}$, dann hat L endlichen Index. Wenn $L \in \mathcal{REG}$, dann existiert ein DEA A mit $L(A) = L$. Also gilt $\mathrm{card}(\Sigma^*/\sim_L) \leq \mathrm{card}(\Sigma^*/\sim_A) = \mathrm{card}(Q) = k$ ist endlich.
- Teil 2: Wenn L endlichen Index hat, dann ist $L \in \mathcal{REG}$.

Beispiel 44 Konstruktion für das Beispiel 42:



Startzustand ist die Äquivalenzklasse mit λ. Finalzustände sind die Äquivalenzklassen, die die Wörter der Sprache enthalten.

Beispiel 45 Wir betrachten $L = \{w \in \{a,b\}^* \mid |w|_a \equiv_5 0\}$.

Die L-Äquivalenz mit der Zerlegung von $\{a,b\}^*$ hat folgende Äquivalenzklassen:

- $[\lambda] = \{w \in \Sigma^* \mid |w|_a \equiv_5 0\}$,
- $[a] = \{w \in \Sigma^* \mid |w|_a \equiv_5 1\}$,
- $[aa] = \{w \in \Sigma^* \mid |w|_a \equiv_5 2\}$,
- $[aaa] = \{w \in \Sigma^* \mid |w|_a \equiv_5 3\}$ und
- $[aaaa] = \{w \in \Sigma^* \mid |w|_a \equiv_5 4\}$.

Beispielhaft zeigen wir, dass a und aa nicht L-äquivalent sind. Wir wählen $v = aaaa$. Dann ist $av \in L$, aber $aav \notin L$.

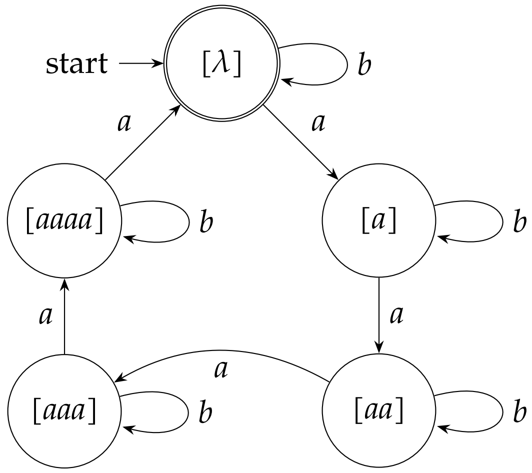
Ein beispielhafter Automat ist in Abbildung 8 zu sehen. Ein regulärer Ausdruck für diese Sprache ist $b^*(ab^*ab^*ab^*ab^*ab^*)^*$.

Beispiel 46 Wir betrachten $L = \{w \in \{a,b\}^* \mid aba \sqsubseteq w\}$.

Es ist die Faktormenge zu bestimmen:

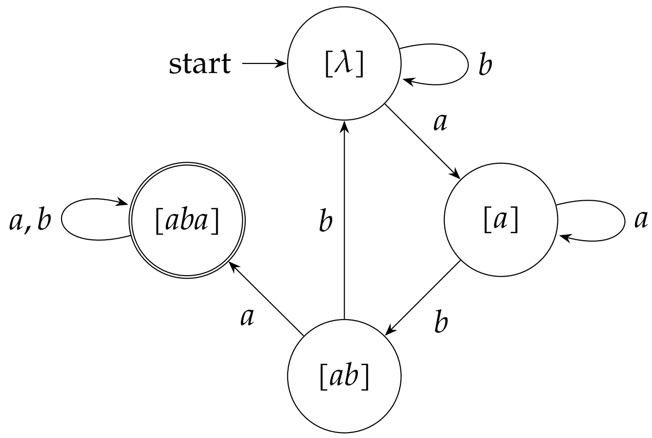
- $[\lambda] = \{w \in \Sigma^* \mid w \notin L \land \exists u \in \Sigma^* : w = ubb\} \cup \{b\} \cup \{\lambda\}$,
- $[aba] = \{w \in \Sigma^* \mid \exists u,v \in \Sigma^* : uabav\}$ (aba ist Teilwort von a),

Abbildung 8 Automat für Modulo-5-Rechnung aus Beispiel 45



- $[a] = \{w \in \Sigma^* \mid w \notin L \land \exists u \in \Sigma^* : w = ua\}$ (Suffix ist Präfix der Länge 1 von *aba*),
- $[ab] = \{w \in \Sigma^* \mid w \notin L \land \exists u \in \Sigma^* : w = uab\}$ (Suffix ist Präfix der Länge 2 von *aba*).

Abbildung 9 Automat zum Teilwortproblem aus Beispiel 46

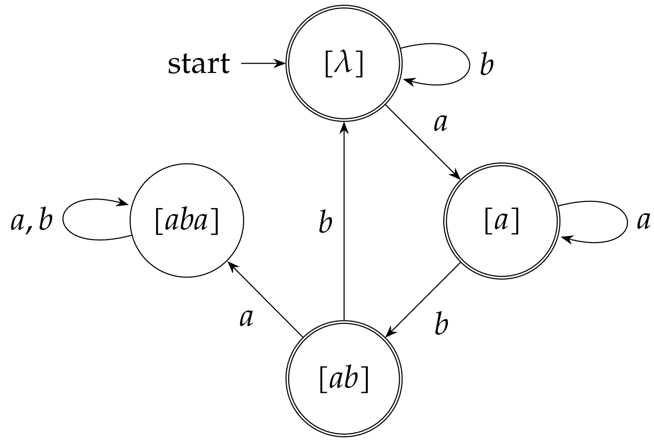


Ein beispielhafter Automat ist in Abbildung 9 zu sehen. Ein regulärer Ausdruck für diese Sprache ist $(a \mid b)^*aba(a \mid b)*$. Die Sprache ist also $L = \{w \in \Sigma^* \mid w = \{a,b\}^* \circ aba \circ \{a,b\}^*\}$.

Beispiel 47 Mit L aus Beispiel 46 und $L' = \{w \in \{a,b\}^* \mid \neg(aba \sqsubseteq w)\} = \overline{L}$.

Ein beispielhafter Automat ist in Abbildung 10 zu sehen. Es wird deutlich, dass sich im Automaten nur die Finalzustände geändert haben, nicht die Übergänge. Insbesondere ändern sich so die Äquivalenzklassen nicht.

Abbildung 10 Automat zum Teilwortproblem aus Beispiel 47



Bemerkung 52 $x \sim_L y \iff \forall v \in \Sigma^* : xv \in L \iff yv \in L$. Anhand dessen können wir für $L' = \overline{L}$ auch schreiben:

$$\begin{aligned} x \sim_{L'} y &\iff \forall v \in \Sigma^* : xv \in L' \iff yv \in L' \\ x \sim_{\overline{L}} y &\iff \forall v \in \Sigma^* : xv \in \overline{L} \iff yv \in \overline{L} \\ &\iff \forall v \in \Sigma^* : xv \notin L \iff yv \notin L \\ &\iff \forall v \in \Sigma^* : xv \in L \iff yv \in L \\ &\iff x \sim_L y \end{aligned}$$

Bemerkung 53 Nachtrag zum Satz von Myhill und Nerode (Satz 9): Aus der L-Äquivalenz folgt $\operatorname{ind}(L)$. Wenn $\operatorname{ind}(L)$ endlich ist, $\Sigma^*/\sim L = \{[x_1], [x_2], \ldots, [x_{k-1}], [x_0 = \lambda]\}$ (d. h. $\operatorname{ind}(L) = k$), dann können wir hierzu einen endlichen Automaten konstruieren, der L akzeptiert.

Der allgemeine Ansatz ist die Konstruktion eines Automaten $A = (Q, \Sigma, \delta, q_0, F)$, wobei $Q = \Sigma^*/\sim L$, $q_0 = [\lambda]$, $F = \{[x] \mid x \in L\}$[8] und $\delta : Q \times \Sigma \mapsto Q$ mit

$$\delta([x], a) = [xa] = [x] \text{ für } x \in Q, \text{ also } x \in \Sigma^*, \text{ und } a \in \Sigma.$$

Wir prüfen noch die Repräsentantenunabhängigkeit von δ, indem wir zeigen, dass gilt $\forall x, y \in \Sigma^* : x \sim_L y \implies [xa] \sim_L [ya]$. Dies entspricht der Aussage von Lemma 2[9].

---

[8] Es gilt $L = \bigcup\{[x] \mid x \in L\}$.

[9] Konkret zeigen wir $xa \sim_L ya$. Dazu muss gelten: Für alle $w \in \Sigma^* : (xa)w \in L \iff (ya)w \in L$. Nach Voraussetzung ist $x \sim_L y$ und damit gilt für alle $w \in \Sigma^*$ und $x(aw) \in L \iff y(aw) \in L$. Aus der Assoziativität der Konkatenation folgt die Äquivalenz der Aussagen und damit gilt die Aussage.

Zu zeigen ist noch die Korrektheit des Automaten, also $L(A) = L$. Es gilt $x \in L \iff [x] \in F \iff [\lambda x] \in F \iff \delta^*([\lambda], x) \in F \iff \delta^*(q_0, x) \in F \iff x \in L(A)$. Also ist der oben konstruierte Automat A *korrekt*.

Der so konstruierte Automat A ist auch *minimal*. Angenommen, es gibt einen DEA A' mit $|Q'| < k$. Dann gilt $k = \operatorname{ind}(L) \leq |Q'| < k$. Das ist ein Widerspruch.

Bemerkung 54 Die Klassen der L-Äquivalenz sind Vereinigungen von Klassen der A-Äquivalenz.

Beispiel 48 Sei $L_{\text{prim}} = \{a^p \mid p \text{ ist Primzahl}\} \subseteq \{a\}^*$. Es ist zu zeigen, dass L_{prim} unendlichen Index hat.

Seien $u, v \in \{a\}^*$ mit $u = a^p \neq a^r = v$. Dann sind p und r verschiedene Primzahlen. Sei o.B.d.A. $p < r$. Da die Differenzen zweier Primzahlen[10] nicht gleich sind, existiert ein ϵ, sodass $p + \epsilon \in \text{Prim}$ und $r + \epsilon \notin \text{Prim}$. Damit ist $ua^\epsilon \in L_{\text{prim}}$ und $va^\epsilon \notin L_{\text{prim}}$. Also ist $[u] \neq [v]$.

Damit ist jedes Wort der Sprache in seiner eigenen Äquivalenzklasse. Es ist bekannt, dass es unendlich viele Primzahlen gibt. Daher ist nach dem Satz von Myhill/Nerode der Index von L_{prim} unendlich.

Beispiel 49 Es ist mit dem Pumping-Lemma die Nichtregularität von L_{prim} zu zeigen.

Indirekter Beweis: Angenommen $L_{\text{prim}} \in \text{REG}$. Dann gibt es eine Pumpingzahl n, sodass für alle Wörter $z \in L_{\text{prim}}$ mit $|z| \geq n$ eine Zerlegung $z = uvw$ mit den Eigenschaften 1 ($|uv| \leq n$), 2 ($v \neq \lambda$) und 3 ($\forall i \in \mathbb{N} : uv^iw \in L$) existiert.

Sei p eine hinreichend große Primzahl ($p \gg n$) und $z = a^p \in L_{\text{prim}}$. Wir wählen die Zerlegung $z = uvw = a^\alpha a^\beta a^\gamma$, sodass $\alpha + \beta \leq n$ (Eigenschaft 1) und $\beta \geq 1$ (Eigenschaft 2). Damit ist $p = \alpha + \beta + \gamma$. Für alle i muss $uv^iw \in L$ sein (Eigenschaft 3), also muss $|uv^iw| = p + (i-1) \cdot \beta$ für alle i prim sein. Wir wählen $i = p+1$, dann ist $|uv^{p+1}w| = p + p \cdot \beta = p(1+\beta)$, welches nicht prim ist, da beide Faktoren echt größer als 1 sind. Damit ist $z_{p+1} \notin L_{\text{prim}}$ und $L_{\text{prim}} \notin \text{REG}$. $\square$

Beispiel 50 Besitzt L_{prim} eine unendliche reguläre Teilmenge?

Nein. Angenommen $L' \subsetneq L_{\text{prim}}$ ist regulär. Dann gilt für L' das Pumpinglemma. Also gibt es eine Pumping-Zahl $n_{L'}$, sodass es ein $z \in L'$

[10] Diese Abfolge von Zahlen ist als OEIS A001223 verzeichnet.

mit $|z| \geq n_{L'}$ gibt. Das heißt, $z = a^p$ für eine Primzahl p mit $p \geq n_{L'}$. Danach analog zu Beispiel 49

Beispiel 51 Gibt es eine nichtreguläre Sprache, die eine unendliche reguläre Teilmenge hat?

$$\begin{aligned} L &= L_{\text{prim}} \cup L_{\text{gerade}} \\ &= \{a^p \mid p \in \text{Prim}\} \cup \{a^{2n} \mid n \in \mathbb{N}\} \end{aligned}$$

Beispiel 52 Gibt es unendliche reguläre Sprachen, die unendliche nichtreguläre Teilmengen haben?

$\Sigma^* = \{a\}^*$ ist regulär, aber $L_{\text{prim}} \subsetneq \{a\}^*$ ist nicht regulär.

Beispiel 53 Ist $L^*_{\text{prim}} = \bigcup_{n \in \mathbb{N}} L^n_{\text{prim}}$ regulär?

Jede natürliche Zahl $2n = \underbrace{2 + 2 + \ldots + 2}_{n\text{-mal}}$. Und $2n + 1 = 2 + 2 + \ldots + 3$.

Damit ist $L^*_{\text{prim}} = \{a^n \mid a \neq 1\} = \{a\}^* \setminus \{a\}$. Das ist regulär.

Beispiel 54 Es die Faktormenge von $L_{\text{pal}} = \{w \in \{a, b\}^* \mid \text{Sp}(w) = w\}$ zu bestimmen.

Jedes Wort hat seine eigene Äquivalenzklasse genau dann, wenn $\forall u, v \in \Sigma^*, u \neq v : u \not\sim_L v$ genau dann, wenn $\exists l \in \Sigma^* : ul \in L_{\text{pal}} \iff vl \in L_{\text{pal}}$. Sei $u, v \in \Sigma^*$ und $u \neq v$ mit $|u| \leq |v|$. Dann ist $l = a\text{Sp}(u)$, also $ul \in L_{\text{pal}}$. Angenommen $vl \in L_{\text{pal}}$. Dann ist $l' = b\text{Sp}(u)$. Es gilt $ul' \in L_{\text{pal}}$. $vl = va\text{Sp}(u)$ und $vl' = vb\text{Sp}(u)$. Damit ist $vl = \text{Sp}(vl) = \text{Sp}(va\text{Sp}(u)) = ua\text{Sp}(v)$ und $vl' = \text{Sp}(vl') = \text{Sp}(vb\text{Sp}(u)) = ub\text{Sp}(v)$. Daraus folgt $u = v$. Widerspruch.

3 Die Chomsky-Hierarchie

3.1 Formale Grammatiken

Bemerkung 55 Natürliche Sprachen werden durch Grammatikregeln beschrieben, die korrekte Sätze erzeugen. Formale Sprachen werden durch formale Grammatiken erzeugt.

Bei einer „natürlichen Grammatik" unterscheiden wir zwei Sorten von Wörtern

- Metawörter: $\langle$Satz$\rangle$,
- richtige Wörter: Hund

und Grammatikregeln: ⟨Satz⟩ → ⟨Subjekt⟩⟨Prädikat⟩⟨Objekt⟩ sowie einem definierten Anfang: ⟨Satz⟩.

Definition 25 Eine *formale Grammatik* $G = (N, T, P, S)$ ist ein Viertupel der Form

N endliche Menge der *Nichtterminale* (Variable V),
T endliche Menge der *Terminale* (Alphabet Σ),
P endliche Menge der *Produktionen* (Regeln) mit $P \subseteq (N \cup T)^+ \times (N \cup T)^*$ und
S $\in N$ *Startsymbol*.

Es gilt $N \cap T = \emptyset$.

Bemerkung 56 Welchen Effekt hat eine Regel (p, q)? Sei $w \in (N \cup T)^+$ ein Zwischenwort der Form $w = w_l p w_r$. Dann wird $w' = w_l q w_r$ in einem *Ableitungsschritt* aus w erzeugt, $w \Rightarrow w'$.

Definition 26 w' ist *ableitbar* aus w genau dann, wenn es eine Folge $v_0, v_1, ..., v_k \in (N \cup T)^*$ gibt, sodass gilt: $w = v_0$, $v_0 \Rightarrow v_1$, $v_1 \Rightarrow v_2$, ..., $v_{k-1} \Rightarrow v_k$, $v_k = w'$. In diesem Fall ist w' in k Ableitungsschritten aus w ableitbar ($w \Rightarrow^k w'$).

Bemerkung 57 Stets ist $w \Rightarrow^0 w$. $\Rightarrow$ ist eine binäre Relation über $(N \cup T)^*$, also $\Rightarrow \subseteq (N \cup T)^* \times (N \cup T)^*$.

$\Rightarrow^*$ ist die reflexive und transitive Hülle von $\Rightarrow$. $w \Rightarrow^* w'$ bedeutet, w' ist aus w ableitbar.

Definition 27 Sei $G = (N, T, P, S)$ eine Grammatik vom Chomsky-Typ. Die von G *erzeugte Sprache* ist $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$.

Bemerkung 58 Sprechweisen: Eine formale Grammatik $G = (N, T, P, S)$ heißt auch Grammatik vom Chomsky-Typ beziehungsweise Grammatik vom Chomsky-Typ 0. Chomsky-Typ 0 heißt, dass es keine Einschränkungen für die Produktionen (p, q) – bis auf $p \neq \lambda$ – gibt.

CH(0) bezeichnet die Klasse aller Sprachen, die durch formale Grammatiken erzeugt werden können. Alle Programmiersprachen gehören zu CH(0).

Beispiel 55 Sei $G = (N, T, P, S)$ eine Grammatik mit $N = \{S, L, R\}$, $T = \{a, \#\}$ und

$$P = \{S \to \#aL\#, \quad (3)$$
$$aL \to Laa, \quad (4)$$
$$\#L \to \#R, \quad (5)$$
$$\#L \to \#, \quad (6)$$
$$Ra \to aaR, \quad (7)$$
$$R\# \to L\#, \quad (8)$$
$$R\# \to \#\}. \quad (9)$$

Beispiele für Ableitungen:

- $S \overset{(3)}{\Longrightarrow} \#aL\# \overset{(4)}{\Longrightarrow} \#Laa\# \overset{(6)}{\Longrightarrow} \#aa\# \in T^*$.
- $S \overset{(3)}{\Longrightarrow} \#aL\# \overset{(4)}{\Longrightarrow} \#Laa\# \overset{(7)}{\Longrightarrow} \#Raa\# \overset{(7)}{\Longrightarrow} \#aaRa\# \overset{(7)}{\Longrightarrow} \#aaaaR\# \overset{(9)}{\Longrightarrow} \#aaaa\# \in T^*$.
- $S \overset{(3)}{\Longrightarrow} \#aL\# \overset{(4)}{\Longrightarrow} \#Laa\# \overset{(7)}{\Longrightarrow} \#Raa\# \overset{(7)}{\Longrightarrow} \#aaRa\# \overset{(7)}{\Longrightarrow} \#aaaaR\# \overset{(8)}{\Longrightarrow} \#aaaaL\# \Rightarrow \ldots$.

$L(G) = \{\#a^{2^n}\# \mid n \geq 1\} = L_{\text{pot}} \in \mathrm{CH}(0) \setminus \mathcal{REG}$.

Lassen sich Einschränkungen an die Produktion formulieren, ohne die Erzeugungskraft der Grammatiken einzuschränken (Ziel: übersichtlichere Regeln)?

Definition 28 Zwei Grammatiken G_1 und G_2 sind *äquivalent*, $G_1 \sim G_2$, genau dann, wenn $L(G_1) = L(G_2)$.

Definition 29 Eine formale Grammatik $G = (N, T, P, S)$ ist eine *Normalform-Grammatik* vom Typ 0 genau dann, wenn für alle Produktionen gilt:

I. für alle Nicht-λ-Regeln gilt:
   1. $X \to YZ$,
   2. $XY \to Z$ oder
   3. $X \to a$,

   mit $X, Y, Z \in N$ und $a \in T$.
II. für alle λ-Regeln gilt: Es gibt höchstens eine λ-Regel.

Satz 10 Zu jeder beliebigen formalen Grammatik $G = (N, T, P, S)$ gibt es eine äquivalente Typ-0-Normalform-Grammatik G'.

Beweis in mehreren Globalschritten:

1. Globalschritt: Wir definieren neue Nichtterminale $T' = \{t' \mid t \in T\}$ für jedes $t \in T$ und ersetzen in jeder Regel $p \to q \in P$ jedes Terminal t durch t' und erhalten so die neue Menge P' und definieren neue Regeln $P_T = \{t' \to t \mid t \in T\}$.
   Im Ergebnis haben wir eine neue Grammatik $G_1 = (N \cup T', T, P' \cup P_T, S)$ mit der Eigenschaft: Alle Regeln mit Terminalen haben die Form 3, $X \to a$. Offensichtlich ist $G_1 \sim G$.
2. Globalschritt (Zwischenschritt): Reduzierung der λ-Regeln. Falls G λ-Regeln besitzt[11], definieren wir ein neues Nichtterminal L und

[11] λ-Regel: $p \to \lambda$ für $p \in (N \cup T)^+$.

ersetzen alle λ-Regeln durch $p \rightarrow L$ und neu $L \rightarrow \lambda$. Dadurch ist II gesichert. $G_2 \sim G$.

3. Globalschritt: Kürzen auf eine Seite mit Länge 1. In diesem Schritt bestimmen wir für jede Regel $p \rightarrow q$ mit $|p| > 1$ und $|q| > 1$ *ein neues* Nichtterminal B (B_i für $p \rightarrow q$) und zwei Regeln $p \rightarrow B_i$ und $B_i \rightarrow q$ anstelle von $p \rightarrow q$. Das Ergebnis ist $G_3 \sim G_2$.
4. Globalschritt: Kürze „lange" rechte Seiten. Sei $X \rightarrow Y_1Y_2...Y_n$ mit $n > 2$ eine Regel aus G_3. Für jede solche Regel definieren wir neue Nichtterminale $C_1, C_2, ..., C_{n-2}$ und neue Regeln $X \rightarrow Y_1C_1$, $C_1 \rightarrow Y_2C_2$, $C_2 \rightarrow Y_3C_3$, ..., $C_{n-3} \rightarrow Y_{n-2}C_{n-2}$, $C_{n-2} \rightarrow Y_{n-1}Y_n$ ($n-1$ neue Regeln anstelle der ursprünglichen). Im Ergebnis erhalten wir $G_4 \sim G_3$. Damit ist 1 erfüllt.
5. Globalschritt: Kürze „lange" linke Seiten. Sei $X_1X_2...X_n \rightarrow Y$ eine Regel mit $n > 2$. Für jede solche Regel definieren wir neue Nichtterminale $D_1, D_2, ..., D_{n-2}$ und neue Regeln $X_1X_2 \rightarrow D_1$, $D_1X_3 \rightarrow D_2$, $D_2X_4...D_3$, ..., $D_{i-2}X_i \rightarrow D_{i-1}$, ..., $D_{n-3}X_{n-1} \rightarrow D_{n-2}$, $D_{n-2}X_n \rightarrow Y$ ($n-1$ neue Regeln ersetzen die ursprüngliche). Im Ergebnis erhalten wir $G_5 \sim G_4$. Damit ist 2 erfüllt.
6. Globalschritt: Beseitige 1:1-Regeln. Falls es Regeln der Form $X \rightarrow Y$ gibt, werden für jede solche Regel zwei neue Nichtterminale U, V und zwei neue Regeln $X \rightarrow UV$ und $UV \rightarrow Y$ definiert. Im Ergebnis erhalten wir $G_6 \sim G_5 \sim G_4 \sim G_3 \sim G_2 \sim G_1 \sim G$. □

Bemerkung 59

- Beim Entwurf von Grammatiken (z. B. für Programmiersprachen) können alle Operationen für die Regeln ausgenutzt werden.
- Beim Nachweis von strukturellen Eigenschaften einer von einer Grammatik erzeugten Sprache kann man sich auf Einschränkungen einer Normalform-Grammatik zurückziehen.

Fazit: Typ-0-Normalform-Grammatiken haben Einschränkungen an die *Form der Regeln*, nicht aber an die *Kapazität* der Erzeugung von Sprachen.

Beispiel 56 Wir setzen Beispiel 55 anhand des Beweises des Satzes fort.

Nach Globalschritt 1: $T' = \{H, A\}$ mit $N_1 = \{S, L, R\} \cup \{H, A\}$.

$$\begin{aligned} P_1 = \{S \rightarrow HALH, \quad && HL \rightarrow H, \quad && RH \rightarrow H, \\ AL \rightarrow LAA, \quad && RA \rightarrow AAR, \quad && A \rightarrow a, \\ HL \rightarrow HR, \quad && RH \rightarrow LH, \quad && H \rightarrow \#\}. \end{aligned}$$

Bemerkung 60 *Wortproblem für Typ-0-Grammatiken*

- Eingabe: ein Paar (G, w) mit einer Typ-0-Grammatik G und ein Wort $w \in T^*$
- Frage: $w \in L(G)$?

„Übersetzung“: Gegeben ist eine Grammatik G für eine Programmiersprache L und ein „Programm“ P. Ist P ein syntaktisch korrektes Programm in L ($P \in L$)? Falls nicht ($P \notin L$), möchte man sogar noch eine Fehleranzeige. So etwas leisten sogenannte Parser.

Schlechte Nachricht: Für Grammatiken vom Typ 0 ist das Wortproblem algorithmisch nicht lösbar. Das heißt, es gibt eine Typ-0-Grammatik, für die es keinen Parser gibt.

Erinnerung an „Ein Wörterbuch der Algorithmen“ mit der Textberechenbarkeit (Vorschriften zur Berechnung von Zahlenfunktionen). Indirekter Beweis: Es gibt keinen Algorithmus, der bei Eingabe einer Vorschrift T entscheidet, ob T eine Vorschrift zur Berechnung einer total definierten Zahlenfunktion ist.

3.2 Chomsky-Grammatiken vom Typ 1

Motivation: Da Typ-0-Grammatiken eine zu große Kapazität besitzten („unlösbare Probleme“), formulieren wir Einschränkungen der *Stärke der Regeln*, die die Kapazität reduzieren.

Definition 30 Eine formale Grammatik $G = (N, T, P, S)$ ist eine *Typ-1-Grammatik* genau dann, wenn

I. für alle Nicht-λ-Regeln gilt: Wenn $p \to q \in P$, dann gilt $|p| \leq |q|$ (Grammatik vom Erweiterungstyp).
II. für alle λ-Regeln gilt: Falls λ-Regeln existieren, dann nur $S \to \lambda$. Falls $S \to \lambda \in P$, dann kommt S auf keiner rechten Seite einer Regel vor.

Beispiel 57 Sei $G = (N, T, P, S)$ mit $N = \{S, S', A, B\}$ und $T = \{a, b, c\}$ und

$$
\begin{aligned}
P = \{S &\to \lambda, && (10) & Ac &\to Bbcc, && (15)\\
S &\to S', && (11) & bB &\to Bb, && (16)\\
S' &\to abc, && (12) & aB &\to aa, && (17)\\
S' &\to aAbc, && (13) & aB &\to aaA\}. && (18)\\
Ab &\to bA, && (14) & & &&
\end{aligned}
$$

Beispiele für Ableitungen:

- $S \overset{(10)}{\Longrightarrow} \lambda$.
- $S \overset{(11)}{\Longrightarrow} S' \overset{(12)}{\Longrightarrow} abc$.
- $S \overset{(11)}{\Longrightarrow} S' \overset{(13)}{\Longrightarrow} aAbc \overset{(14)}{\Longrightarrow} abAc \overset{(15)}{\Longrightarrow} abBbcc \overset{(16)}{\Longrightarrow} aBbbcc \overset{(17)}{\Longrightarrow} aabbcc \in T^*$.
- $S \overset{(11)}{\Longrightarrow} S' \overset{(13)}{\Longrightarrow} aAbc \overset{(14)}{\Longrightarrow} abAc \overset{(15)}{\Longrightarrow} abBbcc \overset{(16)}{\Longrightarrow} aBbbcc \overset{(18)}{\Longrightarrow} aaAbbcc \notin T^*$.

Satz 11 $L(G) = \{a^n b^n c^n \mid n \in \mathbb{N}\}$.

Beweis: Zu zeigen ist $L = \{a^n b^n c^n \mid n \in \mathbb{N}\} \subseteq L(G)$. Beweis durch vollständige Induktion über n:

- Induktionsanfang: für $n = 0$ gilt $\lambda = a^0 b^0 c^0 \in L(G)$, für $n = 1$ gilt $S \Rightarrow^* a^1 b^1 c^1 \in L(G)$ und $S \Rightarrow^* a^1 A b^1 c^1 \in L(G)$ und für $n = 2$ gilt $S \Rightarrow^* a^2 b^2 c^2 \in L(G)$ und $S \Rightarrow^* a^2 A b^2 c^2$.
- Induktionsvoraussetzung: Für $n = k$ gilt: $S \Rightarrow^* a^k b^k c^k$ und $S \Rightarrow^* a^k A b^k c^k$.
- Induktionsbehauptung: Für $n = k + 1$ gilt: $S \Rightarrow^* a^{k+1} b^{k+1} c^{k+1}$ und $S \Rightarrow^* a^{k+1} A b^{k+1} c^{k+1}$.

Außerdem ist zu zeigen, dass $L(G) \subseteq \{a^n b^n c^n \mid n \in \mathbb{N}\}$. Der Beweis wird hier nicht geführt.

Bemerkung 61 $\{a^n b^n c^n \mid n \in \mathbb{N}\} \in \mathrm{CH}(1) \setminus \mathcal{REG}$.

Bemerkung 62 $\mathrm{CH}(1) = \{L \mid L = L(G)\}$ für eine Grammatik G vom Typ 1. Klar ist $\mathrm{CH}(1) \subseteq \mathrm{CH}(0)$.

Bemerkung 63 Erinnerung Typ-0-Normalform-Grammatik (Definition 29):

I. für alle Nicht-λ-Regeln gilt:
   1. $X \to YZ$,
   2. $XY \to Z$ oder
   3. $X \to a$,

   mit $X, Y, Z \in N$ und $a \in T$.

II. für alle λ-Regeln gilt: Es gibt höchstens eine λ-Regel.

Definition 31 Eine Typ-1-Grammatik $G = (N, T, P, S)$ ist eine *Typ-1-Normalform-Grammatik* genau dann, wenn

I. für alle Nicht-λ-Regeln gilt:

1. $X \to YZ$,
2. $XY \to UV$ oder
3. $X \to a$,

mit $U, V, X, Y, Z \in N$ und $a \in T$.

II. für alle λ-Regeln gilt: Es gibt höchstens die Regel $S \to \lambda$. Falls $S \to \lambda \in P$, dann kommt S auf keiner rechten Seite einer Regel vor.

Satz 12 Zu jeder Typ-1-Grammatik $G = (N, T, P, S)$ gibt es eine äquivalente Grammatik G' in Typ-1-Normalform.

Beweis in Globalschritten:

1. Schritt genauso wie bei Typ 0: $T' = \{t' \mid t \in T\}$, $N_1 = N \cup T'$ und $P_1 = P' \cup \{t' \to t \mid t \in T\}$. Im Ergebnis ist $G_1 = (N_1, T, P_1, S) \sim G$.
2. Schritt: Hier ist nichts mehr zu tun. $G_2 = G_1$.
3. Schritt: Ziel hier ist, für Regeln der Form $X_1X_2...X_n \to Y_1Y_2...Y_m$ mit $2 \leq n \leq m$ (Reduzierung der linken Seite auf die Länge 2). Dazu benutzen wir neue Nichtterminale $C_1, C_2, ..., C_{n-1}$ und ersetzen im
   1. Teilschritt: $X_1X_2 \to Y_1C_1$ (gewünschte Form) und $C_1X_3...X_n \to Y_2...Y_m$ (ursprüngliche Form mit der Länge $n-1$ und $m-1$).
   2. Teilschritt: $C_1X_3 \to Y_2C_2$ (gewünschte Form) und $C_2X_4...X_n \to Y_3...Y_m$.

   $\vdots$

   $n-2$. Teilschritt: $C_{n-2}X_n \to Y_{n-1}C_{n-1}$ und $C_{n-1} \to Y_n...Y_m$

   Warum der letzte Teilschritt mit C_{n-1}? Nach dem 3. Schritt haben alle Regeln entweder die Form
   - $XY \to UV$,
   - $X \to w$ mit $w \in N^+$ oder
   - $X \to a$.

   Zu tun bleibt $|w| > 2$ und $|w| = 1$.
4. Schritt: Ziel hier ist, Regeln mit „langer“ rechter Seite zu reduzieren auf Länge 2. Solche Regeln haben die Form $X \to Y_1Y_2...Y_m$ (mit linker Seite Länge 1). Das können wir behandeln wie im alten Schritt 3 bei Typ 0. Im Ergebnis ist $G_4 \sim G$ und alle Nicht-λ-Regeln haben die Form
   - $X \to YZ$,
   - $XY \to UV$,
   - $X \to Y$ oder
   - $X \to a$.
5. Schritt: Eliminieren der Regeln der Form $X \to Y$. Bei Typ 0 ging das ohne Probleme als $X \to UV$, $UV \to Y$. Hier ist dies nicht problemlos.

Wir definieren eine Hilfsmenge $A = \{Z \in N_4 \mid X \Rightarrow^* Z\}$ und berechnen sie wie folgt:

$$A_0 = \{X\} = \{X \Rightarrow^0 X\}$$
$$A_{i+1} = A_i \cup \{Z \in N_4 \mid \exists V \in A_i : V \to Z \in P_4\}$$

Stets ist $A_i \subseteq N_4$, also wird der Prozess stationär, d. h. $A_i = A_{i+1} = \ldots = A$. Wir haben zwei Teilschritte:

1. Falls die Regel $X \to Y$ pur angewendet wird, dann ersetze sämtliche Ableitungen $X \Rightarrow Y \Rightarrow Z_1 \Rightarrow \ldots \Rightarrow Z_k \Rightarrow w$ (w ist Nichtterminal) durch $X \to w$.
   - ⋆ Falls $w \in T$ oder $w = UV$, dann sind wir hier fertig.
   - ⋆ Falls $|w| > 2$, dann wird Schritt 4 ausgeführt.
2. Falls die Regel $X \to Y$ im Kontext $uXv \Rightarrow uYv$ mit $uv \in N^+$ angewendet wird, brauchen wir $UX \to UY$ und $XV \to YV$ für alle $V \in N_4$.

Im Ergebnis von Schritt 5 ist G_5 in Typ-1-Normalform.

Beispiel 58 $X_1X_2X_3X_4X_5 \to Y_1Y_2Y_3Y_4Y_5$ nach dem 3. Schritt:

1. $X_1X_2 \to Y_1C_1$ und $C_1X_3X_4 \to Y_2Y_3Y_4Y_5$
2. $C_1X_3 \to Y_2C_2$ und $C_2X_4 \to Y_3C_3$
3. $C_3 \to Y_4Y_5$

3.3 Kontextsensitive Grammatiken

Definition 32 Eine formale Grammatik $G = (N, T, P, S)$ ist *kontextsensitiv* genau dann, wenn

I. für alle Nicht-λ-Regeln gilt: Alle Regeln haben die Form $uAv \Rightarrow uwv$, wobei $A \in N$, $u, v \in (N \cup T)^*$ und $w \in (N \cup T)^+$.

II. für alle λ-Regeln gilt: Es gibt höchstens die Regel $S \to \lambda$. Falls $S \to \lambda \in P$, dann kommt S auf keiner rechten Seite einer Regel vor.

Bemerkung 64 „Kontextsensitiv“ bedeutet, A wird ersetzt durch $w \neq \lambda$ genau dann, wenn der linke Kontext u und der rechte Kontext v stimmt.

Bemerkung 65 Klar ist: Jede kontextsensitive Grammatik ist nicht verkürzend (vom Typ 1).

Bemerkung 66

$$\mathcal{CSL} = \{L \mid L = L(G) \text{ für eine kontextsensitive Grammatik } G\}$$

ist die Klasse der kontextsensitiven Sprachen. Klar ist, $\mathcal{CSL} \subseteq \mathrm{CH}(1)$.

Bemerkung 67 Die Grammatik, die $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ erzeugt, ist nicht kontextsensitiv. Dort haben wir Regeln $Ab \to bA$ und $bB \to Bb$ (Tauschregeln).

Vermutung: $\mathcal{CSL} \subsetneq \text{CH}(1)$.

Beispiel 59 $G = (N, T, P, S_0), N = (S_0, S, A, B, A', B'), T = \{a, b\}$ und

$$
\begin{aligned}
P = \{S_0 &\to \lambda, && (19) \qquad & A'A &\to BA, && (25)\\
S_0 &\to S, && (20) & BA &\to B'A, && (26)\\
S &\to AB, && (21) & B'A &\to B'B, && (27)\\
S &\to ABS, && (22) & B'B &\to AB, && (28)\\
AB &\to A'B, && (23) & A &\to a, && (29)\\
A'B &\to A'A, && (24) & B &\to b\} && (30)
\end{aligned}
$$

Beispiele für Ableitungen:

- $S_0 \xRightarrow{(19)} \lambda$
- $S_0 \xRightarrow{(20)} S \xRightarrow{(21)} AB \xRightarrow{(29)} aB \xRightarrow{(30)} ab$
- $S_0 \xRightarrow{(20)} S \xRightarrow{(21)} AB \xRightarrow{(23)} A'B \xRightarrow{(24)} A'A \xRightarrow{(25)} BA \xRightarrow{(30)} bA \xRightarrow{(29)} ba$
- $S_0 \xRightarrow{(20)} S \xRightarrow{(22)} ABS \xRightarrow{(21)} ABAB \xRightarrow{(26)} AB'AB \xRightarrow{(27)} AB'BB \xRightarrow{(28)} AABB \xRightarrow{(29)} aABB \xRightarrow{(29)} aaBB \xRightarrow{(30)} aabB \xRightarrow{(30)} aabb$

$L(G) = L_{\text{2equal}} = \{w \in \{a, b\}^* \mid |w|_a = |w|_b\}$. Es gilt $L_{\text{2equal}} \in \mathcal{CSL} \setminus \mathcal{REG}$

Satz 13 Zu jeder Typ-1-Grammatik G gibt es eine äquivalente kontextsensitive Grammatik G'.

Beweis: Sei $G = (N, T, P, S)$ eine Typ-1-Grammatik.

1. Schritt: Es gibt eine zu G äquivalente Normalform-Grammatik vom Typ 1: $G'' = (N'', T, P'', S)$ mit $G'' \sim G$.
   I. für alle Nicht-λ-Regeln gilt:
      1. $X \to YZ$ (das ist kontextsensitiv),
      2. $XY \to UV$ (das ist *nicht* kontextsensitiv) oder
      3. $X \to a$ (das ist kontextsensitiv),

      mit $U, V, X, Y, Z \in N$ und $a \in T$.

II. für alle λ-Regeln gilt: Es gibt höchstens die Regel $S \to \lambda$. Falls $S \to \lambda \in P$, dann kommt S auf keiner rechten Seite einer Regel vor.

2. Wir definieren für jede Regel der Form 2 ein neues Nichtterminal Z und ersetzen diese Regel durch: $XY \to ZY, ZY \to ZV, ZV \to UV$[12]

Also ist $\mathcal{CSL} = \text{CH}(1)$. □

Bemerkung 68 Alle Programmiersprachen (Java, Python, Modula, R, ...) werden durch kontextsensitive Grammatiken erzeugt.

Bemerkung 69 Das Wortproblem (gegeben einer Grammatik G vom Typ 1 und $w \in T^*$, gilt $S \Rightarrow^* w$?) ist für Typ-1-Grammatiken lösbar.

Begründung: Wenn $S \Rightarrow^* w$, dann gibt es eine Ableitungsfolge $S \Rightarrow x_1 \Rightarrow x_2 \Rightarrow \dots \Rightarrow x_k$ mit $x_k = w$. Für alle diese x_i gilt $|x_i| \leq |w|$ und $x_i \in (N \cup T)^*$. $X = \{x_i \in (N \cup T)^* \mid |x_i| \leq |w|\}$ (X ist endlich).

Wir betrachten nun alle Folgen der Form $x_{i_1}, x_{i_2}, \dots, x_{i_l}$ von Wörtern aus X mit der Eigenschaft $x_{i_\nu} \neq x_{i_\mu}$ für $\nu \neq \mu$. F ist die Menge aller dieser Folgen der Länge 1 bis Länge $\text{card}(X)$ (F ist endlich).

1. Schritt: Erzeuge X.
2. Schritt: Erzeuge F.
3. Schritt: Prüfe für jede Folge $x_{i_1}, x_{i_2}, \dots, x_{i_l}$, ob sie eine Ableitung für w ist.
   – Falls ja: $w \in L(G)$.
   – Falls nein: $w \notin L(G)$.

Beispiel 60 $G = (\{S\}, \{a, b\}, \{S \to ab, S \to aSb, S\}$. Wir sehen $L(G) = \{a^n b^n \mid n \in \mathbb{N}^+\}$. $ab \in L(G)$, da $S \Rightarrow ab$.

Das algorithmische Verfahren aus Bemerkung 69 liefert für $|w| = |ab| = 2$:

1. Schritt: $X = \{\lambda, S, a, B, SS, Sa, Sb, aS, aa, ab, bS, ba, bb\}$ mit $|X| = 13$.
2. Schritt: F wird erzeugt mit der Kardinalität:
   – Folgen der Länge 1: 13
   – Folgen der Länge 2: $13 \cdot 12$
   – Folgen der Länge k: k-Permutation von 13, also $P(13, k)$

   Also insgesamt: $\text{card}(F) = \sum_{k=1}^{13} P(13, k) = \sum_{k=1}^{13} \frac{13!}{(13-k)!} \approx 17 \cdot 10^9$.
3. Findet der Algorithmus die Folge S, ab und prüft $S \Rightarrow^* ab$.

[12] Kommentar des Dozenten: „Z wird man nur los, wenn der gesamte Zyklus angewendet wird.“

Bemerkung 70 Das algorithmische Verfahren aus Bemerkung 69 ist (und bleibt) unpraktikabel.

Bemerkung 71 Für Grammatiken, die Programmiersprachen erzeugen, sind die „kontextsensitiven Anteile“ relativ gering.

Als Beispiel für einen kontextsensitiven Anteil: Bevor ein Bezeichner (Variable) verwendet wird, muss sein Typ deklariert werden. Das bedeutet, es findet im linken Kontext statt.

3.4 Kontextfreie Grammatiken

Definition 33 Eine formale Grammatik $G = (N, T, P, S)$ ist eine *Typ-2-Grammatik* bzw. eine *kontextfreie Grammatik* genau dann, wenn alle Regeln von der Form $A \to w$ mit $A \in N$ und $w \in (N \cup T)^*$ sind[13].

$\mathrm{CH}(2) = \mathcal{CFL}$ ist die Klasse der von kontextfreien Grammatiken erzeugten Sprachen.

Beispiel 61 $G = (\{S\}, \{a, b\}, \{S \to \lambda, S \to aSb\}, S)$. Wir sehen $L(G) = \{a^n b^n \mid n \in \mathbb{N}\}$. Also ist $\{a^n b^n \mid \mathbb{N}^+\} \in \mathcal{CFL} \setminus \mathcal{REG}$.

Klar ist: $\mathrm{CH}(2) \subseteq \mathrm{CH}(0)$. Aber G ist keine Typ-1-Grammatik (Zusatz in II verletzt).

Bemerkung 72 Frage: Ist $\mathrm{CH}(2) \subseteq \mathrm{CH}(1)$? Aber $L(G) \in \mathrm{CH}(1) = \mathcal{CSL}$. Problem: Kontextfreie Grammatiken können viele λ-Regeln besitzen.

Beispiel 62 Wir betrachten eine Grammatik mit folgenden Regeln:

$$\begin{array}{ll} A \to BC & D \to \lambda \\ B \to DE & E \to \lambda \\ C \to DF & F \to \lambda \end{array}$$

Wir können λ ganz oft ableiten: $D \Rightarrow^* \lambda$, $E \Rightarrow^* \lambda$, $F \Rightarrow^* \lambda$, $C \Rightarrow^* \lambda$, $B \Rightarrow^* \lambda$, $A \Rightarrow^* \lambda$.

Beispiel 63 Wir betrachten die Sprache $L_{\mathrm{pal}} = \{w \in \{a, b\}^* \mid \mathrm{Sp}(w) = w\}$ der Palindrome und dazu die folgende Grammatik: $G = (\{S\}, \{a, b\}, \{S \to aSa, S \to bSb, S \to a, S \to b, S \to \lambda\}, S)$. $L_{\mathrm{pal}} \in \mathcal{CFL} \setminus \mathcal{REG}$.

[13] $u = \lambda = v$ im Vergleich zu $uAv \to uwv$ mit $w \in (N \cup T)^+$.

Wir betrachten die Klammersprache (Dyck-Sprache), die von der folgenden Grammatik erzeugt wird: $G' = (\{S\}, \{(,)\}, \{S \to (S), S \to SS, S \to \lambda\}, S)$.

Aufgabe: Es ist eine Grammatik zu konstruieren, die korrekt geklammerte Terme mit 2 bzw. 3 Klammern beschreibt.

Satz 14 Zu jeder kontextfreien Grammatik $G = (N, T, P, S)$ gibt es eine nicht verkürzende kontextfreie Grammatik G' mit $L(G') = L(G) \setminus \lambda$.

Bemerkung 73 Folgerung aus Satz 14: Zu jeder kontextfreien Grammatik $G = (N, T, P, S)$ gibt es eine äquivalente kontextfreie Grammatik G'', die Typ-1-Grammatik ist.

Begründung: Wir erweitern G' um ein neues Startsymbol S_0 und die Regeln $S_0 \to \lambda$ und $S_0 \to S$.

Bemerkung 74 Mit der Folgerung gilt $\mathrm{CH}(2) \subseteq \mathrm{CH}(1) \subseteq \mathrm{CH}(0)$ bzw. $\mathcal{CFL} \subseteq \mathcal{CSL}$.

Definition 34 Eine kontextfreie Grammatik $G = (N, T, P, S)$ ist eine *Typ-2-Normalform-Grammatik* (kontextfreie Grammatik in Normalform) genau dann, wenn

I. Für alle Nicht-λ-Regeln gilt:
   1. $X \to YZ$
   2. (wird nicht benötigt)
   3. $X \to a$

   mit $X, Y, Z \in N$ und $a \in T$.
II. Für alle λ-Regeln gilt: Es gibt höchstens die Regel $S \to \lambda$. Falls $S \to \lambda \in P$, dann kommt S auf keiner rechten Seite einer Regel vor.

Bemerkung 75 Jede Typ-2-Normalform-Grammatik ist zugleich eine Typ-1-Normalform-Grammatik bzw. Typ-0-Normalform-Grammatik.

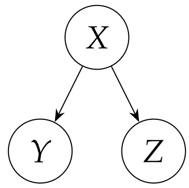
Satz 15 Zu jeder kontextfreien Grammatik $G = (N, T, P, S)$ existiert eine äquivalente Typ-2-Normalform-Grammatik.

Beweis: Sei $L \in \mathcal{CFL}$. Dann gibt es eine Typ-2-Normalform-Grammatik $G = (N, T, P, S)$, die L erzeugt (Satz 15).

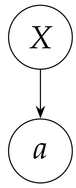
Beobachtung: Wir betrachten für ein $z \in L$ einen Ableitungsbaum in G. Für den gilt:

- Die Wurzel trägt die Marke S.
- Die Blätter (insgesamt $|z|$ viele) sind mit Terminalen $z_1, z_2, \ldots$ markiert.

Für alle inneren Knoten gilt: Entweder hat ein Knoten 2 Söhne (die Form 1)

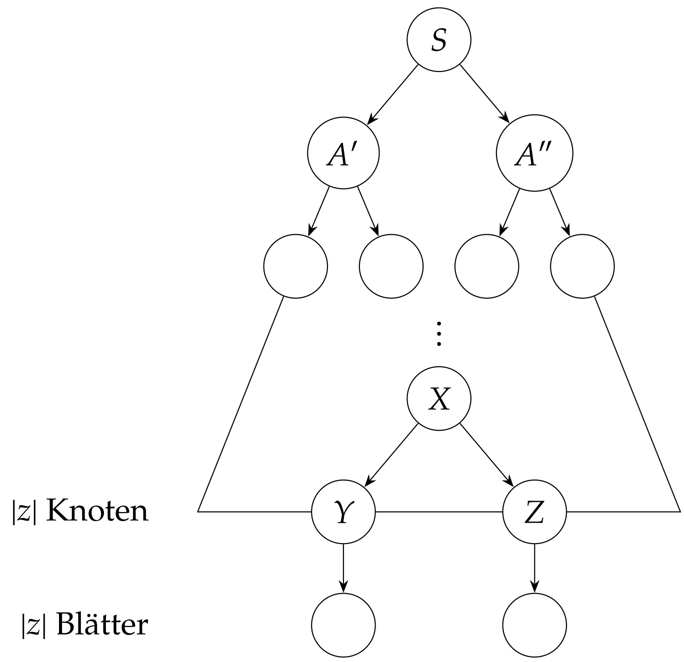


oder 1 Sohn (die Form 3).



Wir wissen, es gibt $|z|$ viele derartige innere Knoten. Der Ableitungsbaum hat damit die in Abbildung 11 dargestellte Struktur.

Abbildung 11 Struktur des Ableitungsbaums für z



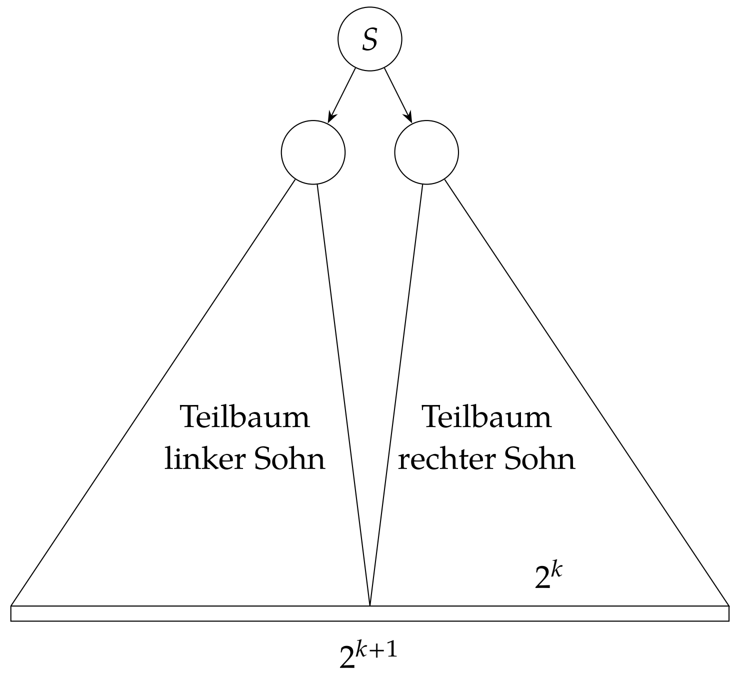
Jeder Ableitungsbaum ist ein binärer Baum.

Lemma 3 Kombinatorisches Lemma für binäre Bäume: Jeder binäre Baum mit mindestens 2^m Blättern besitzt einen Pfad der Länge m (d. h. hat eine Höhe von mindestens m).

Beweis durch vollständige Induktion über m:

- Induktionsanfang mit $m = 0$: $2^0 = 1$ Blatt (trivialer Graph) mit der Höhe 0.
- Induktionsvoraussetzung mit $m = k$: Jeder binäre Baum mit mindestens 2^k Blättern besitzt einen Pfad der Länge k.
- Induktionsbehauptung mit $m = k + 1$: Jeder binäre Baum mit mindestens 2^{k+1} Blättern besitzt einen Pfad der Länge $k + 1$.

Abbildung 12 Darstellung der Blätter eines binären Baumes



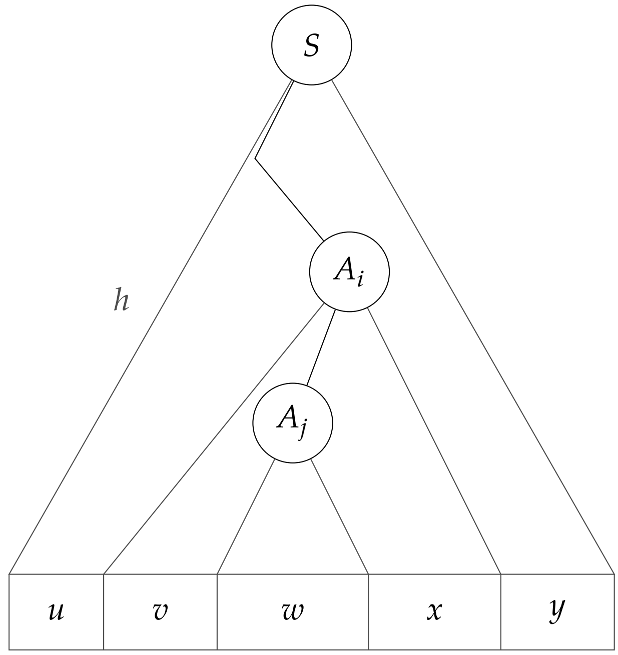
Induktionsbeweis: Wir betrachten Abbildung 12. Einer der Teilbäume hat mindestens 2^k Blätter. Dieser Teilbaum besitzt nach Induktionsvoraussetzung einen Pfad mindestens der Länge k. Folglich hat der Ausgangsbaum eine Pfad der Länge $k + 1$. □

Sei $n = |N|$. Wir setzen $n_L = 2^n$ und betrachten ein Wort z der Länge $|z| \geq n_L$. Für den Ableitungsbaum für z gilt: Die Höhe ohne Blätter (für innere Knoten) $h \geq \log_2(|z|) \geq \log_2(n_L) = \log_2(2^n) = n$. Mit den Blättern gilt $h + 1 \geq n + 1$.

Wir betrachten einen solchen längsten Pfad: Der hat die Knoten $v_0, v_1, v_2, \ldots, v_{n+1}$ (Länge $n+1$) und die Markierungen $M(v_0) = S, M(v_i) \in N$, $M(v_{n+1}) \in T$. Wir haben also $h+1 \geq n+1$ Markierungen von v_0 bis v_n mit n verschiedenen Nichtterminalen. $M(v_i) = A_i$ für $0 \leq i \leq n$. Also gibt es Paare (i, j) mit $i \neq j$ mit $A_i = A_j$.

Wir betrachten die erste derartige Dopplung von unten. Das heißt, in dem „Abschnitt“ $A_{i+1}, A_{i+2}, \ldots, A_j, A_{j+1}, \ldots, A_n$ gibt es keine Dopplung, d. h. hier stehen höchstens n verschiedene Nichtterminale.

Abbildung 13 Teilbäume im Ableitungsbaums für z



Wir betrachten Abbildung 13:

- Der Teilbaum mit der Wurzel A_j erzeugt das Teilwort w.
- Der Teilbaum mit der Wurzel A_i erzeugt das Teilwort vwx.
- Der Baum mit der Wurzel $A_0 = S$ erzeugt das Wort $z = uvwxy$.

Für diese Zerlegung gilt: Die Höhe des Teilbaums mti der Wurzel A_i ist durch n beschränkt. Das heißt, die Länge $|vwx|$ ist beschränkt durch $2^n = n_L$. Das ist 1.

Sei $A_i \to A'A''$. Ohne Beschränkung der Allgemeinheit sei A_j Nachfolger von A'. Das nichtleere Teilwort x'', das aus A'' abgeleitet wird, ist ein Suffix von x. Also gilt $0 < |x''| \leq |x| \leq |vx|$. Das ist 2.

Wir brauchen noch 3:

- für $i = 0$: Wir „kleben“ den Teilbaum mit Wurzel A_j bei A_i ein und haben eine Ableitung für $z_0 = uwy$.
- für $i = 2$: Wir „kleben“ den Teilbaum mit Wurzel A_i bei A_j ein und haben eine Ableitung für $uvvwxxy$.
- usw. □

Satz 16 *Pumping-Lemma für kontextfreie Sprachen*: Wenn L eine kontextfreie Sprache ist, dann existiert eine natürliche Zahl $n_L \geq 1$ (Pumping-Zahl), sodass für alle Wörter $z \in L$ mit $|z| \geq n_L$ eine Zerlegung $z = uvwxy$ mit folgenden Eigenschaften existiert:

1. $|vwx| \leq n_L$,
2. $|vx| \geq 1$ ($vx \neq \lambda$) und
3. für alle $i \in \mathbb{N}$ gilt: $z_i = uv^iwx^iy \in L$.

Bemerkung 76 Wenn für jede natürliche Zahl $n_L \geq 1$ ein Wort $z \in L$ mit $|z| \geq n_L$ existiert, sodass für alle Zerlegungen $z = uvwxy$ mit

1. $|vwx| \leq n_L$ und
2. $|vx| \geq 1$

ein $i \in \mathbb{N}$ existiert mit $z_i = uv^iwx^iy \notin L$, dann ist L nicht kontextfrei.

Beispiel 64 $L_3 = \{a^nb^nc^n \mid n \in \mathbb{N}\}$. Wir wissen $L_3 \in \mathcal{CSL}$. Behauptung: $L_3 \notin \mathcal{CFL}$.

Angenommen $L_3 \in \mathcal{CFL}$. Dann bekommen wir eine Pumping-Zahl n_{L_3}. Wir setzen $z = a^{n_{L_3}}b^{n_{L_3}}c^{n_{L_3}}$. Für jede Zerlegung $z = uvwxy$ mit 1 und 2 finden wir ein $i \in \mathbb{N}$, sodass $z_i \notin L$.

<table>
<tr><td colspan="3">$a^{n_{L_3}}$</td><td colspan="2">$b^{n_{L_3}}$</td><td>$c^{n_{L_3}}$</td></tr>
<tr><td>u</td><td>v</td><td>w</td><td>x</td><td colspan="2">y</td></tr>
</table>

Da wegen 1 $|vwx| \leq n_{L_3}$ ist, „überlappt“ das Teilwort vwx *nicht* alle 3 Abschnitte von z, d. h. vwx enthält nicht alle 3 Terminale a, b und c zugleich. Ohne Beschränkung der Allgemeinheit sei c der Buchstabe, der nicht vorkommt.

Da wegen 2 $|vx| \geq 1$, kommt in vx mindestens ein Buchstabe vor. Ohne Beschränkung der Allgemeinheit sei a ein Buchstabe, der in v vorkommt.

Dann gilt für $z_0 = uwy$: $|uwy|_a \leq |uvwxy|_a = |uvwxy|_c = |v|_c$. Das heißt, $z_0 \notin L_3$. □

Beispiel 65 Dagegen ist $L_2 = \{a^nb^n \mid n \in \mathbb{N}\}$ typisch kontextfrei.

Beispiel 66 Eine andere Sprache, die typisch kontextfrei ist, ist $L_{\text{pal}} = \{w\text{Sp}(w) \mid w \in \{a,b\}^*\}$. Frage: Was ist mit $L_{\text{dup}} = \{ww \mid w \in \{a,b\}^*\}$. Behauptung: $L_{\text{dup}} \notin \mathcal{CFL}$.

Indirekter Beweis: Angenommen $L_{\text{dup}} \in \mathcal{CFL}$. Dann bekommen wir eine Pumping-Zahl n_L. Wir setzen $z = a^{n_L} b^{n_L} a^{n_L} b^{n_L}$. Für jede Zerlegung $z = uvwxy$ mit 1 und 2 finden wir ein $i \in \mathbb{N}$ mit $z_i \notin L_{\text{dup}}$.

<table>
<tr><td colspan="3">a^{n_L}</td><td colspan="2">b^{n_L}</td><td colspan="2">a^{n_L}</td><td>b^{n_L}</td></tr>
<tr><td colspan="2">u</td><td>v</td><td>w</td><td>x</td><td colspan="3">y</td></tr>
</table>

Da wegen 1 $|vwx| \leq n_L$, überlappt das Teilwort vwx nicht zugleich 3 der 4 Abschnitte. Das heißt, entweder vwx liegt in einem der 4 Abschnitte oder über der Grenze zwischen zweien. Wegen 2 enthält vx Buchstaben a oder b oder beide.

1. Fall: $vx = a^k$. Dann gilt für $z_0 = a^{n_L-k} b^{n_L} a^{n_L} b^{n_L}$ oder $z_0 = a^{n_L} b^{n_L} a^{n_L-k} b^{n_L}$ in jedem Fall $z_0 \notin L_{\text{dup}}$.
2. Fall: $vx = b^j$. Das folgt analog zum Fall 1.
3. Fall: $vx = b^j a^k$. Dann gilt $z_0 = a^{n_L} b^{n_L-j} a^{n_L-k} b^{n_L}$. Damit ist $z_0 \notin L_{\text{dup}}$.
4. Fall: $vx = a^k b^j$. Dann gilt $z_0 = a^{n_L-k} b^{n_L-j} a^{n_L} b^{n_L}$ oder $z_0 = a^{n_L} b^{n_L} a^{n_L-k} b^{n_L-j}$. In beiden Fällen ist $z_0 \notin L_{\text{dup}}$.

Die Fallunterscheidung ist vollständig. $\square$

Beispiel 67 $L_{\text{quad}} = \{a^{n^2} \mid n \in \mathbb{N}\}$. Wir wissen, $L_{\text{quad}} \notin \mathcal{REG}$. Frage: Ist L_{quad} kontextfrei? Behauptung: $L_{\text{quad}} \in \mathcal{CFL}$.

Indirekter Beweis: Angenommen L_{quad} ist kontextfrei. Dann gibt es eine Pumping-Zahl n_q, sodass für alle Wörter $z \in L_{\text{quad}}$ mit $|z| \geq n_q$ eine Zerlegung $z = uvwxy$ mit 1, 2, 3 existiert. Wir setzen $z = a^{n_q^2}$. 1 heißt $|vwx| \leq n_q$ und 2 heißt $|vx| = k \geq 1$.

Dann gilt für $i = 2$: $z_2 = uv^2wx^2y$ mit $|z_2| = |uv^2wx^2y| = n_q^2 + k \leq n_q^2 + n_q < n_q^2 + 2n_q + 1 = (n_q+1)^2$. Folglich ist $z_2 \notin L_{\text{quad}}$. $\square$

Bemerkung 77 Für einbuchstabige Sprachen fallen die Strukturen der Pumping-Lemmata für $\mathcal{REG}$ und $\mathcal{CFL}$ zusammen:

$$uv^iwx^iy \text{ (kontextfrei)} = uwv^ix^iy = uv^iw \text{ (regulär)}$$

Tatsächlich gilt für $\Sigma = \{a\}$ $\mathcal{REG}_\Sigma = \mathcal{CFL}_\Sigma$. Sonst gilt $\mathcal{REG} \subsetneq \mathcal{CFL}$.

Bemerkung 78 Sei $L \in \mathcal{CFL}$ und $G = (N, T, P, S)$ eine Typ-2-Normalform-Grammatik, die L erzeugt, d. h. $L(G) = L$. Es gilt:

I. Für alle Nicht-λ-Regeln gilt:

1. $X \to YZ$
2. (wird nicht benötigt)
3. $X \to a$

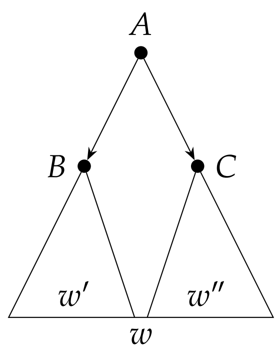
II. Für alle λ-Regeln gilt: Es gibt höchstens die Regel $S \to \lambda$. Falls $S \to \lambda \in P$, dann kommt S auf keiner rechten Seite einer Regel vor.

Damit gilt $\lambda \in L(G) \iff S \to \lambda \in P$. Im Folgenden betrachten wir nur nichtleere Wörter. Sei $u = u_1 u_2 ... u_n \in T^*$. Frage: $S \Rightarrow^* u$?

Hilfsmengen: $M_{i,j}(u) = \{A \in N \mid A \Rightarrow^* u_{i+1}...u_j\}$ mit $0 \leq i < j \leq n$. Damit gilt $S \Rightarrow^* u$ genau dann, wenn $S \in M_{0,n}$. Wie lassen sich diese Mengen berechnen?

$$M_{i,i+1}(u) = \{A \in N \mid A \to u_{i+1} \in P\} \tag{31}$$

Dann existiert eine Regel der Form 1 $A \to BC$ und $B \Rightarrow^* w'$ und $C \Rightarrow^* w''$ und $w'w'' = w = u_{i+1}...u_{i+d}$. Der Ableitungsbaum sieht folgendermaßen aus:



$$\begin{aligned} M_{i,i+d} = \{A \in N \mid{} & \text{es existiert ein } k \text{ mit } i < k < i+d, \text{ sodass gilt:} \\ & \text{es gibt eine Regel } A \to BC \text{ und} \\ & B \Rightarrow^* u_{i+1}...u_{i+k}, \text{ d. h. } B \in M_{i,k}(u), \\ & \text{sowie } C \Rightarrow^* u_{i+k+1}...u_{i+d}, \text{ d. h. } C \in M_{k,i+d}(u)\} \end{aligned} \tag{32}$$

Für Gleichung (31):

```
for i = 0 to u − 1 do
    M_{i,i+1} = ∅
    for all A ∈ N do
        if (A → u_{i+1}) ∈ P then
            M_{i,i+1} = M_{i,i+1} ∪ {A}
        end if
    end for
```

```
    end for
```

- Berechnung einer solchen Menge: konstanter Aufwand
- $\mathcal{O}(n)$ solche Mengen

Für Gleichung (32):

```
for d = 2 to n do
    for i = 0 to d − n do
        M_{i,i+d} = ∅
        for k = i + 1 to i + d − 1 do
            for all B ∈ N do
                for all C ∈ N do
                    if (A → BC) ∈ P and B ∈ M_{i,k}(u) and C ∈
                    M_{k,i+d}(u) then
                        M_{i,i+d} = M_{i,i+d} ∪ {A}
                    end if
                end for
            end for
        end for
    end for
end for
```

Dabei ist d die Wortlänge, i der Startindex und k ein Zwischenindex. Die beiden For-Each-Schleifen zur Abarbeitung aller B und C aus N haben konstanten Aufwand. Der Aufwand für Gleichung (32): $\mathcal{O}(d \cdot i \cdot k) = \mathcal{O}(n^3)$. Das ist praktikabel.

- Anzahl der Mengen $M_{i,i+d}$: $\mathcal{O}(n^2)$
- Aufwand für eine Menge: $\mathcal{O}(n)$

Satz 17 Das Wortproblem für kontextfreie Sprachen ist *effizient lösbar* und der Algorithmus von Cocke, Kasani und Younger liefert eine Realisierung in $\mathcal{O}(n^3)$.

Bemerkung 79 Alle Programmiersprachen sind „nahezu kontextfrei“. Die Deklarationen für Bezeichner werden als Kommentare beschrieben.

Konsequenz: Syntaxanalyse und Compiler werden mit Techniken realisiert, die auf kontextfreien Sprachen beruhen.

Beispiel 68 $L = \{u \, \$ \, v \mid u, v \in \{a,b\}^*, |u|_a = |v|_b\}$. $L \in \mathcal{CFL} \setminus \mathcal{REG}$. Beweis mit Pumping-Lemma.

Wir finden eine Grammatik $G = (\{D, L, R, A', B'\}, \{a, b, \$\}, P, D)$ mit

$$
\begin{aligned}
P = \{D &\to LDR, & A' &\to \lambda, \\
L &\to B'aB', & B' &\to B'b, \\
R &\to A'bA', & B' &\to \lambda\} \\
A' &\to A'a,
\end{aligned}
$$

Beispiel 69 $L = \{uv \mid u, v \in \{a, b\}^*, |u|_a = |v|_b\}$. $L \in \mathcal{REG}$, da $L = \{a, b\}^*$. Wir zeigen induktiv: jedes $w \in \{a, b\}^*$ besitzt eine Zerlegung $w = uv$ mit der Eigenschaft $|u|_a = |v|_b$.

- Induktionsanfang: Für $|w| = 0$ gilt $w = \lambda = \lambda\lambda = uv$ mit $|u|_a = |v|_b = 0$.
  Für $|w| = 1$: $w = a = \lambda a = uv$ mit $|u|_a = |v|_b = 0$ und $w = b = b\lambda = uv$ mit $|u|_a = |v|_b = 0$.
- Induktionsvoraussetzung: Für alle Wörter w der Länge n gibt es eine Zerlegung $w = uv$.
- Induktionsbehauptung: Für alle Wörter w der Länge $n + 1$ gibt es eine Zerlegung $w = uv$.

Induktionsschluss:

1. Fall: $w_{n+1} = a$: $w' = u'v'$ mit $u' = u$ und $v' = va$. Dann ist $|u'|_a = |u|_a = |v|_b = |v'|_b$.
2. Fall: $w_{n+1} = b$
   1. Fall: $v = \lambda$, d. h. $w = uv = u\lambda = u$ mit $|u|_a = |\lambda|_b = 0$, d. h. $w = b^n$, also ist $w' = b^{n+1}$ und $u' = w'$ und $v' = v = \lambda$.
   2. Fall: $v \neq \lambda$.
      1. $v = bv''$ (1. Buchstabe von v ist b). Dann ist $w = uv = ubv''$ und $w' = ubv''b$. Wir setzen $u' = ub$ und $v' = v''b$. Dann ist $|u'|_a = |u|_a$ und $|v'|_b = |v|_b = |v''|_b + 1$.
      2. $v = av''$ (1. Buchstabe von v ist a). Wir setzen $u' = ua$ und $v' = v''b$. Dann ist $|u'|_a = |u|_a + 1 = |v|_b + 1 = |v'|_b$.

Beispiel 70 Wir betrachten $L = \{\mathrm{Sp}(\mathrm{bin}(n)) \,\$\, \mathrm{bin}(n) \mid n \geq 1\}$. Sie ist kontextfrei, da wir eine kontextfreie Grammatik $G = (\{D', D\}, \{0, 1, \$\}, P = \{D' \to 1D1, D' \to 0D0, D \to 1D1, D \to 0D0, D \to 1\,\$\,1\}, D')$ dafür finden.

Beispiel 71 Es ist zu untersuchen, ob $L' = \{\mathrm{Sp}(\mathrm{bin}(n)) \,\$\, \mathrm{bin}(n+1) \mid n \geq 1\}$ kontextfrei ist. L ist kontextfrei, da wir eine kontextfreie Grammatik $G' = (\{S, A, B, C\}, \{0, 1\}, P, S)$ finden:

$$
\begin{aligned}
P = \{S &\to 0\,\$\,1, & S &\to 1B0, & A &\to 1A0, \\
S &\to 1A0, & S &\to 0C1, & A &\to \$1,
\end{aligned}
$$

$$B \to 1B0, \qquad C \to 0C0, \qquad C \to 1\,\$\,1\}$$
$$B \to 0C1, \qquad C \to 1C1,$$

Es gibt drei Fälle von Wörtern, die von G' generiert werden müssen, um die Sprache abzudecken:

- Das Wort $0\,\$\,1$ als einziges Wort der Länge 3.
- Wörter, für die die Binärrepresentation des Nachfolgers von n eine Stelle länger ist als die von n. Das betrifft alle Zahlen, deren Binärrepresentation nur aus r 1en besteht. Diese werden in der Expansionskette $S \Rightarrow 1A0 \Rightarrow^* 1^r\,\$\,10^r$ abgehandelt.
- Wörter, für die die Binärrepresentation des Nachfolgers von n genauso lang ist wie die von n. Dann ist $n_2 = 1\{0,1\}^r 01^i$. Hierbei ist die Grundidee, dass die Binärdarstellung des Inkrements einer solchen Zahl derart gestaltet ist, dass von rechts gelesen alle 1en invertiert werden und die erste 0 zur 1 gemacht. Alle links davon stehenden Stellen werden kopiert. Falls mindestens eine 1 vor der ersten 0 steht, wird in G' eine Ableitungskette $S \Rightarrow 1B0 \Rightarrow^* 1...1B0...0 \Rightarrow 1...10C10...0 \Rightarrow^* 1^i0\{0,1\}^r1\,\$\,1\{0,1\}^r10^i$. In G' wird der Fall $i = 0$ gesondert gehandelt (als $S \Rightarrow 0C1$).

So werden alle Wörter von L' durch G' erzeugt.

Beispiel 72 $L_3 = \{a^nb^nc^n \mid n \in \mathbb{N}\}$ ist nicht kontextfrei, aber kontextsensitiv. Es ist zu prüfen, in welcher Klasse $\overline{L_3}$ liegt.

$\overline{L_3} = \{a^ib^jc^k \mid i \neq j \vee j \neq k\} \cup \{w \in \{a,b,c\}^* \mid ba \sqsubseteq w \vee ca \sqsubseteq w \vee cb \sqsubseteq w\} = \{a^ib^jc^k \mid (i < j \vee j < i) \vee (j < k \vee k < j)\} \cup \{w \in \{a,b,c\}^* \mid ba \sqsubseteq w \vee ca \sqsubseteq w \vee cb \sqsubseteq w\} = T_1 \cup T_2 \cup ... \cup T_7$ mit

- $T_1 = \{a^ib^jc^k \mid 0 \leq i < j, k \in \mathbb{N}\}$ – kontextfrei (siehe unten)
- $T_2 = \{a^ib^jc^k \mid 0 \leq j < i, k \in \mathbb{N}\}$ – kontextfrei (analog zu T_1)
- $T_3 = \{a^ib^jc^k \mid 0 \leq j < k, i \in \mathbb{N}\}$ – kontextfrei (analog zu T_1)
- $T_4 = \{a^ib^jc^k \mid 0 \leq k < j, i \in \mathbb{N}\}$ – kontextfrei (analog zu T_1)
- $T_5 = \{\{a,b,d\}^* \mid ba \sqsubseteq w\}$ – regulär (damit kontextfrei)
- $T_6 = \{\{a,b,d\}^* \mid ca \sqsubseteq w\}$ – regulär (damit kontextfrei)
- $T_7 = \{\{a,b,d\}^* \mid cb \sqsubseteq w\}$ – regulär (damit kontextfrei)

$G_{T_1} = (N, \{a,b,c\}, P, S)$ mit

$$\begin{aligned} P = \{S &\to AC, & B &\to \lambda, \\ A &\to aAb, & C &\to Cc, \\ A &\to Bb, & C &\to \lambda\} \\ B &\to Bb, \end{aligned}$$

Beispiel 73 Wir betrachten $L_4 = \{a^n b^n c^n d^n \mid n \in \mathbb{N}\}$. Dafür formulieren wir die folgende Grammatik $G = (\{S, A, B, C, D, X, Y, Z, B', C'\}, \{a, b, c, d\}, P, S)$.

$$
\begin{aligned}
P = \{S &\to \lambda, & DC' &\to C'D,\\
S &\to ABCD, & B'B &\to BB,\\
S &\to AXZYD, & Z &\to BC,\\
XZY &\to XXZYY, & A &\to a,\\
X &\to AB', & B &\to b,\\
Y &\to C'D, & C &\to c,\\
B'A &\to AB', & D &\to d\}
\end{aligned}
$$

Das ist eine Typ-1-Grammatik, da jede Regel nicht verkürzend ist und wir nur eine λ-Regel haben, die $S \to \lambda$ ist und S auf keiner rechten Seite vorkommt. Damit liegt $L_4 \in \mathrm{CH}(1)$. Die Argumentation für die Nichtkontextfreiheit ist analog zur Argumentation für L_3.

Beispiel 74 Es ist zu zeigen, dass $L = \{\mathrm{bin}(n)\$\mathrm{bin}(n{+}1) \mid n \in \mathbb{N}^+\}$ nicht kontextfrei ist.

Angenommen, L ist kontextfrei. Dann ist p die Pumpingzahl für L. Für alle $z \in L$ mit $|z| \geq p$ existiert daher eine Zerlegung $z = uvwxy$ mit $|vwx| \leq p$, $|vx| \geq 1$ und $\forall i \in \mathbb{N} : z_i = uv^iwx^iy \in L$. Wir setzen $z = 1^p0^p\,\$\,1^p0^{p-1}1$.

Es ist offensichtlich, dass \$ in w, da sonst \$ als nicht pumpbares Symbol in u oder y läge (mit $wxy = \lambda$ bzw. $uvw = \lambda$), sodass durch einseitiges Pumpen die Bedingung 3 nicht erfüllt werden kann. Fallunterscheidung auf Basis von $|vwx| \leq p$:

- $0 \in vx$, dann $0 \in v$, dann ist für beliebige $i \neq 1$ die Anzahl der 0 größer als 1 und damit das rechte Teilwort nicht die Repräsentation des Nachfolgers, also ist $z_i \notin L$.
- $0 \notin vx$, dann ist $v = \lambda$, also $x = 1^r$ (wegen $|vx| \geq 1$). Damit wird für $i \neq 1$ die Anzahl der 1 im rechten Teilwort zu groß, sodass es nicht mehr der Nachfolger des linken Teilworts ist. Damit $z_i \notin L$.

Die Fallunterscheidung ist vollständig. Für z existiert keine pumpbare Zerlegung, damit ist L nicht kontextfrei. $\square$

Beispiel 75 Es ist die Suffix-Version des Pumping-Lemmas zu zeigen.

Sei L eine reguläre Sprache. Dann gibt es einen deterministischen endlichen Automaten $A = (Q, \Sigma, \delta, q_1, F)$, der L akzeptiert. Wir definieren $n_L = |Q|$.

Sei $z = z_1 z_2 ... z_l \in L(A)$ mit $l \geq n_L$. Dann gibt es eine akzeptierende Berechnung $B_A(z) = (K_0, K_1, ..., K_l)$ von A mit $K_0 = \text{Start-}K_A(z), K_{i-1} \vdash K_i$ für $i = 1, ..., l$ und K_l ist akzeptierende Finalkonfiguration.

Es sei weiter $(p_0, p_1, ..., p_l)$ die zugehörige Folge der Zustände. Für die Länge dieser Folge gilt $l + 1 \geq n_L + 1$. Also gibt es ein Paar (i, j), für das gilt $p_j = p_i$ und (i, j) ist das letzte Paar mit dieser Eigenschaft und $i < j$, das heißt $p_0, p_1, ..., p_{i-1}, p_i, \underbrace{p_{i+1}, ..., p_{j-1}, p_j, p_{j+1}, ..., p_l}_{\leq n_L}$.

Wir definieren $u = z_1 z_2 ... z_i$, $v = z_{i+1} ... z_{j-1} z_j$ und $w = z_{j+1} ... z_l$. Hierfür gilt:

1. $|vw| \leq n$, da $|z_{i+1} ... z_l| = |vw|$ und die Länge der Folge $(p_{i+1}, ..., p_l)$ durch n_L beschränkt.
2. $|v| \geq 1$, da $j \neq i$ und $i < j$.

Es bleibt 3. $uv^i w \in L$ für alle i zu zeigen.

Für $i = 0$ gilt $w \sim_A vw$ für $q = p_i = p_j$. Denn $x \sim_A y \iff (q_1, x) \vdash^* (q, \lambda) \wedge (q_1, y) \vdash^* (q, \lambda)$, d. h. es existiert $q \in Q$ mit $x \in [q]$ und $y \in [q]$.

$w \sim_A vw$ bedeutet: Für alle Wörter $y \in \Sigma^*$ gilt $yw \sim_A yvw$, insbesondere $yw \in L(A) \iff yvw \in L(A)$. Das heißt, für $y = u$, dass $uw \in L(A) \iff uvw \in L(A)$, also $z \in L(A) \implies uv^0 w \in L(A)$.

Weiter für $y = uv$: $uvw \in L(A) \iff uvvw \in L(A)$. Da $uvw = z \in L(A)$, ist $uv^2 w \in L(A)$. Analog für $y = uv^2$: $uv^2 w \in L(A) \iff uvv^2 w \in L(A)$. Da $uv^2 w \in L(A)$, gilt $uv^3 w \in L(A)$. □

Beispiel 76 Es ist eine nichtreguläre Sprache zu finden, deren Nichtregularität mit der Suffix-, aber nicht mit der Präfixversion gezeigt werden kann.

Wir untersuchen die Sprache $L = \{a^m b^n c^n \mid m, n \geq 1\} \cup \{b^m c^n \mid m, n \geq 0\}$. Mit dem Präfix-Pumping-Lemma erfüllt L alle Eigenschaften. Alle $z \in L$ lassen sich in uvw zerlegen, sodass für alle $i \geq 0$ $uv^i w \in L$, indem z. B. v als erster Buchstabe gewählt wird. Ist dieser ein a, kann er gepumpt werden (die Anzahl führender as ist beliebig). Ist er ein b oder c, fällt es in die zweite Menge ohne führende as. Dann ist aber die Anzahl von führenden bs oder cs beliebig, also sind die bs oder cs beliebig pumpbar.

In der Suffixversion kann die Nichtregularität gezeigt werden. Wäre L regulär, dann existierte eine Pumpingzahl p. Für alle $z \in L$ mit $|z| \geq p$ existiert daher eine Zerlegung uvw mit $|vw| \leq p$, $|v| \geq 1$ und $\forall i \in \mathbb{N} : z_i = uv^i w \in L$. Wir setzen $z = ab^p c^p$.

vw kann wegen $|vw| \leq p$ nur cs enthalten. Sei $v = c^r \neq \lambda$, also $r \geq 1$. Wir setzen $z_p = ab^p c^{rp} c^p$. Da $rp + p > p$, ist $z_p \notin L$. L ist nicht regulär. □

3.5 Grammatiken von Chomsky-Typ 3

Rechtslineare Grammatiken

Definition 35 Eine kontextfreie Grammatik $G = (N, T, P, S)$ ist eine *Typ-3-Grammatik* (rechtslineare Grammatik) genau dann, wenn alle Regeln von der Gestalt $A \to wB$ oder $A \to w$ mit $A, B \in N$ und $w \in T^*$ sind.

Definition 36 $\text{CH}(3) = \mathcal{RLIN}$ bezeichnet die Klasse der Sprachen, die von Typ-3-Grammatiken erzeugt werden.

Bemerkung 80 Klar ist $\text{CH}(3) \subseteq \text{CH}(2)$ (per Definition). Damit gilt $\text{CH}(3) \subsetneq \text{CH}(2) \subsetneq \text{CH}(1) \subsetneq \text{CH}(0)$.

Die erste Inklusion ist echt wegen $L_2 = \{a^n b^n \mid n \in \mathbb{N}\}$, die zweite wegen $L_3 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$.

Beispiel 77 $G_5 = (\{S, A\}, \{0, 1\}, P, S)$ mit

$$\begin{aligned} P = \{S &\to 0, & A &\to 1A, & A &\to \lambda\} \\ S &\to 1A, & A &\to 0A, \end{aligned}$$

Diese Grammatik ist kontextfrei und rechtslinear.

Frage: Was ist $L(G_5)$? $S \Rightarrow^* 0$, $S \Rightarrow^* 1$, $S \Rightarrow^* 11$, $S \Rightarrow^* 10$. Aber wir bekommen $00, 01, 000, \ldots \notin L(G_5)$. Damit ist $L(G_5) = \{\text{bin}(n) \mid n \in \mathbb{N}\}$.

Definition 37 Eine Typ-3-Grammatik $G = (N, T, P, S)$ ist eine *Typ-3-Normalform-Grammatik* genau dann, wenn

I. für alle Nicht-λ-Regeln gilt:
   1. $A \to aB$,
   2. (nicht relevant), oder
   3. $A \to a$

   mit $A, B \in N$ und $a \in T$.
II. Für alle λ-Regeln gilt: Es gibt höchstens die Regel $S \to \lambda$. Falls $S \to \lambda \in P$, dann kommt S auf keiner rechten Seite einer Regel vor.

Satz 18 Zu jeder Typ-3-Grammatik gibt es eine äquivalente Typ-3-Normalform-Grammatik.

Satz 19 $\mathrm{CH}(3) = \mathcal{REG}$.

Beweis: Wir zeigen $\mathrm{CH}(3) \subseteq \mathcal{REG}$. Sei $G = (N, T, P, S)$ eine Typ-3-Normalform-Grammatik. Wir konstruieren einen nichtdeterministischen endlichen Automaten $N_A = (Q, \Sigma, \delta, q_0, F)$, der $L(G)$ akzeptiert.

Dabei ist $Q = N \cup \{q_+\}$, $\Sigma = T$, $q_0 = S$ und $F = \{q_+\}$. Definition von δ:

- Für jede Regel $A \to aB$ ist $B \in \delta(A, a)$.
- Für jede Regel $A \to a$ ist $q_+ \in \delta(A, a)$.

Das heißt, $\delta(A, a) = \{B \mid (A \to aB) \in P\} \cup \{q_+ \mid (A \to a) \in P\}$. Sonderfall: Falls $(S \to \lambda) \in P$, dann setze $\delta(S, \lambda) = \{q_+\}$[14].

Korrektheit von N_A.

1. Fall $w = \lambda$: $w \in L(N_A)$ genau dann, wenn es den λ-Übergang $\delta(S, \lambda) = \{q_+\}$ gibt, genau dann, wenn es die λ-Regel $S \to \lambda$ in P gibt, genau dann, wenn $\lambda \in L(G)$.
2. Fall $w = a_1 a_2 \ldots a_n \neq \lambda$. $w \in L(N_A)$
   - genau dann, wenn es einen akzeptierenden Berechnungspfad von N_A bei Eingabe w gibt, d. h. es gibt eine Folge $K_0, K_1, \ldots, K_n$ mit $K_0 =$ Start-$K_{N_A}(w)$ und $K_{i-1} \vdash K_i$ und K_n ist akzeptierende Finalkonfiguration,
   - genau dann, wenn es eine Folge von Zuständen $A_0, A_1, \ldots, A_n$ mit $A_0 = S = q_0$, $A_n = q_+$ und $A_i \in \delta(A_{i-1}, a_i)$ für $i = 1, \ldots, n-1$ und $q_+ \in \delta(A_{n-1}, a_n)$ gibt,
   - genau dann, wenn es eine Folge von Nichtterminalen $A_0, A_1, \ldots, A_{n-1}$ mit $A_0 = S$, $A_{i-1} \to a_i A_i$ für $i = 1, \ldots, n-1$ und $A_{n-1} \to a_n$ gibt,
   - genau dann, wenn es in G eine Ableitung $S \Rightarrow^* w$ gibt,
   - genau dann, wenn $w \in L(G)$.

Also gilt $\mathcal{RLIN} \subseteq \mathcal{REG}$.

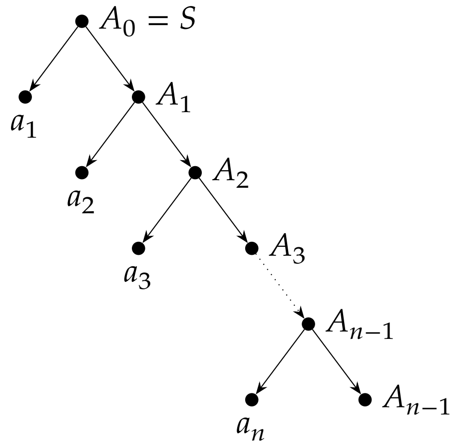
Wir zeigen noch $\mathcal{REG} \subseteq \mathcal{RLIN}$. Sei $L \in \mathcal{REG}$ und $A = (Q, \Sigma, \delta, q_0, F)$ ein DEA mit $L(A) = L$. Wir definieren die folgende Grammatik $G = (N, T, P, S)$ durch $N = Q$, $T = \Sigma$, $S = q_0$ und P durch folgende Regeln:

- $q \to ap$ (Form 1), falls $\delta(q, a) = p$ und
- $q \to a$ (Form 3), falls $\delta(q, a) = p$ und $p \in F$.

Korrektheit: $L(N) = L(A)$.

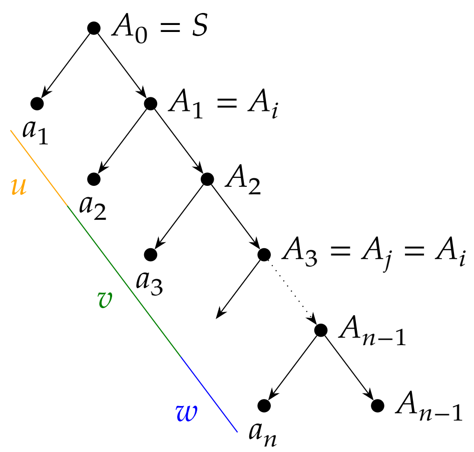
[14] Wir erinnern uns $\delta : Q \times \Sigma_\lambda \to P(Q)$.

1. Fall $w = \lambda$: $w \in L(A) \iff q_0 \in F \iff (S \to \lambda) \in P \iff w \in L(G)$.
2. Fall $w = a_1 a_2 ... a_n \neq \lambda$: $w \in L(G)$
   - genau dann, wenn es eine Ableitung $S \Rightarrow^* w$ in G gibt, die die folgende Form hat:



   - genau dann, wenn es eine Folge $A_0, A_1, ..., A_{n-1}$ von Nichtterminalen mit $A_0 = S$ gibt mit $A_0 \Rightarrow a_1 A_1 \Rightarrow a_1 a_2 A_2 \Rightarrow a_1 a_2 a_3 A_3 \Rightarrow ... \Rightarrow a_1 a_2 ... a_{n-1} A_{n-1} \Rightarrow a_1 a_2 ... a_n$,
   - genau dann, wenn es eine Folge von Zuständen $A_0, A_1, ..., A_{n-1}$ mit $A_0 = S = q_0$ und $\delta(A_{i-1}, a_i) = A_i$ für $i = 1, ..., n-1$ und $\delta(A_{n-1}, a_n) \in F$ gibt,
   - genau dann, wenn es einen akzeptierenden Berechnungspfad von A bei Eingabe w gibt,
   - genau dann, wenn $w \in L(A)$. □

Abbildung 14 Darstellung der Zerlegungen in regulären Sprachen für $z = uvw$ als Raupe mit Haarlänge 1, wobei der Teilbaum mit Wurzel A_j w erzeugt, der Teilbaum mit Wurzel A_i vw und der Teilbaum mit Wurzel $A_0 = S$ uvw erzeugt



Linkslineare Grammatiken

Definition 38 Eine kontextfreie Grammatik $G = (N, T, P, S)$ ist *linkslinear* genau dann, wenn alle Regeln in der Form $A \to Bw$ oder $A \to w$ für $A, B \in N$ und $w \in T^*$ sind.

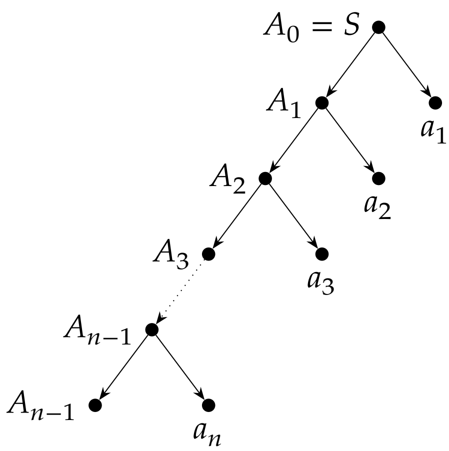
Satz 20 Zu jeder linkslinearen Grammatik G gibt es eine äquivalente linkslineare Grammatik in Normalform, d. h.

I. für alle Nicht-λ-Regeln gilt:
   1. $A \to Ba$,
   2. (nicht relevant), oder
   3. $A \to a$ mit $A, B \in N$ und $a \in T$.

II. Für alle λ-Regeln gilt: Es gibt höchstens die Regel $S \to \lambda$. Falls $S \to \lambda \in P$, dann kommt S auf keiner rechten Seite einer Regel vor.

Bemerkung 81 $\mathcal{LLIN}$ ist die Klasse der Sprache, die von linkslinearen Grammatiken erzeugt werden. Klar ist $\mathcal{LLIN} \subseteq \mathcal{CFL}$. Frage: Ist $\mathcal{LLIN} = \mathcal{RLIN}$?

Bemerkung 82 Ableitung einer linkslinearen Grammatik in Normalform:



Daraus folgt ein Pumping-Lemma für $\mathcal{LLIN}$.

Beispiel 78 Wir betrachten die linkslineare Grammatik $G_6 = (\{S, A, B\}, \{0, 1\}, P, S)$ mit

$$
\begin{aligned}
P_l = \{S &\to 0, & A &\to A0,\\
S &\to A0, & A &\to A1,\\
S &\to A1, & A &\to B1,\\
S &\to B1, & B &\to \lambda\}
\end{aligned}
$$

Beispiele für Wörter: $S \Rightarrow 0$, $S \Rightarrow B1 \Rightarrow 1$, $S \Rightarrow A0 \Rightarrow B10 \Rightarrow 10$, ...

$L(G_6) = \{\text{bin}(n) \mid n \in \mathbb{N}\} = L(G_5)$.

Definition 39 Sei $G = (N, T, P, S)$ eine linkslineare Grammatik in Normalform. Dann ist die *Spiegelgrammatik* $\text{Sp}(G)$ definiert durch

I. $A \to aB$, falls $(A \to Ba) \in P$, und $A \to a$, falls $(A \to a) \in P$, und
II. $S \to \lambda$ mit Zusatz *, falls $(S \to \lambda) \in P$.

Satz 21 Sei G eine linkslineare Grammatik in Normalform. Es gilt $L(\text{Sp}(G)) = \text{Sp}(L(G))$.

Beispiel 79 Beispiele rechtslinearer Sprachen:

- $L_1 = \{w \in \{a, b\}^* \mid |w|_a \equiv_5 0\}$. $\text{Sp}(L_1) = L_1$.
- $L_2 = \{w \in \{a, b\}^* \mid aba \sqsubsetneq w\}$. $\text{Sp}(L_2) = L_2$.
- $L_3 = \{w \in \{a, b\}^* \mid \text{das drittletzte Zeichen ist } a\}$. $\text{Sp}(L_3) \in \mathcal{REG}$

Satz 22 $\mathcal{REG} = \mathcal{LLIN}$.

In unserem Fall ist die Spiegelgrammatik einer linkslinearen Grammatik eine rechtslineare Grammatik in Normalform. Das bedeutet $\text{Sp}(\mathcal{LLIN}) = \mathcal{RLIN}$.

Wir wissen: $\mathcal{L}(\text{NEA})$ ist abgeschlossen bezüglich der Spiegeloperation, d. h. $\text{Sp}(\mathcal{L}(\text{NEA})) = \mathcal{L}(\text{NEA})$, d. h. $\text{Sp}(\mathcal{REG}) = \mathcal{REG}$. Also gilt $\mathcal{LLIN} = \text{Sp}(\text{Sp}(\mathcal{LLIN})) = \text{Sp}(\mathcal{RLIN}) = \text{Sp}(\mathcal{REG}) = \mathcal{REG}$. □

Beispiel 80 Grammatiken über $\{a, b\}^*$ für die Sprache, der Wörter, in denen die Anzahl der as durch fünf teilbar ist.

- rechtslinear: $G = (\{S, B, A_1, A_2, A_3, A_4\}, \{a, b\}, \{S \to \lambda, S \to b, S \to bB, B \to bB, B \to b, B \to aA_4, S \to aA_4, A_4 \to bA_4, A_4 \to aA_3, A_3 \to bA_3, A_3 \to aA_2, A_2 \to bA_1, A_2 \to aA_1, A_1 \to bA_1, A_1 \to a, A_1 \to aB\}, S)$
- linkslinear: alle Nichtterminale links statt rechts anhängen

Beispiel 81 Die Sprache der Wörter, in denen die Zeichenkette aba nicht vorkommt.

- rechtslinear: $G = (\{A, B\}, \{a, b\}, \{A \to \lambda, A \to a, A \to aA, A \to abB, B \to \lambda, B \to bB, B \to bA\}, A)$
- linkslinear: $G = (\{A, B\}, \{a, b\}, \{A \to \lambda, A \to a, A \to Aa, A \to Bba, B \to \lambda, B \to Bb, B \to Ab\}, A)$

Die rechtslineare Grammatik lässt das Einfügen von ab zu, aber nach einem ab nur ein leeres oder ein mit b beginnendes Wort. Die linkslineare Grammatik fügt vor ein ba nur ein leeres oder mit b endendes Wort ein. Beides führt dazu, dass das Teilwort aba nicht im Wort vorkommt. Die Links- bzw. Rechtsregularität ist offensichtlich.

Beispiel 82 Die Sprache der Wörter, in denen kein Paar aufeinanderfolgender as mehr vorkommt, sobald ein Paar aufeinanderfolgender bs vorgekommen ist.

- rechtslinear: $G = (\{A, A'\}, \{a, b\}, \{A \to \lambda, A' \to \lambda, A \to bA', A' \to bA', A \to aA, A \to baA, A' \to b, A' \to baA'\}, A)$
- linkslinear: $G = (\{A, A'\}, \{a, b\}, \{A \to \lambda, A' \to \lambda, A \to A'a, A \to Ab, A \to Aba, A' \to A'a, A' \to b, A' \to A'ab\}, A)$

Die rechtslineare Grammatik fügt so lange beliebig ein, bis zwei aufeinanderfolgende bs erscheinen. Danach wird mit der Produktionsregel A' sichergestellt, dass nur noch so eingefügt wird, dass ein b die Möglichkeit zweier aufeinanderfolgender as zerstört. Die linkslineare Grammatik löst durch vornanfügen das gleiche Problem. Bildete man die Spiegelregeln (die Reihenfolge der Symbole der rechten Seite spiegeln), dann löst sie in gleicher Weise wie die rechtslineare Grammatik das Problem, dass kein Paar aufeinanderfolgender bs mehr vorkommt, sobald ein Paar aufeinanderfolgender as vorgekommen ist. Die Links- bzw. Rechtsregularität ist offensichtlich.

4 Berechenbarkeit

4.1 Turingmaschinen

Einführung in Turingmaschinen

Turingmaschinen sind nach Alan Turing benannt (1936). Sie stellen eine Formalisierung des Algorithmenbegriffs dar.

Vorbild für die Turingmaschine ist der Mensch als Rechner:

- ein Rechenblatt
- Stift und Radiergummi
- den Kopf als Gedächtnis

Komponenten einer Turingmaschine:

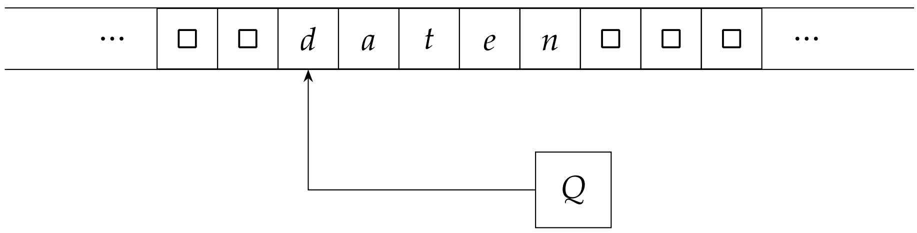
- ein Turing-Arbeitsband, das in Zellen unterteilt und beidseitig unendlich ist,
- ein Lese-Schreib-Kopf und
- ein endliches Gedächtnis – endliche Zustandsmenge

Definition 40 Eine *Turingmaschine* M ist ein 6-Tupel der Form $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$.

Q ist eine endliche Menge (Zustandsmenge)
Σ ist eine endliche Menge (Eingabealphabet)
Γ ist eine endliche Menge (Arbeitsalphabet) mit $\Sigma \subseteq \Gamma$ und $\square \in \Gamma \setminus \Sigma$ (Blank-Symbol)
q_0 $\in Q$ (Startzustand)
F $\subseteq Q$ (Finalzustände)
δ Überführungsfunktion mit $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, N, R\}$, das bedeutet

$$\delta(q, x) = \begin{cases} (p, x', L) \\ (p, x', N) \\ (p, x', R) \end{cases}$$

Abbildung 15 Darstellung des Turing-Arbeitsbandes



Die Arbeitsweise von M wird mittels *Konfiguration* und *Überführungsrelation* $\vdash$ beschrieben.

Definition 41 Gegeben sei eine Turingmaschine $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$. Eine *Konfiguration* K von M ist eine vollständige Beschreibung einer Momentansituation von M und die ist gegeben als Wort der Form $\Gamma^* \circ Q \circ \Gamma^*$, etwa $y_m \ldots y_2 y_1 q x_0 x_1 x_2 \ldots x_n$.

Dabei ist

- $y_m \ldots y_2 y_1 q x_0 x_1 x_2 \ldots x_n$ der beschriebene Teil des Arbeitsbandes (ein Symbol pro Zelle, stets endlich).
- q ist der aktuelle Zustand (stets links neben der aktuellen Position des Lese-Schreib-Kopfes).
- x_0 ist der Inhalt der Zelle, die der Lese-Schreib-Kopf sieht.

Definition 42 Eine Konfiguration K' ist *unmittelbare Nachfolgekonfiguration* von $K = y_m...y_2y_1qx_0x_1x_2...x_n$ genau dann, wenn

$$K' = \begin{cases} y_m...y_2py_1x'x_1x_2...x_n & \text{falls } \delta(q, x_0) = (p, x', L) \\ y_m...y_2y_1px'x_1x_2...x_n & \text{falls } \delta(q, x_0) = (p, x', N) \\ y_m...y_2y_1x'px_1x_2...x_n & \text{falls } \delta(q, x_0) = (p, x', R) \end{cases} .$$

Schreibweise: $K \vdash K'$. Dieser Übergang beschreibt einen Schritt bzw. einen Takt der Arbeit von M.

Definition 43 Für $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ und die Eingabe $w = w_1w_2...w_n \in \Sigma^*$ ist die *Startkonfiguration* Start-$K_M(w) = \lambda q_0 w_1 w_2...w_n$ (Spezialfall $w = \lambda$: Start-$K_M(\lambda) = q_0\square$.

Definition 44 Eine *Finalkonfiguration* ist eine Konfiguration mit einem Finalzustand.

Eine *normierte Finalkonfiguration* hat die Form $\lambda p x_1 x_2...x_k$ für $p \in F$.

Definition 45 Die *Berechnung* von M bei Eingabe w ist die Folge Start-$K_M(w) = K_0 \vdash K_1 \vdash K_2 \vdash K_3 \vdash ...$. Die Berechnung terminiert, falls es ein t gibt, sodass $K_0 \vdash K_1 \vdash K_2 \vdash K_3 \vdash ... \vdash K_t = K_{t+1} = K_{t+2}$. Dabei ist t die Anzahl der Takte (Dauer) der Berechnung.

Beispiel 83 $M = (\{q_0, q_1\}, \{1, 2\}, \{1, 2, \square\}, \delta, q_0, \{q_1\})$, wobei

δ	**1**	**2**	$\square$
q_0	$(q_0, 1, R)$	$(q_0, 2, R)$	$(q_0, \square, R)$
q_1	$(q_1, 1, N)$	$(q_1, 2, N)$	$(q_1, \square, N)$

In jedem Takt geht diese Maschine um eine Zelle nach rechts. Für jede Eingabe $w \in \{1, 2\}^*$ terminiert die Maschine nicht.

Diese Turingmaschine berechnet die nirgends definierte Funktion ν. Dabei ist $\nu(w) = \perp$ für alle $w \in \{1, 2\}^*$, d. h. $D_\nu = \emptyset$, also ist $\nu = \emptyset$.

Bemerkung 83 Funktionen, die durch Turingmaschinen berechnet werden, sind im Allgemeinen partiell definiert.

Definition 46 Eine Turingmaschine $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ berechnet eine *n-stellige Wortfunktion* $f : (\Sigma^*)^n \to \Delta^*$[15] genau dann, wenn

1. Fall: $(u_1, u_2, ..., u_n) \in D_f$. Dann gilt: aus der Start-Konfiguration $K_0 = (q_0 u_1 \# u_2 \# ... \# u_n)$ (mit $\# \in \Gamma \setminus \Sigma$) terminiert die Berechnung in einer normierten Finalkonfiguration $K_t = (q_E f(u_1, u_2, ..., u_n))$.

[15] Δ ist das Ausgabealphabet der Turingmaschine.

2. Fall: $(u_1, u_2, ..., u_n) \notin D_f$. Dann gilt: aus der Start-Konfiguration K_0 terminiert die Berechnung nicht oder terminiert, aber nicht mit einer Finalkonfiguration.

Definition 47 Eine Turingmaschine $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ berechnet eine *n-stellige Zahlenfunktion* $g : \mathbb{N}^n \to \mathbb{N}$ genau dann, wenn M die folgende n-stellige Wortfunktion $g' : (\{1,2\}^*)^n \to \{1,2\}$ berechnet. Für $g(z_1, z_2, ..., z_n)$ ist g' definiert oder ist nicht definiert. Dann ist $g'(\text{dya}(z_1), \text{dya}(z_2), ..., \text{dya}(z_n))$ mit $g(z_1, z_2, ..., z_n) = \text{dya}^{-1}(g'(\text{dya}(z_1), \text{dya}(z_2), ..., \text{dya}(z_n)))$.

Bemerkung 84 Es wird die dyadische Darstellung zu Grunde gelegt, da sie eine ein-eindeutige Zuordnung zwischen Zahlen und Wörtern garantiert.

Bemerkung 85 $\mathcal{TM}$ bezeichnet die Klasse der Turing-berechenbaren (Zahlen- bzw. Wort-)Funktionen.

Beispiel 84 Die Funktion $g(x) = 2x + 1$ für $x \in \mathbb{N}$ (total definiert) ist Turing-berechenbar.

Wir konstruieren folgende Turingmaschine $M = (Q, \{1,2\}, \{1,2,\square\}, \delta, q_0, F)$.

δ	**1**	**2**	$\square$
q_0	$(q_R, 1, R)$	$(q_R, 2, R)$	$(q_F, 1, N)$
q_R	$(q_R, 1, R)$	$(q_R, 2, R)$	$(q_L, 1, L)$
q_L	$(q_L, 1, L)$	$(q_L, 2, L)$	$(q_F, \square, L)$
q_F	$(q_F, 1, N)$	$(q_F, 2, N)$	$(q_F, \square, N)$

$Q = \{q_0, q_R, q_L, q_F\}$.

Bemerkung 86 δ ist total definiert. M arbeitet deterministisch.

Bemerkung 87 Alternative Formulierungen zu den letzten Definitionen.

Wortfunktionen: $f : (\Sigma^*)^n \to \Delta^*$ ist *Turing-berechenbar* genau dann, wenn

1. Fall: $(u_1, ..., u_n) \in D_f$. Dann gibt es eine terminierende Berechnung mit Finalzustand $q_0 u_1 \# u_2 \# ... \# u_n \vdash^* q_F f(u_1, u_2, ..., u_n)$[16].
2. Fall: $(u_1, u_2, ..., u_n) \notin D_f$. Dann gibt es keine terminierende Berechnung mit Finalzustand.

Im 1. Fall ist $f(u_1, u_2, ..., u_n) = \text{Res}_M(u_1, u_2, ..., u_n)$.

[16] $\vdash^*$ bezeichnet wie üblich die reflexive und transitive Hülle von $\vdash$.

Zahlenfunktionen: $g : \mathbb{N}^n \to \mathbb{N}$ ist Turing-berechenbar genau dann, wenn die durch $g'(u_1, u_2, ..., u_n) = \mathrm{dya}(g(\mathrm{dya}^{-1}(u_1), \mathrm{dya}^{-1}(u_2), ..., \mathrm{dya}^{-1}(u_n)))$ definierte Wortfunktion Turing-berechenbar ist.

Bemerkung 88 Der Begriff der Turing-Berechenbarkeit hängt nicht von der dyadischen Codierung der Zahlen ab. Wenn es für eine Zahlenfunktion g eine Turingmaschine M_{dya} gibt, die g in dyadischer Darstellung (mittels g') berechnet, dann lassen sich Turingmaschinen $M_{\mathrm{bin}}, M_{\mathrm{dezi}}, M_{\mathrm{un}}, ...$ konstruieren, die g in binärer bzw. dezimaler bzw. unärer bzw. ... Codierung berechnen.

Beispiel 85 $\mathrm{succ}(n) = n + 1$ für $n \in \mathbb{N}$ ist eine total definierte Funktion. Dann ist $\mathrm{succ}'(\mathrm{dya}(n)) = \mathrm{dya}(n + 1)$ für $n \in \mathbb{N}$. $\mathrm{succ}'(u) = \mathrm{dya}(\mathrm{dya}^{-1}(u) + 1)$ für $u \in \{1, 2\}^*$.

Wir konstruieren M_{dya} wie folgt: $M_{\mathrm{dya}} = (Q, \Sigma, \Gamma, \delta, q_0, F)$ mit $\Sigma = \{1, 2\}$, $\Gamma = \{1, 2, \square\}$ und $Q = \{q_0, q_R, q_C, q_L, q_F\}$.

δ	**1**	**2**	$\square$
q_0	$(q_R, 1, R)$	$(q_R, 2, R)$	$(q_F, 1, N)$
q_R	$(q_R, 1, R)$	$(q_R, 2, R)$	$(q_C, \square, L)$
q_C	$(q_L, 2, L)$	$(q_C, 1, L)$	$(q_F, 1, N)$
q_L	$(q_L, 1, L)$	$(q_L, 2, L)$	$(q_F, \square, R)$
q_F	$(q_F, 1, N)$	$(q_F, 2, N)$	$(q_F, \square, N)$

Beispiel 86 Testen, ob eine Eingabe $z = 0$ bzw. $\mathrm{dya}(z) = \lambda$. Ist ein solcher Test eine Funktion? Ja, sie kann für $z \in \mathbb{N}$ und $u = \{1, 2\}^*$ als charakteristische Funktion dargestellt werden:

$$\chi_{\{0\}}(z) = \begin{cases} 1 & \text{falls } z = 0 \\ 0 & \text{sonst} \end{cases} \qquad \text{bzw.} \qquad \chi_{\{\lambda\}}(u) = \begin{cases} 1 & \text{falls } u = \lambda \\ 0 & \text{sonst} \end{cases}$$

$M_{\mathrm{dya}} = (Q, \Sigma, \Gamma, \delta, q_0, F)$ mit $\Sigma = \{1, 2\}$, $\Gamma = \{1, 2, 0, \square\}$ und $Q = \{q_0\}$.

δ	**0**	**1**	**2**	$\square$
q_0	$(q_F, 0, N)$	$(q_A, \square, R)$	$(q_A, \square, R)$	$(q_F, 1, N)$
q_A	$(q_F, 0, N)$	$(q_A, \square, R)$	$(q_A, \square, R)$	$(q_F, 0, N)$
q_F	$(q_F, 0, N)$	$(q_F, 1, N)$	$(q_F, 2, N)$	$(q_F, \square, N)$

Techniken zur Konstruktion von Turingmaschinen

Idee: Aus „einfachen“ Turingmaschinen durch „Zusammensetzen“ komplexe Maschinen konstruieren.

- Hintereinanderschalten von Turingmaschinen. Wir betrachten $M_{\mathrm{dya}} = M_{+1}$. Für eine Eingabe $u \in \{1,2\}^*$ erhalten wir

  $$u \longrightarrow \boxed{M_{+1}} \longrightarrow \mathrm{dya}(\mathrm{dya}^{-1}(u) + 1)$$

  bzw. für eine Eingabe $z \in \mathbb{N}$

  $$z \longrightarrow \boxed{M_{+1}} \longrightarrow z + 1$$

  Das Hintereinanderschalten hat den Effekt, dass bei Eingabe $z \in \mathbb{N}$ die Ausgabe $z + 2$ ist:

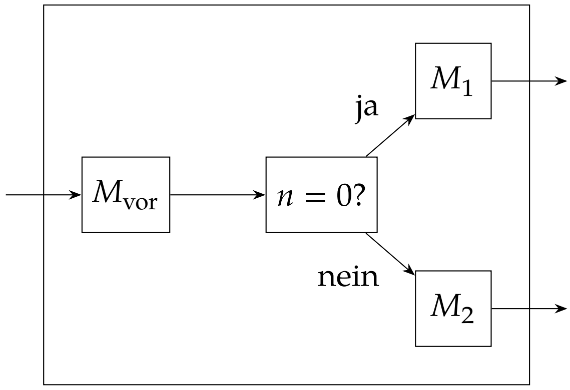
  $$z \longrightarrow \underbrace{\boxed{\boxed{M_{+1}} \longrightarrow z + 1 \longrightarrow \boxed{M_{+1}}}}_{M_{+2}} \longrightarrow z + 2$$

  Allgemein wollen wir folgenden Effekt für eine Eingabe z:

  $$z \longrightarrow \underbrace{\boxed{\boxed{M_g} \longrightarrow g(z) \longrightarrow \boxed{M_h}}}_{M} \longrightarrow h(g(z))$$

  Gegeben zwei Turingmaschinen $M_i = (Q_i, \Sigma, \Gamma_i, \delta_i, q_{0_i}, F_i)$ für $i = 1, 2$. Dann ist die neue Maschine M wie folgt definiert: $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, wobei
  - $Q = Q_1 \cup Q_2$, wobei $Q_1 \cap Q_2 = \emptyset$,
  - $\Gamma = \Gamma_1 \cup \Gamma_2$,
  - $q_0 = q_{0_1}$,
  - $F = F_2$ und
  - $\delta = \delta_1 \cup \delta_2 \cup \{(q_F, a, q_{0_2}, a, N) \mid q_F \in F_1, a \in \Sigma\}$.

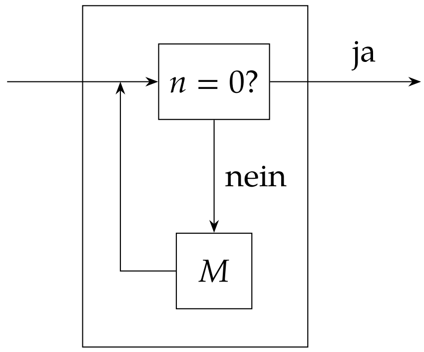
  Schreibweise: $M = M_1; M_2$. Wenn M_1 die Funktion g berechnet und M_2 die Funktion h berechnet, dann berechnet M die Funktion $g \circ h$.
- Turingmaschinen, die eine Fallunterscheidung realisieren.



M_{FU}

$M_{\text{FU}} = M_{\text{vor}}$; if $n = 0$ then M_1 else M_2.

- Turingmaschinen, die Schleifen realisieren.



M_S

M_S = while $n \neq 0$ do M

- Turingmaschinen, die Unterprogramme realisieren.

Beispiel 87 Ein Unterprogramm M_u von M, das nach Aufruf den aktuellen Bandinhalt von M um eine Zelle nach rechts verschiebt. Wir konstruieren M_u wie folgt: $Q_u = \{q_0, q_L\} \cup \{q_x \mid x \in \Gamma\}$ (q_0 wird aufgerufen, wenn sich der Lese-Schreib-Kopf von M auf dem ersten Symbol, d. h. links, der Eingabe befindet).

$\boldsymbol{\delta_u}$	$\boldsymbol{x}$	$\square$
q_0	$(q_x, \square, R)$	$(q_T, \square, N)$
q_y	(q_x, y, R)	(q_L, y, L)
q_L	(q_L, x, L)	$(q_T, \square, R)$

Bemerkung 89 Turingmaschinen können Programme realer Programmiersprachen simulieren. Mit anderen Worten: Alles, was durch Programme berechenbar ist, ist Turing-berechenbar.

Beispiel 88 Es ist eine Standard-Turingmaschine M_f zu konstruieren, die $f : \{0,1\}^* \rightarrow \{|\}^*$ berechnet, sodass $f(\text{bin}(n)) = \text{un}(n)$ mit $n \in \mathbb{N}$. f ist 1-stellig und f ist total definiert für alle $\text{Bin}(\mathbb{N}) = \{\text{bin}(n) \mid n \in \mathbb{N}\}$. $D_f = \text{Bin}(\mathbb{N})$.

M_f wird konstruiert für Eingaben, die Binärzahlen sind, wobei f für jede derartige Eingabe stoppt. Aus einer Eingabe der Länge l wird eine Ausgabe der Länge 2^l erzeugt. Das heißt, jede Maschine, die f berechnet, hat exponentiellen Aufwand $\mathcal{O}(2^l)$. Strategien für die Arbeitsweise:

- pro Ziffer der Binärdarstellung eine Berechnungsetappe der Maschine (z. B. die Unärzahl verdoppeln und die Ziffer addieren, also Horner-Schema)
- pro Einheit der Binärzahl 1 von der Binärzahl abziehen und 1 auf die Unärzahl addieren

Illustration an einem Beispiel. Sonderfall: $\square 0 \square$ wird zu $\square\square\square$.

Beispiel 89 $L = \{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$ mit der charakteristischen Funktion

$$\chi(w) = \begin{cases} 1 & \text{für } |w|_a = |w|_b \\ 0 & \text{sonst} \end{cases}$$

$M = (\{q_0, q_{S_a}, q_{S_b}, q_R, q_C, q_S, q_F\}, \{a,b\}, \{a,b,0,1,\#,\square\}, \delta, q_0, \{q_F\})$

δ	**a**	**b**	**0**	**1**	$\square$	#
q_0	$(q_{S_b}, \#, R)$	$(q_{S_a}, \#, R)$	$(q_F, 0, N)$	$(q_F, 0, N)$	$(q_C, 1, L)$	$(q_0, \#, R)$
q_{S_a}	$(q_R, \#, L)$	(q_{S_a}, b, R)	$(q_F, 0, N)$	$(q_F, 0, N)$	$(q_C, 0, L)$	$(q_{S_a}, \#, R)$
q_{S_b}	(q_{S_b}, a, L)	$(q_R, \#, R)$	$(q_F, 0, N)$	$(q_F, 0, N)$	$(q_C, 0, L)$	$(q_{S_b}, \#, R)$
q_R	(q_R, a, L)	(q_R, b, L)	$(q_F, 0, N)$	$(q_F, 0, N)$	$(q_0, \square, R)$	$(q_R, \#, L)$
q_C	$(q_C, \square, L)$	$(q_C, \square, L)$	$(q_F, 0, N)$	$(q_F, 0, N)$	$(q_S, \square, R)$	$(q_C, \square, L)$
q_S	(q_F, a, N)	(q_F, b, N)	$(q_F, 0, N)$	$(q_F, 1, N)$	$(q_S, \square, R)$	$(q_F, \#, N)$
q_F	(q_F, a, N)	(q_F, b, N)	$(q_F, 0, N)$	$(q_F, 1, N)$	$(q_F, \square, N)$	$(q_F, \#, N)$

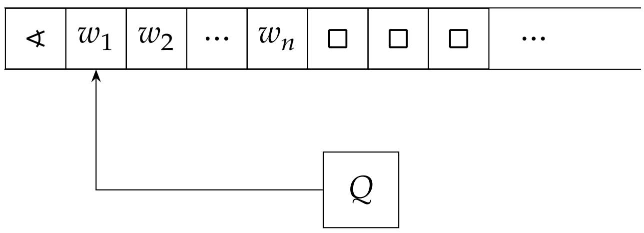
Die Turingmaschine überschreibt ein a mit # (q_0) und sucht nach einem entsprechenden b (q_{S_b}), das dann ebenso überschrieben wird. Für b und a funktioniert das genauso. Nach jeder Iteration wird mit q_R zurück zum Wortanfang gesprungen. Kann kein Partner gefunden werden, wird der Misserfolg aufs Band geschrieben und der restliche Bandinhalt ausgelöscht (q_C). Im durchgehenden Erfolgsfall, stößt q_0 auf ein Blank, schreibt eine 1 und lässt dann ebenfalls löschen. Nach der Löschung wird mit q_S zum Output (0 oder 1) gesprungen, wo der Lese-Schreib-Kopf final verharrt.

Dabei bezeichnen wir das bisherige Modell $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ als *Standard-Turingmaschine*.

Turingmaschinen mit Halbband

Das Turingarbeitsband ist nur einseitig unendlich. Vereinbarung: Der linke Rand ist markiert mit einer Endmarke $\sphericalangle$, die weder überschrieben noch überschritten werden darf. Die Eingabe steht links.

Abbildung 16 Skizze einer Turingmaschine mit Halbband



Satz 23 Jede Standard-Turingmaschine kann durch eine Halbband-Turingmaschine M' simuliert werden.

Beweisidee: Grundsätzlich arbeiet M' wie M. Falls M' die linke Endmarke sieht, dann ruft sie das „Unterprogramm“ M_U auf, das den kompletten Bandinhalt um eine Zelle nach rechts schiebt und in die erste Zelle ein (Pseudo-)Blank $\square'$ schreibt und damit den freien Platz erzeugt, falls M eine Bewegung L vollzieht.

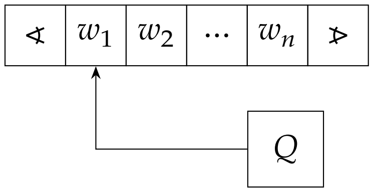
Bemerkung 90 Welchen Preis zahlt M'? Welchen Aufwand zahlt M'? Wir nehmen an, dass M für Eingabe $w \in \Sigma^*$ der Länge $|w| = n$ $\mathcal{O}(t(n))$ Teile arbeitet. Die Länge des benutzten Arbeitsbandes ist dann ebenfalls durch $\mathcal{O}(t(n))$ beschränkt.

Im „worst case“ wird in jedem Takt der Arbeit von M durch M' das Unterprogramm M_U aufgerufen. Ein solcher Aufruf kostet $\mathcal{O}(t(n))$ Takte. Also ist der Aufwand von M' durch $\mathcal{O}(t^2(n))$ beschränkt.

Linear beschränkte Automaten

Das sind Turingmaschinen, deren Arbeitsband beidseitig der Eingabe „abgeschnitten ist“.

Abbildung 17 Skizze eines linear beschränkten Automaten



Bemerkung 91 Linear beschränkte Automaten (LBA) haben *nicht* die Berechnungskraft von Turingmaschinen.

Beispiel 90 Es gibt einen linear beschränkten Automaten, der die charakteristische Funktion der Sprache $L_3 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ berechnet. Mit anderen Worten: Es gibt eine Standard-Turingmaschine M, die die charakteristische Funktion von L_3 berechnet und dazu keinen zustäzlichen Platz benötigt.

Wir wissen $L_3 \in \mathcal{CSL} \setminus \mathcal{CFL}$, also $L_3 \in \mathrm{CH}(1) \setminus \mathrm{CH}(2)$. L_3 wird durch eine nicht-verkürzende Grammatik erzeugt. Das bedeutet: alle „Zwischenwörter“ sind nicht länger als $a^n b^n c^n$. Das ist der Grund, weshalb das Wortproblem für CH(1)-Sprachen lösbar ist.

Satz 24 Linear beschränkte Automaten können nicht verkürzende Grammatiken simulieren.

Bemerkung 92 Das bedeutet: Linear beschränkte Automaten können für Sprachen aus CH(1) die charakteristische Funktion berechnen.

Turingmaschinen mit mehreren Spuren

Idee: Das Turingarbeitsband wird – während der Berechnung – in mehrere Etagen (Spuren) unterteilt.

Beispiel 91 Illustration an der Addtition:

- Standardturingmaschine:

$\cdots$	$\square$	$\mathrm{dya}(n)$	#	$\mathrm{dya}(m)$	$\square$	$\cdots$

- Band mit 3 Spuren:

<table>
<tr><td>⋯</td><td>□</td><td>□</td><td>□</td><td>dya(n)</td><td>□</td><td>⋯</td></tr>
<tr><td>⋯</td><td>□</td><td>□</td><td colspan="2">dya(m)</td><td>□</td><td>⋯</td></tr>
<tr><td>⋯</td><td>□</td><td colspan="3">dya($n + m$)</td><td>□</td><td>⋯</td></tr>
</table>

rechtsbündig

Aufwand der Addition mit 3 Spuren: $\mathcal{O}(l)$ mit l als der Länge der Eingabe. Mit einem gewöhnlichen Turing-Band wird $\mathcal{O}(l \cdot l) = \mathcal{O}(l^2)$ benötigt.

Aber: Wie erhalten wir 3 Spuren? Das Arbeitsalphabet besteht aus 3-Tupeln $(\gamma_1, \gamma_2, \gamma_3)$, speziell $(\square, \square, \square)$ als neues Blank.

Bemerkung 93 Eine Turingmaschine mit k Spuren ist im Grunde kein neues Modell, sondern besitzt ein besonderes Arbeitsalphabet, dessen Elemente k-Tupel von Buchstaben sind.

Für jede Standard-Turingmaschine lassen sich prinzipiell die k Spuren einrichten. Illustration:

- Eingabe:

<table>
<tr><td>⋯</td><td>□</td><td>dya(n)</td><td>#</td><td>dya(m)</td><td>□</td><td>⋯</td></tr>
</table>

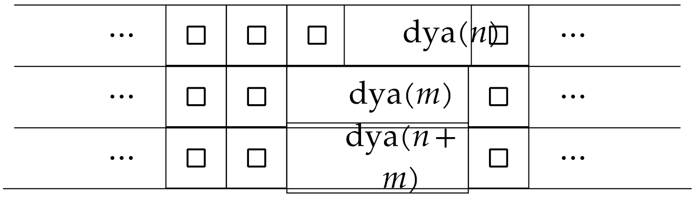
- 1. Etappe: Einrichten der Spuren mit Aufwand $\mathcal{O}(l)$

<table>
<tr><td>⋯</td><td>□</td><td colspan="3">dya(n)</td><td>#</td><td colspan="3">dya(m)</td><td>□</td><td>⋯</td></tr>
<tr><td>⋯</td><td>□</td><td>□</td><td></td><td>□</td><td>□</td><td></td><td></td><td>□</td><td>□</td><td>⋯</td></tr>
<tr><td>⋯</td><td>□</td><td>□</td><td></td><td>□</td><td>□</td><td></td><td></td><td>□</td><td>□</td><td>⋯</td></tr>
</table>

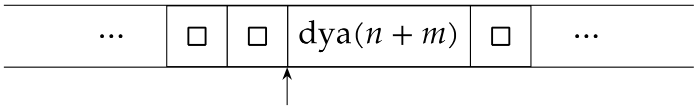
- 2. Etappe: stellenweise Addition mit Aufwand $\mathcal{O}(l^2)$

<table>
<tr><td>⋯</td><td>□</td><td>□</td><td>□</td><td colspan="2">dya(n)</td><td>□</td><td>⋯</td></tr>
<tr><td>⋯</td><td>□</td><td>□</td><td colspan="3">dya(m)</td><td>□</td><td>⋯</td></tr>
<tr><td>⋯</td><td>□</td><td>□</td><td>⋯</td><td>⋯</td><td>⋯</td><td>□</td><td>⋯</td></tr>
</table>

- 3. Etappe: stellenweise Addition mit Aufwand $\mathcal{O}(l)$

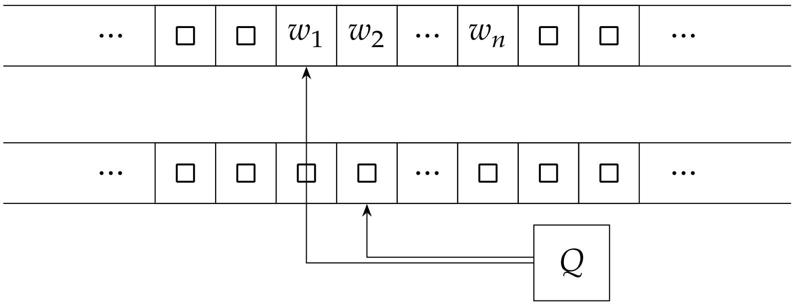


- 4. Etappe: „Band aufräumen“ mit Aufwand $\mathcal{O}(l)$



Turingmaschinen mit mehreren Bändern

2 Turingbänder:

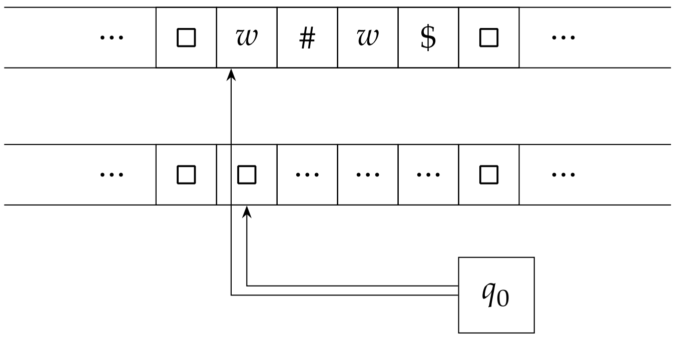


Vereinbarung:

- Jedes Band besitzt einen Lese-Schreib-Kopf, der sich „frei“ bewegen kann.
- Die Eingabe steht auf Band 1, der Kopf auf dem 1. Symbol. Alle anderen Bänder sind zunächst leer; die Köpfe stehen irgendwie.
- Die Ausgabe steht auf Band 1, alle anderen sind leer.

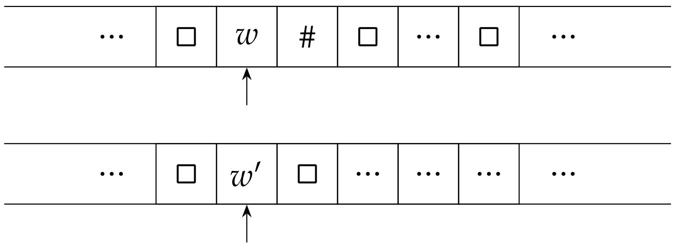
Beispiel 92 Eine 2-Band-Turingmaschine, die die charakteristische Funktion von $L_{\text{copy}} = \{w\,\#\,w\$ \mid w \in \{a,b\}^*\}$ berechnet.

- Eingabe:



- 1. Etappe:

  - Kopf 1 läuft nach rechts bis zu #
  - Kopf 2 bleibt stehen
  - Aufwand: $\mathcal{O}(l)$
- 2. Etappe:
  - Kopf 1 läuft nach rechts bis zu $
  - Kopf 2 kopiert die gelesenen Symbole
  - Aufwand: $\mathcal{O}(l)$
- 3. Etappe:
  - Kopf 1 und Kopf 2 nach links
  - Aufwand: $\mathcal{O}(l)$



- 4. Etappe:
  - stellenweiser Vergleich
  - Aufwand $\mathcal{O}(l)$
- 5. Etappe: Abschluss in $\mathcal{O}(l)$

Insgesamt ist der Aufwand der 2-Band-Turingmaschine $\mathcal{O}(l)$. Dagegen ist der Aufwand einer 1-Band-Turingmaschine insgesamt $\mathcal{O}(l^2)$.

Satz 25 Für jede k-Band-Turingmaschine M gibt es eine Standard-Turingmaschine M'.

Beweis: Gegeben ist eine k-Band-Turingmaschine $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ mit $\delta : Q \times \Gamma^k \to Q \times (\Gamma \times \{L, N, R\})^k$. Wir konstruieren eine Turingmaschine M' mit $2k + 1$ Spuren, wobei

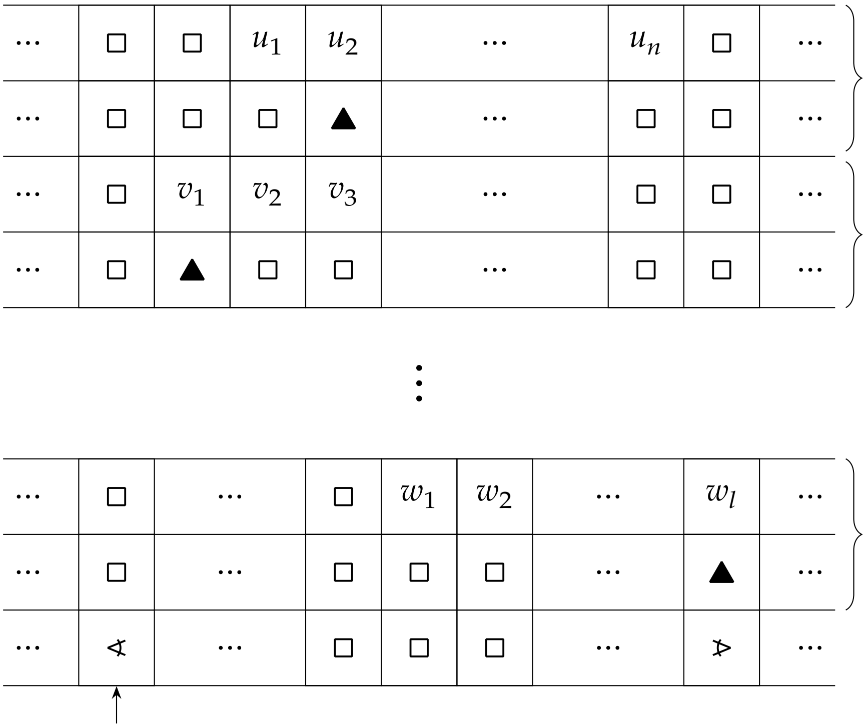
- die Spur $2i - 1$ den Inhalt von Band i enthält (für $1 \leq i \leq k$),
- die Spur $2i$ die Kopfposition auf Band i markiert (für $1 \leq i \leq k$) und
- die Spur $2k + 1$ eine linke $\ntriangleleft$ und rechte $\ntriangleright$ Endmarke enthält.

Dies ist in Abbildung 18 skizziert.

Um einen Takt der Arbeit von M zu simulieren, arbeitet M' wie folgt:

- 1. Etappe: Der Lese-Schreib-Kopf von M' steht auf der linken Endmarke $\ntriangleleft$. Der Kopf bewegt sich zur rechten Endmarke $\ntriangleright$ und in den Zuständen von M' werden die k aktuellen Symbole gespeichert.
- 2. Etappe: Der Lese-Schreib-Kopf von M' steht auf $\ntriangleright$. Der Kopf bewegt sich nach links zu $\ntriangleleft$ und

Abbildung 18 Darstellung der k-Band-Turingmaschine als $2k+1$-Spuren-Turingmaschine



- schreibt die k neuen Symbole,
- markiert die k neuen Kopfpositionen
- mit der Besonderheit, rechte bzw. linke Endmarke werden je um eine Zelle verschoben, falls sie erreicht werden.

Um einen Takt der Arbeit von M zu simulieren, benötigt M' $\mathcal{O}(t(n))$ Takte, wobei $\mathcal{O}(t(n))$ die Rechenzeit von M ist. Also ist der Gesamtaufwand von M' $\mathcal{O}(t^2(n))$.

Bemerkung 94 Das heißt im Konkreten: Wenn M eine 2-Band-Turingmaschine ist, die in der Zeit $\mathcal{O}(n)$ arbeitet, dann benötigt M' die Zeit $\mathcal{O}(n^2)$.

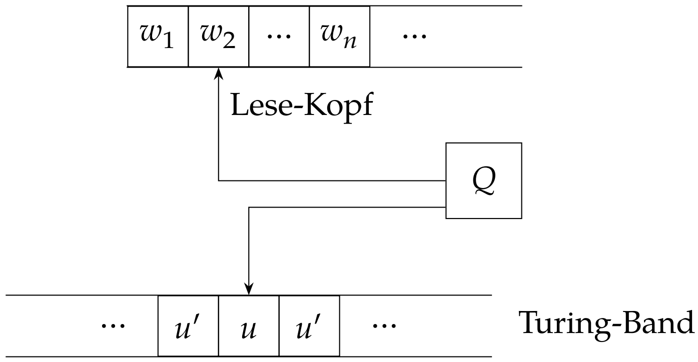
Weitere Varianten

- Turingbänder mit mehreren Köpfen.
  Nutzen: Eine 1-Band-Turingmaschine mit 2 Köpfen „schafft“ L_{copy} ebenfalls in linearer Zeit $\mathcal{O}(n)$.
- mehrdimensionale Turingbänder
- „Mischformen“

Alle weiteren Varianten und Mischformen können letztendlich nicht mehr leisten als die Standard-Turingmaschine.

Eine hervorzuhebende Variante ist die Turingmaschine mit Eingabe-Band. Diese nützt ebenfalls und „schafft“ L_{copy} in linearer Zeit.

Abbildung 19 Skizze einer Turingmaschine mit Eingabe-Band



Beispiel 93 Sei $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ eine Standard-Turingmaschine, die auf Eingaben der Länge n höchstens $s(n)$ Zellen des Arbeitsbandes benutzt. Es ist zu zeigen: Wenn M auf einer Eingabe der Länge n stoppt, dann stoppt M nach höchstens $|Q| \cdot |\Gamma|^{s(n)} \cdot s(n)$ Takten.

Wenn M auf einer Eingabe der Länge n stoppt, dann hat sie in diesem Prozess höchstens $s(n)$ Zellen des Arbeitsbandes benutzt. Im Prozess der Verarbeitung in der Turingmaschine können höchstens alle $|\Gamma|^{s(n)}$ verschiedenen Konfigurationen auftreten, auch wenn einige unzulässig sein könnten. Dabei kann jeweils der Lese-Schreib-Kopf an jeder Position des Arbeitsbereiches stehen, also an $s(n)$ verschiedenen Zellen. Jedes Kopf-Konfigurations-Tupel kann in jedem der $|Q|$ verschiedenen Zustände erreicht werden. Würde die Maschine mehr Takte benötigen, würde sie ununterscheidbar von einem anderen Kopf-Konfigurations-Zustands-Tripel erfasst werden. Diese Ununterscheidbarkeit sorgt für eine nicht endende Schleife im Ablauf. Da wir allerdings wissen, dass die Maschine terminiert, darf dieser nicht auftreten. □

Turingmaschine als Erkenner bzw. als Akzeptoren

- Erkenner (bzw. Entscheider): Zu jeder Turingmaschine M, die eine charakteristische Funktion χ_L berechnet, gibt es eine äquivalente Turingmaschine M' ohne Ausgabe, die einen annehmenden Finalzustand q_+ erreicht, falls M 1 produziert, und die einen ablehnenden Finalzustand q_- erreicht, falls M 0 produziert.

- Akzeptoren: Zu jeder Turingmaschine, die eine partielle (charakteristische) Funktion berechnet, gibt es eine äquivalente Turingmaschine M' ohne Ausgabe, die einen annehmenden Finalzustand q_+ erreicht, falls M terminiert, und die nicht terminiert, falls M nicht terminiert.

Definition 48 Sei $L \subseteq \Sigma^*$ bzw. $A \subseteq \mathbb{N}$. Die *charakteristische Funktion* χ_L (bzw. χ_A) ist für $w \in \Sigma^*$ definiert durch

$$\chi_L(w) = \begin{cases} 1 & \text{falls } w \in L \\ 0 & \text{falls } \notin L \end{cases}$$

mit $D_{\chi_L} = \Sigma^*$.

Die *partiell-charakteristische Funktion* χ_L^p (bzw. χ_A^p ist definiert durch

$$\chi_L^p(w) = \begin{cases} 1 & \text{falls w??L} \\ \perp & \text{sonst (nicht definiert)} \end{cases}$$

mit $D_{\chi_L^p} = L$.

Bemerkung 95 *These von* Church:

- Jede in einem intuitiven Sinne durch einen Algorithmus berechenbare Funktion f ist Turing-berechenbar.
- Jedes in einem intutiven Sinne durch einen Algorithmus lösbare Problem L ist Turing-„lösbar“.

Bemerkung 96 Kommentar zur Church-Turing-These: Diese *These* ist nicht beweisbar, da der Begriff „intuitiv“ nicht formalisiert ist.

Satz 26 *Hauptsatz der Algorithmentheorie*: Sei X eine konkrete Formalisierung des intuitiven Algorithmenbegriffs. Dann gilt:

- Eine Funktion $f : (\Sigma^*)^n \to \Delta^*$ ist X-berechenbar genau dann, wenn f Turing-berechenbar ist.
- Ein Problem $L \subseteq \Sigma^*$ ist X-lösbar genau dann, wenn L Turing-lösbar ist.

Bemerkung 97 Für X gilt dieser Satz, wenn X eine der folgenden Formalisierungen ist: „Markov-Algorithmen“, „Flussbilder“, „partiell-rekursive Funktionen“, „Programme“, ...

- Darstellung als Transitionsdiagramm.

$$q \xrightarrow{x;y;L} p$$

- Tabelle für δ

δ	$\mathbf{\Gamma}$
Q	

- „Liste“ der einzelnen Befehle
- 3-Band-Turingmaschine $M = (Q, \Sigma, \Gamma = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3, \delta, q_0, F)$ mit $\delta : Q \times \Gamma^3 \longrightarrow Q \times (\Gamma \times \{L, N, R\})^3$, Möglichkeit einer Tabelle der Form

δ	$\mathbf{\Gamma_1}$	$\mathbf{\Gamma_2}$	$\mathbf{\Gamma_3}$	$\boldsymbol{Q}$	$\mathbf{\Gamma_1}$	$\{\boldsymbol{L}, \boldsymbol{N}, \boldsymbol{R}\}$	$\mathbf{\Gamma_2}$	$\{\boldsymbol{L}, \boldsymbol{N}, \boldsymbol{R}\}$	$\mathbf{\Gamma_3}$	$\{\boldsymbol{L}, \boldsymbol{N}, \boldsymbol{R}\}$
Q										

- Konfigurationen

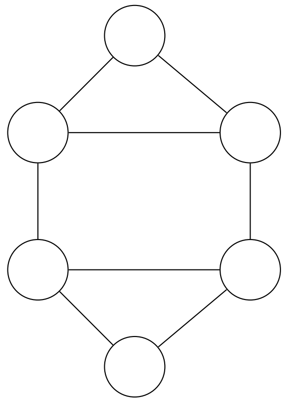
Beispiel 94 Es ist eine Turingmaschine ohne Eingabe gesucht, die für $\Sigma = \{a, b\}$ durch # getrennt Σ^* in kanonischer Ordnung erzeugt.

Da die kanonische Ordnung als dyadische Darstellung aufgefasst werden kann, können wir einen Inkrementer implementieren. Angefangen bei der Folge #*a*# wird das letzte Wort kopiert und dann inkrementiert. Dabei muss ggf. nach rechts geschoben werden.

4.3 Entscheidbarkeit und Aufzählbarkeit

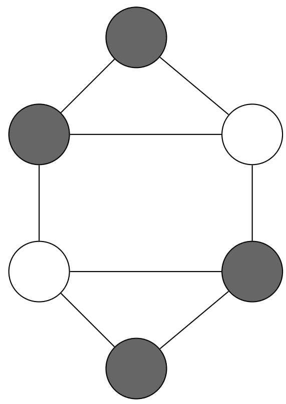
Beispiel 95 Wir betrachten das Graphenproblem der Knotenüberdeckung.

- Eingabe: ein Paar $(\text{Code}(G), \text{dya}(k))$, z. B. $(G, 3)$



- Frage: Gibt es eine Knotenüberdeckung von G mit der maximalen Größe 3?

Definition 49 $G = (V, E)$ sei ein einfacher Graph, das heißt $E \subseteq \binom{V}{2}$. $V' \subseteq V$ ist eine *Knotenüberdeckung* von G genau dann, wenn für alle $e \in E$ gilt $\exists v \in V' : e \cap \{v\} \neq \emptyset$, das heißt jede Kante hat einen Endknoten in V'.



Eine Knotenüberdeckung mit 4 Knoten ist möglich. Da ein K_3 stets zwei Knoten zur Überdeckung benötigt, geht es auch nicht besser.

$$\text{KÜ} = \{(\text{Code}(G), \text{dya}(k)) \mid G \text{ besitzt eine Knotenüberdeckung maximaler Größe } k\}$$

Beispiel 96 Prim $= \{n \in \mathbb{N} \mid n \text{ ist Primzahl}\}$.

Bemerkung 98 Die obigen beiden Beispiele sind zwei Probleme, die Turing-entscheidbar sind.

Definition 50

- Eine Sprache $L \subseteq \Sigma^*$ ist *Turing-entscheidbar* genau dann, wenn die charakteristische Funktion

$$\chi_L(w) = \begin{cases} 1 & \text{falls } w \in L \\ 0 & \text{falls } w \notin L \end{cases}$$

  für $w \in \Sigma^*$ als Wortfunktion Turing-berechenbar ist.
- Eine Zahlenmenge $A \subseteq \mathbb{N}$ ist *Turing-entscheidbar* genau dann, wenn die charakteristische Funktion

$$\chi_A(n) = \begin{cases} 1 & \text{falls } n \in A \\ 0 & \text{falls } n \notin A \end{cases}$$

  für ein $n \in \mathbb{N}$ als Zahlenfunktion Turing-berechenbar ist.

Beispiel 97 Prim$' = \{\text{dya}(n) \mid n \text{ ist Primzahl}\}$ ist Turing-entscheidbar.

Bemerkung 99 $\mathcal{REC}$ bezeichnet die Klasse aller Turing-entscheidbaren Probleme (Sprachen und Zahlenmengen).

Es gilt zum Beispiel KÜ, Prim, Prim$' \in \mathcal{REC}$.

Beispiel 98 Ist $L = \{ww \mid w \in \{a,b\}^*\}$ Turing-entscheidbar? Es ist trivial zu zeigen, dass für $w\,\#\,w\#$ sowie $w\,\#\,w$ statt ww die Turing-Entscheidbarkeit gegeben ist. Für das gegebene L gibt es mehrere Möglichkeiten, dies zu zeigen. Eine einfache ist ein Vorprozess, der die Mitte bestimmt, um das Problem auf $w\,\#\,w$ zurückzuführen.

- Trennung: Dazu nehmen wir eine Turingmaschine, die die äußeren Blanks „in die Mitte wandern lässt". Das Wort ist getrennt, wenn zwei Blanks nebeneinander stehen. Dann hat es de facto die Form $w\,\#\,w$ mit $\square\square$ statt # und da dies Turing-entscheidbar ist, ist auch L Turing-entscheidbar. Das hat einen Aufwand in $\mathcal{O}(n^2)$.
- 2-Band-Turingmaschine: Auf dem zweiten Band speichern wir die Länge von ww in unärer Form $(\mathcal{O}(n))$. Danach wird jedes zweite unäre Symbol gestrichen, sodass nur die Länge von w stehen bleibt $(\mathcal{O}(n))$. Im Anschluss schreiben wir auf jeden verbleibenden Strich ein Symbol vom ersten Band, sodass das zweite Band die Form $\square w_1 \square w_2 \square \ldots \square w_n$ hat $(\mathcal{O}(n))$.
- Zusatzspur: Die Zusatzspur wird zur Markierung der ersten und zweiten Hälfte genutzt. Die Markierungen treffen in der Mitte aufeinander.

Der bitweise Vergleich am Ende kostet kostet mit zwei Bändern $\mathcal{O}(n)$ und $\mathcal{O}(n^2)$ sonst.

Definition 51

- Eine Sprache $L \subseteq \Sigma^*$ ist *Turing-semientscheidbar* genau dann, wenn die partiell-charakteristische Funktion

$$\chi_L^p(w) = \begin{cases} 1 & \text{falls } w \in L \\ \perp & \text{falls } w \notin L \end{cases}$$

  für $w \in \Sigma^*$ als Wortfunktion Turing-berechenbar ist.
- Eine Zahlenmenge $A \subseteq \mathbb{N}$ ist *Turing-semientscheidbar* genau dann, wenn die partiell-charakteristische Funktion

$$\chi_A^p(n) = \begin{cases} 1 & \text{falls } n \in A \\ \perp & \text{falls } n \notin A \end{cases}$$

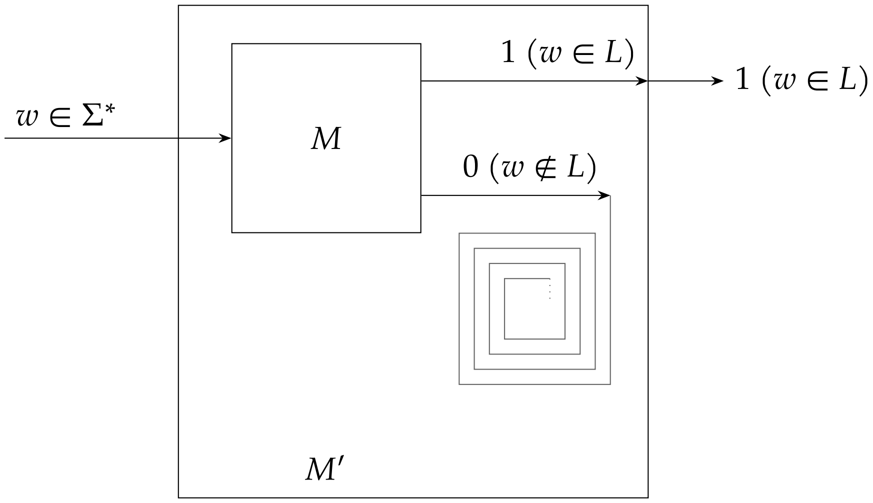
  für $n \in \mathbb{N}$ als Zahlenfunktion Turing-berechenbar ist.

Bemerkung 100 $\mathcal{RE}$ bezeichnet die Klasse aller Turing-semientscheibaren Probleme.

Lemma 4 Es gilt $\mathcal{REC} \subseteq \mathcal{RE}$.

Beweis: Sei $L \in \mathcal{REC}$. Dann gibt es eine Turingmaschine M, die χ_L berechnet.

Abbildung 20 Skizze zum Beweis von Lemma 4

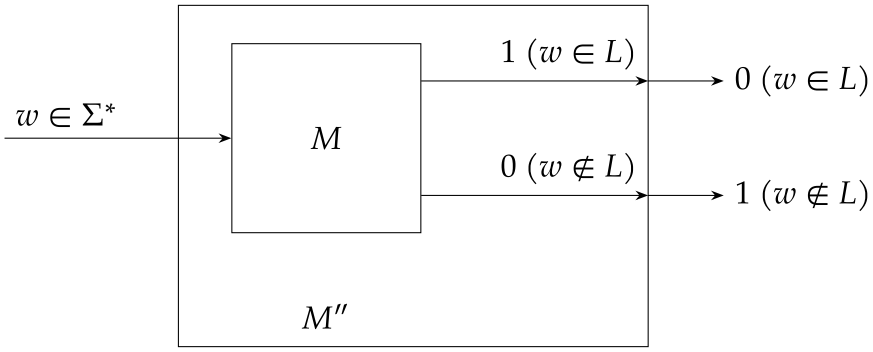


M' (siehe Abbildung 20) berechnet χ_L^p, d. h. $L \in \mathcal{RE}$.

Lemma 5 $L \in \mathcal{REC} \implies \overline{L} \in \mathcal{REC}$.

Beweis: Sei $L \in \mathcal{REC}$. Dann gibt es eine Turingmaschine M, die χ_L berechnet.

Abbildung 21 Skizze zum Beweis von Lemma 5



M'' (siehe Abbildung 21) berechnet $\chi_{\overline{L}}$.

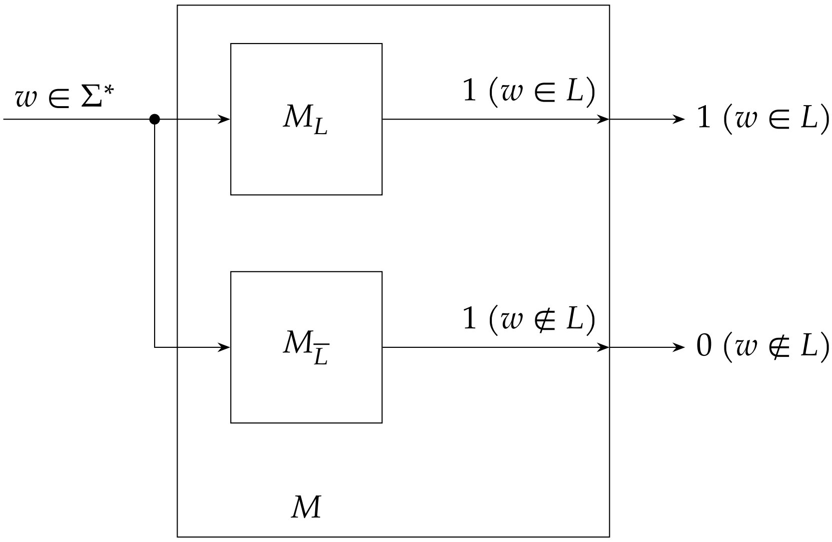
Satz 27 L ist Turing-entscheidbar genau dann, wenn L und $\overline{L}$ Turing-semientscheidbar sind.

Beweis:

- $\Rightarrow$: Mit Lemma 5 ist $\overline{L}$ Turing-entscheidbar. Mit Lemma 4 sind L und $\overline{L}$ Turing-semientscheidbar.

- Sei $L \in \mathcal{RE}$, d. h. es gibt eine Turingmaschine M_L, die χ_L^p berechnet. Weiter sei $\overline{L} \in \mathcal{RE}$, d. h. es gibt eine Turingmaschine $M_{\overline{L}}$, die $\chi_{\overline{L}}^p$ berechnet. Wir konstruieren eine neue Turingmaschine M wie in Abbildung 22, die die Arbeit von M_L und $M_{\overline{L}}$ „parallel“ ausführt.
  – Die Zustände von M sind Paare („Produktautomat“).
  – M_L ist auf Band 1.
  – $M_{\overline{L}}$ ist auf Band 2.

Abbildung 22 Konstruktion einer Turingmaschine für M_L und $M_{\overline{L}}$



Das heißt, für jedes $w \in \Sigma^*$ gilt: entweder $w \in L$ oder $w \notin L$. Also terminiert eine der beiden Maschinen. Damit terminiert M bei jeder Eingabe. □

Definition 52

- Eine Sprache $L \subseteq \Sigma^*$ ist *Turing-aufzählbar* genau dann, wenn es eine total definierte und Turing-berechenbare Wortfunktion $f : \Sigma^* \twoheadrightarrow L$ von Σ^* auf L gibt. f heißt *aufzählende Funktion* für L (Turing-berechenbar und surjektiv).
- Eine Zahlenmenge $A \subseteq \mathbb{N}$ ist *Turing-aufzählbar* genau dann, wenn es eine total definierte und Turing-berechenbare Zahlenfunktion $g : \mathbb{N} \twoheadrightarrow A$ von $\mathbb{N}$ auf A gibt. g heißt *aufzählende Funktion* für A.
- Eien Sprache $L \subseteq \Sigma^*$ ist *Turing-aufzählbar* genau dann, wenn es eine totaldefinierte und Turing-berechenbare Funktion $h : \mathbb{N} \twoheadrightarrow L$ von $\mathbb{N}$ auf L gibt. h heißt *aufzählende Funktion* für L.

Bemerkung 101 Situation: $h(n) = w \in L$, wobei n die Platznummer von w ist. $w \in L$ hat „viele“ Platznummern. Im Allgemeinen ist h nicht injektiv.

Definition 53 Sei $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ eine Standard-Turingmaschine, die eine Funktion $f : \Sigma^* \to \Delta^*$ berechnet. Wir bezeichnen

- f als *Resultat* von M: $\mathrm{Res}_M = f$,
- die Menge $H_M = \{u \in \Sigma^* \mid \exists v \in \Delta^* : \mathrm{Res}_M(u) = v\}$ als *Haltebereich* von M (M terminiert bei Eingabe u) und
- die Menge $W_M = \{v \in \Delta^* \mid \exists u \in \Sigma^* : \mathrm{Res}_M(u) = v\}$ als *Wertebereich* von M (M berechnet die Ausgabe v).

Satz 28 Die folgenden Aussagen sind äquivalent:

1. L ist Turing-aufzählbar.
2. L ist Wertebereich einer Turingmaschine M_W.
3. L ist Haltebereich einer Turingmaschine M_H.
4. L ist Turing-semientscheidbar.

Bemerkung 102 Charakterisierung der Turing-Aufzählbarkeit. $\mathcal{RE}$ bezeichnet die Klasse der Turing-aufzählbaren Mengen.

Bemerkung 103 Sonderfall: die leere Menge $\emptyset$. Wir wissen, es gibt Turingmaschinen, die für keine Eingabe stoppen. Eine solche Turingmaschine M berechnet die nirgends definierte Funktion $\nu = \mathrm{Res}_M$. Weiter gilt für ein solches M: $H_M = \emptyset$, $W_M = \emptyset$ und $\mathrm{Res}_M = \chi^p_\emptyset$.

Deshalb ergänzen wir die Definition 52 der Turing-Aufzählbarkeit: Entweder es gibt eine aufzählende Funktion für L oder $L = \emptyset$.

Bemerkung 104 Beweis zu Satz 28 als Ringbeweis: $1 \to 2 \to 3 \to 4 \to 1$

- $1 \Rightarrow 2$:
  1. Fall (Sonderfall): $L = \emptyset$. L ist Ergebnisbereich einer geeigneten Turingmaschine.
  2. Fall: $L \neq \emptyset$. Es gibt eine aufzählende Funktion $h : \mathbb{N} \twoheadrightarrow L$. h werde von einer Turingmaschine M_h berechnet. Der Ergebnisbereich von M ist $W_M = L$.
- $2 \Rightarrow 3$:
  1. Fall (Sonderfall): $L = \emptyset$.
  2. Fall: $L \neq \emptyset$. Wir wissen, es gibt eine Turingmaschine M mit dem Ergebnisbereich $L \subseteq \Sigma^*$. Wir erzeugen Σ^* in kanonischer Ordnung[17]. Wir betrachten Wörter der Form $\mathrm{Konf}^i_M(w_0)\#\mathrm{Konf}^{i-1}_M(w_1)\# \ldots \# \mathrm{Konf}^{i-j}_M(w_j) \# \ldots \# \mathrm{Konf}^0_M(w_i)$.

[17] $\Sigma^* = \{w_0, w_1, w_2, \ldots\}$, wobei Σ b Buchstaben habe. Dann entspricht die kanonische Ordnung gerade der b-adischen Darstellung. Jedes Wort hat genau eine Platznummer: Platznummer i hat die b-adische Darstellung w_i.

- ⋆ Situation 1: Eine dieser Berechnungen ist terminiert und liefert ein Resultat u. Wir konstruieren eine Turingmaschine M', die bei Eingabe w derartige Wörter erzeugt. Die Initialisierung ist hierbei $\mathrm{Konf}_M^0(w_0)$, Startkonfiguration von M bei Eingabe w_0. Liegt Situation 1 vor?
  - ▷ Falls nein: Berechne für jede Konfiguration $\mathrm{Konf}_M^{i-j}(w_j)$ die unmittelbare Nachfolgekonfiguration $\mathrm{Konf}_M^{i-j+1}(w_j)$ und erzeuge das nächste Wort: $\mathrm{Konf}_M^{i+1}(w_0)\#\mathrm{Konf}_M^{i}(w_1)\#$ $\ldots \# \mathrm{Konf}_M^{i+j+1}(w_j) \# \ldots \# \mathrm{Konf}_M^{1}(w_i) \# \mathrm{Konf}_M^{0}(w_{i+1})$ (die Wörter werden von Runde zu Runde immer länger).
  - ▷ Falls ja: M' vergleicht das Resultat u mit der Eingabe w. Ist $u = w$?
    - ∘ Falls nein: M' erzeugt das „nächste" Wort.
    - ∘ Falls ja: Dann räumt M' auf und terminiert. $u = w$ bedeutet: w ist Ergebnisbereich von M und damit im Haltebereich von M' Damit gilt $H_{M'} \subseteq W_M$. Es sei jetzt $w \in W_M$. Dann gibt es eine Eingabe x und eine Taktzahl t mit $\mathrm{Res}_M(x) = w = \mathrm{Konf}_M^t(x)$. Die Platznummer von x sei k, das heißt $x = w_k$. Dann erzielt M' in Runde $t + k$ $w = \mathrm{Konf}_M^t(w_k)$ und stoppt. Damit ist $w \in H_{M'}$ und $W_M \subseteq H_{M'}$.

- $3 \Rightarrow 4$: L ist Haltebereich einer Turingmaschine M. Wir konstruieren eine Turingmaschine M', die im Wesentlichen wie M arbeitet, aber mit einer zusätzlichen Spur zur Markierung der von M benutzten Zellen.
  Wenn M für eine Eingabe w terminiert, dann terminiert auch M', räumt zuvor aber das Band auf und schreibt die Ausgabe 1. Wenn M für eine Eingabe w nicht terminiert, dann terminiert auch M' nicht. Also gilt: $L = H_M = H_{M'}$ und $\mathrm{Res}_{M'} = \chi_L^p$. Das heißt, L ist Turing-semientscheidbar.

Bemerkung 105 Bis zu diesem Punkt wissen wir im Beweis des Satzes 28 (Bemerkung 104), dass aus L ist Turing-aufzählbar folgt, L ist Turing-semientscheidbar.

Lemma 6 Cantor-*Nummerierung*:

1. Es gibt eine Bijektion $c : \mathbb{N} \times \mathbb{N} \mapsto\!\!\!\rightarrow \mathbb{N}$ mit den Umkehrabbildungen $l, r : \mathbb{N} \mapsto\!\!\!\rightarrow \mathbb{N}$.
2. Alle diese Funktionen c, l, r sind Turing-berechenbar.

Zum Beweis:

$$c(x,y) = \frac{(x+y)\cdot(x+y+1)}{2} + x$$

Wenn $c(x,y) = n$, dann $l(n) = x$ und $r(n) = y$.

Bemerkung 106 Fortsetzung des Beweises aus Bemerkung 104 (Beweis von Satz 28). $4 \Rightarrow 1$: Voraussetzung: L ist Turing-semi-entscheidbar. Das bedeutet, es gibt eine Turingmaschine M, die χ_L^p berechnet. Aufgabe: Definiere eine aufzählende Funktion für den Definitionsbereich von χ_L^p. Für $L \subseteq \Sigma^*$ sei $\Sigma^* = \{w_0, w_1, \ldots\}$ wieder die kanonische Ordnung. Eine Turingmaschine M', die eine aufzählende Funktion M zu berechnen versucht, hat bei der Simulation von M ein ernstes Problem. Für keine Eingabe w_K ist klar, ob M stoppt oder nicht.

1. Fall (Sonderfall): $L = \emptyset$.
2. Fall: $L \neq \emptyset$. Es sei $z_{\text{fix}} \in L$ ein beliebiges, aber festgehaltenes Element. Wir konstruieren jetzt eine Turingmaschine M', die die Cantor-Nummerierung nutzt (M' arbeitet nach der Dove-tailing-Methode). Es sei $n \in \mathbb{N}$ Eingabe für M'.
   1. Abschnitt der Arbeit von M': M' berechnet $k = l(n)$ und $t = r(n)$.
   2. Abschnitt der Arbeit von M': M' simuliert die Arbeit von M bei Eingabe w_k (k-tes Wort der kanonischen Ordnung) in t Takten. Das heißt, M' erzeugt $\text{Konf}_M^t(w_k)$.
      1. Fall: M terminiert bei Eingabe w_k innerhalb der t Takte. Das heißt $w_k \in H_M$. M liefert die Ausgabe 1, mit anderen Worten $w_k \in L$. Dann räumt M' auf und schreibt die Ausgabe w_k.
      2. Fall: M terminiert bei Eingabe w_k innerhalb der t Takte (noch) nicht. Dann räumt M' auf und schreibt die Ausgabe z_{fix}.

   Es gilt:
   1. M' berechnet eine total definierte Funktion h mit $D_h = \mathbb{N}$.
   2. Alle Ausgaben von M' gehören zu L, d. h. $h(\mathbb{N}) \subseteq L$.
   3. Es gilt auch $L \subseteq h(\mathbb{N})$. Damit ist h eine aufzählende Funktion für L. Das heißt, L ist Turing-aufzählbar.

   Sei $z \in L$. Dann gilt $\chi_L^p(z) = 1$. Das heißt, die Turingmaschine M erzeugt bei Eingabe z nach t Takten die Ausgabe 1. Sei $z = w_s$ die Platznummer von z. Wir setzen $n' = c(s,t)$. M' simuliert bei Eingabe n' die Arbeit von M bei Eingabe w_s in t Takten und M' erzeugt die Ausgabe $w_s = z$ gemäß Fall 1. Damit ist $z \in h(\mathbb{N})$, also $z = h(n')$.

Bemerkung 107 χ_L^p ist Turing-berechenbar. Dann konstruieren wir eine aufzählende Funktion h für $L \subseteq \Sigma^*$. Eingabe sei n.

1. Schritt: Bestimme $l(n) = k$ und $r(n) = t$ (Dove-Tail-Methode).

Σ^* | **Takte**	**0**	**1**	**2**	**3**
w_0	$\text{Konf}_M^0(w_0)$	$\text{Konf}_M^1(w_0)$	$\text{Konf}_M^2(w_0)$	$\text{Konf}_M^3(w_0)$
w_1	$\text{Konf}_M^0(w_1)$	$\text{Konf}_M^1(w_1)$	$\text{Konf}_M^2(w_1)$	$\text{Konf}_M^3(w_1)$
w_2	$\text{Konf}_M^0(w_2)$	$\text{Konf}_M^1(w_2)$	$\text{Konf}_M^2(w_2)$	$\text{Konf}_M^3(w_2)$
w_3	$\text{Konf}_M^0(w_3)$	$\text{Konf}_M^1(w_3)$	$\text{Konf}_M^2(w_3)$	$\text{Konf}_M^3(w_3)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

- 2. Fall: Die Berechnung von M bei Eingabe w_k terminiert.
- 1. Fall: Die Berechnung von M bei Eingabe w_k terminiert nicht. Dann wird w_k und $z_{\text{fix}} \in L$ ausgegeben.

$h : \mathbb{N} \mapsto\!\!\!\rightarrow L$

Satz 29 $L \subseteq \Sigma^*$ ist *Turing-aufzählbar* genau dann, wenn $L = \emptyset$ oder es gibt eine Turingmaschine (ohne Eingabe), die auf einem Ausgabeband aller Wörter aus L in irgendeiner Reihenfolge erzeugt (aufzählt).

Definition 54 Der *Graph* G_f einer Funktion ist definiert als

$$G_f = \{u \# v \mid u \in \Sigma^*, v \in \Delta^*, f(u) = v\}$$

Satz 30 Eine Funktion $f : \Sigma^* \to \Delta^*$ ist *Turing-berechenbar* genau dann, wenn der Graph G_f Turing-aufzählbar ist.

Beweis:

- $\Rightarrow$: f ist Turing-berechenbar, d. h. es gibt eine Turingmaschine M_f, die f berechnet. Der Haltebereich H_{M_f} dieser Turingmaschine ist der Definitionsbereich D_f. Wegen Satz 28 ist H_{M_f} Turing-aufzählbar. Also existiert eine aufzählende Funktion M für H_{M_f}, d. h. $h(\mathbb{N}) = H_{M_f}$. h wird durch eine Turingmaschine M_h berechnet. Wir konstruieren eine Turingmaschine M_G, die eine aufzählende Funktoin h_G für den Graphen G_f berechnet. Eingabe für M_G ist $n \in \mathbb{N}$.
  1. Teil: Simuliere M_h mit Eingabe n. Da h total definiert ist, gibt es ein Resultat $h(n) = u \in D_f$.
  2. Teil: Simuliere M_f mit Eingabe u. Da $u \in D_f$, gibt es ein Resultat $f(u) = v \in \Delta^*$.

  Ausgabe: M_G erzeugt $u \# v$. Korrektheit von M_G:
  - Alle Resultate $u \# v \in G_f$.
  - Jedes Element $u \# v \in G_f$ kommt als Resultat vor, denn h ist surjektiv, d. h. $\forall u \in D_f : \exists n \in \mathbb{N} : h(n) = u$.

- $\Leftarrow$: Beweis mit Hilfe von Satz 29. Nach Voraussetzung ist G_f Turing-aufzählbar. Da $G_f \neq \emptyset$, gibt es eine Turingmaschine M', die auf einem Ausgabeband alle Wörter aus G_f aufzählt. Wir konstruieren eine Turingmaschine, die f berechnet. Eingabe für M ist $u \in \Sigma^*$.
  M simuliert die Arbeit von M'. Wird ein (Zwischen-)Ergebnis $x \# y$ erreicht, dann prüft M, ob $x = u$.
  – falls ja: liefert M die Ausgabe $y = v$.
  – falls nein: erzeugt M das nächste (Zwischen-)Ergebnis.
  Korrektheit von M:
  1. Fall: $u \notin D_f$. Dann gibt es kein Wort $u\#v$ und M terminiert nicht.
  2. Fall: $u \in D_f$. Dann wird $u \# y$ irgendwann aufgezählt und die Ausgabe ist $f(u)$.

Satz 31 Eine Sprache $L \subseteq \Sigma^*$ ist *Turing-aufzählbar* genau dann, wenn L durch eine formale Grammatik erzeugt wird.

Bemerkung 108 Folgerung aus Satz 31: $\mathcal{RE} = \mathrm{CH}(0)$.

Beispiel 99 Es ist zu zeigen $L \subseteq \Sigma^*$ ist Turing-aufzählbar genau dann, wenn $L = \emptyset$ oder es gibt eine TM, die auf einem Ausgabeband alle Wörter $w \in L$ durch # in irgendeiner Reihenfolge erzeugt.

- $\Rightarrow$: L ist Turing-aufzählbar. Dann ist $L = \emptyset$ und wir sind fertig oder $L \neq \emptyset$. In letzterem Fall ist L Haltebereich einer Turingmaschine M. Da der Haltebereich eine Menge ist, ist die Reihenfolge beliebig, aber es werden alle Wörter von L erzeugt. Das Trennzeichen kann in der Berechnungsvorschrift von M verankert werden, indem vor dem Übergang zum Nachfolger in der aufzählenden Funktion ein # geschrieben und in die Aufzählungsrichtung gerückt wird.
- $\Leftarrow$: $L = \emptyset$ ist trivial. $L \neq \emptyset$ und es gibt eine Turingmaschine, die auf einem Ausgabeband alle Wörter $w \in L$ durch # in irgendeiner Reihenfolge erzeugt. Dann ist L Wertebereich einer Turingmaschine. Damit ist L Turing-aufzählbar. □

Beispiel 100 Es ist zu zeigen, $L \subseteq \Sigma^*$ ist Turing-entscheidbar genau dann, wenn es eine Turingmaschine gibt, die auf einem Ausgabeband alle Wörter $w \in L$ durch # in kanonischer Reihenfolge erzeugt.

- $\Rightarrow$: L ist Turing-entscheidbar. Wir wissen, dass Σ^* von einer Turingmaschine M in kanonischer Ordnung konstruiert werden kann. Wir erweitern diese Turingmaschine darum, dass nach Erzeugen eines Wortes χ_L auf 1-Ausgabe getestet wird und bei negativem Ausgang das Wort wieder gelöscht wird. Vor Schreiben des nächsten Wortes wird eine

geschrieben und nach rechts gerückt. Da diese keine Änderung an der Reihenfolge vornimmt, werden alle Wörter von L in kanonischer Reihenfolge erzeugt.

- $\Leftarrow$: Gegeben sei eine Turingmaschine M, die auf einem Ausgabeband alle Wörter $w \in L$ durch # in kanonischer Reihenfolge erzeugt. Dann lässt sich eine Turingmaschine M' konstruieren, die die charakteristische Funktion χ_L für ein Wort w berechnet, indem das Band durchgelaufen wird. Dabei gibt es für das auf dem Band gelesene zu vergleichende Wort w' drei Fälle (p bezeichnet die Platznummer eines Wortes, die in der kanonischen Ordnung einfach ermittelbar ist):
  - $p(w') < p(w)$: Suche das nächste Wort.
  - $p(w') = p(w)$: Schreibe 1 und lösche den Rest des Bandes.
  - $p(w') > p(w)$: w hätte bereits gefunden werden müssen, wurde dies aber nicht; daher ist es kein Wort der Sprache. Schreibe 0 und lösche den Rest des Bandes.

  Da M' die charakteristische Funktion berechnen kann, die total definiert ist, ist L Turing-entscheidbar. □

Beispiel 101 Seien L_1 und L_2 zwei Turing-aufzählbare Sprachen. Es ist zu zeigen, dass $L_1 \cap L_2$ Turing-aufzählbar ist.

L_1 und L_2 sind Turing-semi-entscheidbar. Wir können eine Funktion

$$\chi_L^p = \begin{cases} 1 & \chi_{L_1}^p = \chi_{L_2}^p = 1 \\ \perp & \text{sonst} \end{cases}$$

definieren, die eine partiell charakteristische Funktion einer Turing-semi-entscheidbaren Maschine M ist, da sie als Wortfunktion von einer Turingmaschine berechnet werden kann, weil sowohl χ_{L_1} als auch χ_{L_2} als auch der Vergleich zweier Buchstaben (der Test, ob beides 1 ist) von einer Turingmaschine berechnet werden können (wobei anzumerken ist, dass das Verhalten nicht definiert ist, sobald eine der beiden zu untersuchenden partiell charakteristischen Funktionen nicht 1 ist). Diese Funktion bildet die partiell charakteristische Funktion für $L_1 \cap L_2$ ab und damit ist dies Turing-semi-entscheidbar, also Turing-aufzählbar. □

Beispiel 102 Seien L_1 und L_2 zwei Turing-entscheidbare Sprachen. Es ist zu zeigen, dass $L_1 \cap L_2$ Turing-entscheidbar ist.

L_1 und L_2 sind Turing-entscheidbar. Wir können eine Funktion

$$\chi_L^p = \begin{cases} 1 & \chi_{L_1}^p = \chi_{L_2}^p = 1 \\ 0 & \text{sonst} \end{cases}$$

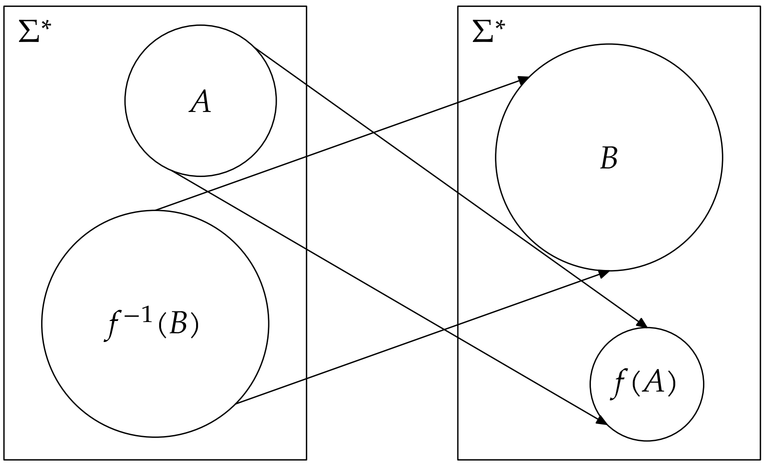
definieren, die eine partiell charakteristische Funktion einer Turing-entscheidbaren Maschine M ist, da sie als Wortfunktion von einer Turingmaschine berechnet werden kann, weil sowohl χ_{L_1} als auch χ_{L_2} als auch der Vergleich zweier Buchstaben (der Test, ob beides 1 ist) von einer Turingmaschine berechnet werden können. Diese Funktion bildet die partiell charakteristische Funktion für $L_1 \cap L_2$ ab und damit ist dies Turing-entscheidbar. □

Beispiel 103 Seien L_1 und L_2 zwei Turing-entscheidbare Sprachen. Es ist zu zeigen, dass $L_1 \cup L_2$ Turing-entscheidbar ist.

Sowohl L_1 als auch L_2 Turing-entscheidbar ist. Nach Beispiel 102 ist $\overline{L_1} \cap \overline{L_2}$ Turing-entscheidbar. Aus der Vorlesung wissen wir, dass $L \in \mathrm{REC} \implies \overline{L} \in \mathrm{REC}$, also in diesem Fall ist wegen der Turing-Entscheidbarkeit von $\overline{L_1} \cap \overline{L_2}$ auch $L_1 \cup L_2$ Turing-entscheidbar. □

Bemerkung 109 Sei $A \subseteq \Sigma^*$ und $B \subseteq \Sigma^*$. Das Bild von A bezüglich f sei $f(A) = \{v \in \Sigma^* \mid \exists u \in A : f(u) = v\}$. Das Urbild von B bezüglich f sei $f^{-1}(B) = \{u \in \Sigma^* \mid \exists v \in B : f(u) = v\}$. Dies ist schematisch in Abbildung 23 dargestellt.

Abbildung 23 Bild und Urbild



Beispiel 104 Sei $f : \Sigma^* \to \Sigma^*$ Turing-berechenbar und $L \subseteq \Sigma^*$ Turing-aufzählbar. Es ist zu zeigen, dass $f^{-1}(L)$ Turing-aufzählbar ist.

Wir zeigen, dass zu $f^{-1}(L)$ eine aufzählende Funktion $h : \mathbb{N} \mapsto\!\!\!\!\rightarrow f^{-1}(L)$ existiert. Wir wissen, L ist Turing-aufzählbar, d. h. es existiert eine aufzählende Funktion $g : \mathbb{N} \mapsto\!\!\!\!\rightarrow L$. Wir haben zwei Turingmaschinen M_f und M_g, die f bzw. g berechnen. Wir konstruieren eine Turingmaschine M', die h aus der Eingabe u berechnet.

1. Etappe: Berechne $f(u)$ mit der Turingmaschine M_f.

1. Fall: $u \notin D_f$. Es existiert kein $v \in \Sigma^*$ mit $f(u) = v$. Das heißt, $u \notin f^{-1}(L)$. Das heißt, M' terminiert nicht, da M_f nicht terminiert.
2. Fall: $u \in D_f$. Das heißt, M_f terminiert und liefert $f(u)$. Dann gehe über in die 2. Etappe.

2. Etappe: Teste, ob $f(u) \in L$. Dazu nutzen wir eine Turingmaschine, die die partiell-charakteristische Funktion χ_L^p berechnet.

Im Ergebnis erhalten wir $\chi_{f^{-1}(L)}^p = f \circ \chi_L^p$.

Beispiel 105 Wenn $L \subseteq \Sigma^*$ Turing-entscheidbar und eine total-definierte Funktion f Turing-berechenbar ist, dann ist auch $f^{-1}(L)$ Turing-entscheidbar.

Zwei Argumentationswege:

- Wir wisssen, es gibt eine Turingmaschine M_f, die für jedes $u \in \Sigma^*$ terminiert, und eine Turingmaschine M_χ, die die charakteristische Funktion $\chi_L(v) = \begin{cases} 1 & \text{falls } v \in L \\ 0 & \text{falls } v \notin L \end{cases}$ für $v \in \Sigma^*$ berechnet. Es ist zu zeigen, dass die charakteristische Funktion

$$\chi_{f^{-1}(L)}(u) = \begin{cases} 1 & \text{falls } u \in f^{-1}(L) \\ 0 & \text{falls } u \notin f^{-1}(L) \end{cases} = \begin{cases} 1 & \text{falls } f(u) \in L \\ 0 & \text{falls } f(u) \notin L \end{cases}$$

  für alle $u \in \Sigma^*$ Turing-berechenbar ist. Wir konstruieren die Turingmaschine M', die $\chi_{f^{-1}(L)}$ aus einer Eingabe $u \in \Sigma^*$ berechnet:
  1. Etappe: Berechne $f(u)$ mit der Turingmaschine M_f.
  2. Etappe: Wende darauf M_χ an und berechne $\chi_L(f(u))$. Gib das Resultat aus.

  Das heißt, M' berechnet $\chi_{f^{-1}(L)} = f \circ \chi_L$.
- Wir wissen, dass L Turing-entscheidbar ist, genau dann, wenn L und $\overline{L}$ Turing-semi-entscheidbar sind. Nach Beispiel 104 wissen wir, dass auch $f^{-1}(L)$ und $f^{-1}(\overline{L})$ Turing-semi-entscheidbar sind (L und $\overline{L}$ sind Turing-aufzählbar, f ist Turing-berechenbar). Angenommen $f^{-1}(L)$ wäre nicht Turing-entscheidbar. Dann kann, da $f^{-1}(L)$ Turing-semi-entscheidbar ist, $\overline{f^{-1}(L)}$ nicht Turing-semi-entscheidbar sein. Es ist bekannt, dass $\overline{f^{-1}(L)} = f^{-1}(\overline{L})$[18]. Daher ist auch $\overline{f^{-1}(L)}$ nicht nicht Turing-semi-entscheidbar. Widerspruch.

[18] Auch wenn man davon ausgehen kann, dass bekannt ist, dass das Komplement des Urbilds gleich dem Urbild des Komplements ist, hier ein Beweis: $x \in f^{-1}(\overline{L}) \iff f(x) \notin L \iff \text{nicht } f(x) \in L \iff \text{nicht } x \in f^{-1}(L) \iff x \notin f^{-1}(L) \iff x \in \overline{f^{-1}(L)}$.

4.4 Gödelisierung von Turingmaschinen

Codierung von Standard-Turingmaschinen

Bemerkung 110 Sei $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ eine Standard-Turingmaschine. Wir vereinbaren

- $|Q| = n$ und es sei $Q = \{0, 1, 2, \dots, n-1\}$ mit $q_0 = 0$ und $F = \{1\}$.
- Weiter sei $|\Gamma| = k$, $\Gamma = \{a_0, a_1, \dots, a_{k-1}\}$ mit $a_0 = \square$ und
- $\Sigma \subseteq \Gamma$ mit $\Sigma = \{a_{i_1}, a_{i_2}, \dots, a_{i_l}\}$.

Die Überführungsfunktion $\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, N, R\}$ ist für jedes Paar $(q, a) \in Q \times \Gamma$ ein 5-Tupel $(q, a, \delta(q, a))$ und kann in einer $n \times k$-Matrix dargestellt werden.

Definition 55 Als *Standardcodierung* einer Standard-Turingmaschine M bezeichnen wir folgendes Codewort

$$
\begin{aligned}
w_M = n; a_0, a_1, a_2, \dots, a_{k-1}; a_{i_1}, a_{i_2}, \dots, a_{i_l};; \\
&(0, a_0, \delta(0, a_0)), (0, a_1, \delta(0, a_1)), \dots, (0, a_{k-1}, \delta(0, a_{k-1})); \\
&(1, a_0, \delta(1, a_0)), (1, a_1, \delta(1, a_1)), \dots, (1, a_{k-1}, \delta(1, a_{k-1})) \\
&\quad \vdots \\
&(n-1, a_0, \delta(n-1, a_0)), \dots, (n-1, a_{k-1}, \delta(n-1, a_{k-1}));;
\end{aligned}
$$

Bemerkung 111 w_M beschreibt M vollständig. Mit anderen Worten erlaubt w_M eine Simulation von M. w_M ist ein Wort über dem Alphabet $A = \{0, 1, \dots, n-1\} \cup \{a_0, a_1, \dots, a_{k-1}\} \cup \{,, ;, (,)\}$[19].

Trotzdem: Es gibt einen Algorithmus, der für ein Wort über A entscheidet, ob es Standardcodierung einer Turingmaschine M ist oder nicht.

Bemerkung 112 Wir betrachten das (universelle) Alphabet $B = \{0, 1\}$ und folgenden Homomorphismus $h : A^* \mapsto B^*$. Dazu schreiben wir $A = \{b_1, b_2, \dots, b_m\}$ (wir nummerieren die Buchstaben von A) und definieren $h(\lambda) = \lambda$, $h(b_i) = 01^i$ und $h(uv) = h(u)h(v)$.

Definition 56 Als *binäre Standardcodierung* einer Standard-Turingmaschine M bezeichnen wir folgendes Codewort

$$b_{w_M} = h(w_M) \in \{0, 1\}^*$$

[19] A hängt von M ab (mit anderen Worten: jedes M hat sein eigenes A).

Bemerkung 113 Fakt: Es gibt einen Algorithmus, der entscheidet, ob ein Wort $w \in \{0,1\}^*$ binäre Standardcodierung einer Turingmaschine M ist oder nicht.

Churchsche These: Also gibt es eine Turingmaschine, die entscheidet, ob eine Eingabe $w \in \{0,1\}^*$ binäre Standardcodierung einer Standardturingmaschine ist. Also gibt es eine Turingmaschine, die auf einem Ausgabeband alle Codewörter $b_{w_0}, b_{w_1}, b_{w_2}, \ldots$in kanonischer Reihenfolge erzeugt (auflistet).

Definition 57 Es sei M_n die Turingmaschine mit der binären Standardcodierung b_{w_n}. Die Zahl n heißt *Gödelnummer*[20] von M.

Bemerkung 114 Wir erhalten eine effektive Nummerierung aller Standard-Turingmaschinen $M_0, M_1, \ldots$.

Bemerkung 115 Alternative Formulierung zur letzten Vorlesung: Für Standard-Turingmaschinen M haben wir eine binäre Standardcodierung definiert: $M \rightsquigarrow \langle M \rangle = b_{W_M}$. Dabei ist $h : A_M^* \mapsto B^*$ mit $A = \{b_1, b_2, \ldots, b_m\}$ und $h(b_i) = 01^i$. Es gibt einen Algorithmus, der bei Eingabe n alle Wörter aus B^* in kanonischer Reihenfolge erzeugt und dabei alle Codewörter bis zum n-ten aussortiert mit der Ausgabe $b_{w_0} \# b_{w_1} \# \ldots \# b_{w_n}$. Auf diese Weise erhalten wir eine effektive Nummerierung aller (Standard-)Turingmaschinen $(M_n)_{n\in\mathbb{N}} = (M_0, M_1, \ldots)$. Den Index n von M_n heißt *Gödelnummer*. Die effektive Nummerierung $(M_n)_{n\in\mathbb{N}}$ heißt *Gödelisierung* der Turingmaschine.

Bemerkung 116 Res_M bezeichnet die Funktion f, die durch M berechnet wird. Damit gilt $(\mathrm{Res}_{M_n})_{n\in\mathbb{N}}$ ist eine effektive Nummerierung aller Turing-berechenbaren Funktionen, also eine effektive Nummerierung der Klasse $\mathcal{TM}$.

$(H_{M_n})_{n\in\mathbb{N}}$ und $(W_{M_n})_{n\in\mathbb{N}}$ sind effektive Nummerierungen aller Turing-aufzählbaren Mengen, also der Klasse $\mathcal{RE}$.

Bemerkung 117 Jede derartige Codierung von Turingmaschinen liefert eine spezielle effektive Nummerierung.

Bemerkung 118 Eine Konsequenz: Die Kardinalität der algorithmisch beherrschbaren Probleme (die Klasse $\mathcal{RE}$) ist abzählbar. $\mathcal{RE} \subseteq \mathcal{P}(\Sigma^*)$, mit anderen Worten, $\mathcal{RE}$ ist ein „verschwindender Bruchteil“ von $\mathcal{P}(\Sigma^*)$.

[20] Benannt nach dem deutschen Mathematiker Kurt Gödel.

Das Halteproblem

Definition 58 Wir definieren das *allgemeine Halteproblem* als folgende Sprache

$$H = \{b_{w_M}00u \mid u \in \{0,1\}^*\}$$

Dabei ist

- $b_{w_M} = \langle M \rangle$ die binäre Standardcodierung einer Turingmaschine M,
- 00 ein Trennsymbol,
- u eine Eingabe für M und
- M terminiert bei Eingabe u.

Satz 32 *Anhalte-Satz*:

1. H ist Turing-aufzählbar.
2. H ist nicht Turing-entscheidbar.
3. $\overline{H}$ ist nicht Turing-aufzählbar.

Beweis:

1. $H \subseteq \{0,1\}^*$. Wir beschreiben den folgenden Algorithmus mit $w \in \{0,1\}^*$.
   1. Etappe: Überprüfe, ob die Eingabe die Form $w = v00u$ hat, wobei $v = b_{w_M} = b_{w_n}$.
      1. Fall: Nein. Dann geht der Algorithmus in eine unendliche Schleife.
      2. Fall: Ja. Dann starte die 2. Etappe.
   2. Etappe:
      - ⋆ Erzeuge die Start-Konfiguration Start-$K_M(u) = \mathrm{Konf}^0_M(u)$.
      - ⋆ Simuliere die Arbeit von M Takt für Takt, d. h. $\mathrm{Konf}^i_M(u) \vdash \mathrm{Konf}^{i+1}_M(u) \vdash \ldots$.

      Daraus ergeben sich zwei Fälle:
      1. Fall: M terminiert bei der Eingabe u nicht. Dann terminiert der Algorithmus ebenfalls nicht.
      2. Fall: M terminiert bei der Eingabe u. Der Algorithmus räumt auf und erzeugt die Ausgabe 1.

   Die Churchsche These besagt: Es gibt eine Turingmaschine U, die diesen Algorithmus realisiert. Für diese Turingmaschine gilt:
   1. Der Haltebereich von U ist H.
   2. U berechnet die χ^p_H.

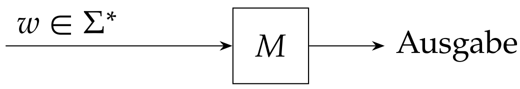
Damit ist H Turing-semientscheidbar und damit Turing-aufzählbar.

2. Indirekter Beweis durch Diagonalisierung: Angenommen H ist Turing-entscheidbar. Wir definieren $H' = \{b_{w_M} \mid b_{w_M}00b_{w_M} \notin H\}$. Da H Turing-entscheidbar ist, ist H' ebenfalls Turing-entscheidbar[21]. Da H' Turing-entscheidbar ist, ist H' auch Turing-aufzählbar. Also ist H' Haltebereich einer Turingmaschine (Satz 28). Diese Turingmaschine sei M mit $\langle M \rangle = b_{w_M}$.
   Wir erzeugen einen Widerspruch durch folgende Frage: Ist $b_{w_M} \in H'$ (Selbstbezüglichkeit)?
   1. Fall: Angenommen $b_{w_M} \in H'$. Dann gehört b_{w_M} zum Haltebereich von M. Dann gilt $b_{w_M}00b_{w_M} \in H$. Damit ist $b_{w_M} \notin H'$. Widerspruch.
   2. Fall: Angenommen $b_{w_M} \notin H'$. Dann gehört b_{w_M} nicht zum Haltebereich von M. Das heißt $b_{w_M}00b_{w_M} \notin H$. Aber damit ist $b_{w_M} \in H'$. Widerspruch.

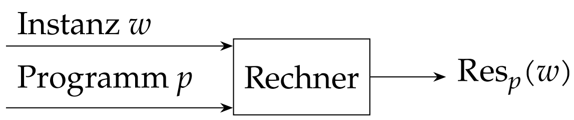
   Folglich ist H nicht Turing-entscheidbar.
3. Indirekter Beweis: Angenommen $\overline{H}$ ist Turing-aufzählbar. Dann ist $\overline{H}$ Turing-semientscheidbar (Satz 28). Da H Turing-aufzählbar (Eigenschaft 1) ist, ist H ebenfalls Turing-semientscheidbar. Nach Satz 27 ist H somit Turing-entscheidbar. Widerspruch zu Eigenschaft 2.

Bemerkung 119 Konsequenzen aus dem Anhalte-Satz (Satz 32):

1. Hierarchie der Sprachklassen: $\mathcal{REC} \subsetneq \mathcal{RE} \subsetneq \mathcal{P}(\Sigma^*)$. Die Echtheit der ersten Teilmenge wird durch H, die der zweiten durch $\overline{H}$ bezeugt.
2. Universelle Turingmaschinen: Ursprünglich waren Turingmaschine für spezielle Aufgaben konstruiert (z. B. Entscheider für Sprachen wie $L_3 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$).

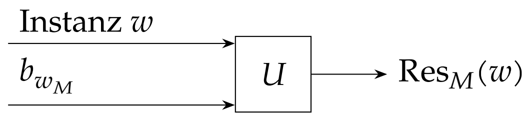
   

   Reale Rechner funktionieren anders:

   Eine *universelle Turingmaschine* U simuliert jede beliebige Turingmaschine.

[21] Für eine Eingabe b_{w_M} lässt sich $b_{w_M}00b_{w_M}$ erzeugen. Darauf wird der Entscheidungsalgorithmus für H angesetzt. Liefert dieser 1, dann wird 0 ausgegeben. Liefert dieser 0, wird 1 ausgegeben.



3. Das *spezielle Halteproblem* $H_{\text{spezi}} = \{b_{w_M}\}$ umfasst alle binären Standardcodierungen einer Turingmaschine M, die bei der Eingabe b_{w_M} terminiert.

Bemerkung 120 Was ist $\overline{H'}$? Wir bezeichnen $\text{BCode} = \{b_{w_M} \mid M \text{ ist TM}\}$. $\overline{H'} = \text{BCode} \setminus H' = \text{BCode} \setminus \{b_{w_M} \mid b_{w_M}00b_{w_M} \notin H\} = \{b_{w_M} \mid b_{w_M}00b_{w_M} \in H\} = H_{\text{spezi}}$ und es gilt H' ist nicht Turing-entscheidbar genau dann, wenn H_{spezi} nicht Turing-entscheidbar ist.

Satz 33

1. $H_{\text{spezi}} \in \mathcal{RE}$.
2. $H_{\text{spezi}} \notin \mathcal{REC}$.
3. $\overline{H}_{\text{spezi}} \notin \mathcal{RE}$.

Bemerkung 121 Zur Methode der Diagonalisierung: Mit derselben Methode beweist man indirekt, dass das reelle Intervall überabzählbar ist.

Die Diagonalisierung in „Reinkultur“ beim speziellen Halteproblem $H_{\text{spezi}} = \{b_{w_M} \mid b_{w_M}00b_{w_M} \in H\}$. Wir nehmen Aussage 2 aus Satz 33. Indirekter Beweis: Angenommen H_{spezi} ist Turing-entscheidbar. Dann gibt es eine Turingmaschine, die $\chi_{H_{\text{spezi}}}$ berechnet. Wir konstruieren eine neue Turingmaschine M', die wie folgt arbeitet: Für eine Eingabe b_{w_M}

- erzeugt M die Ausgabe 0 und M' stoppt oder
- erzeugt M die Ausgabe 1 und M' stoppt nicht.

Wir betrachten $b_{w_{M'}}$ in der Liste aller Codewörter mit der Frage $b_{w_{M'}} \in H_{\text{spezi}}$ (Selbstbezüglichkeit). $b_{w_{M'}} \in H_{\text{spezi}}$ genau dann, wenn M' bei Eingabe $b_{w_{M'}}$ stoppt, genau dann, wenn M bei Eingabe $b_{w_{M'}}$ die Ausgabe 0 erzeugt, genau dann, wenn $\chi_{H_{\text{spezi}}}(b_{w_{M'}}) = 0$ genau dann, wenn $b_{w_{M'}} \notin H_{\text{spezi}}$.

Definition 59 Sei $\emptyset \neq R \neq \mathcal{TM}$ eine nicht-triviale Teilmenge von Turing-berechenbaren Funktionen. Die *Indexmenge* von R ist definiert als $I(R) = \{n \in \mathbb{N} \mid \exists f \in R : \text{Res}_{M_n} = f\}$.

Satz 34 Satz von Rice: Für jede nicht-triviale Teilmenge R von Turing-berechenbaren Funktionen ist die Indexmenge $I(R)$ nicht Turing-entscheidbar.

Bemerkung 122 Das (allgemeine) Halteproblem hat zwei Unendlichkeitsdimensionen:

1. Liste der Codewörter: $b_{w_0}, b_{w_1}, \ldots$
2. Liste der Eingaben: $u_0, u_1, \ldots$

Die Nicht-Entscheidbarkeit bleibt mit einer Unendlichkeitsdimension erhalten:

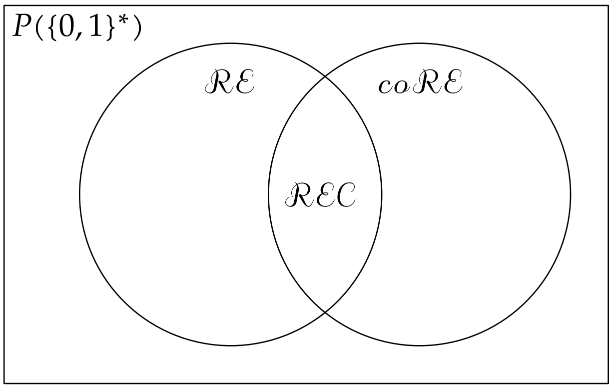
- Wir definieren $H_\lambda = \{b_{w_M} \mid M \text{ stoppt bei Eingabe } \lambda\}$. Es gilt: H_λ ist nicht Turing-entscheidbar.
- Es gibt eine Turingmaschine M mit H_M ist nicht Turing-entscheidbar. Wir wissen $(H_M)_M = (H_n)_{n\in\mathbb{N}} = \mathcal{RE}$ und es gibt Turing-aufzählbare Mengen, die nicht Turing-entscheidbar sind.

Definition 60 Eine Menge A ist *co-Turing-aufzählbar* genau dann, wenn $\overline{A}$ Turing-aufzählbar ist.

Beispiel 106 $\overline{H}$ ist co-Turing-aufzählbar.

Bemerkung 123 $co\mathcal{RE}$ bezeichnet die Klasse co-Turing-aufzählbarer Mengen.

Gibt es den Durchschnitt $\mathcal{RE} \cap co\mathcal{RE}$? $B \in \mathcal{RE} \cap co\mathcal{RE}$ genau dann, wenn B Turing-aufzählbar ist und $\overline{B}$ Turing-aufzählbar ist, genau dann, wenn B Turing-entscheidbar ist.



Bemerkung 124 Das Äquivalenzproblem $\text{Äqui} = \{b_{w_M}00b_{w_{M'}} \mid \text{Res}_M = \text{Res}_{M'}\}$ ist noch unentscheidbarer als das Halteproblem, denn es ist nicht Turing-entscheidbar und weder Äqui noch $\overline{\text{Äqui}}$ ist Turing-aufzählbar.

Bemerkung 125 Die Halteprobleme als Zahlenmengen (sei jeweils M eine Turingmaschine mit Gödelnummer n):

$$H_{\text{allg}} = \{(n,m) \mid M \text{ stoppt bei Eingabe des } m\text{-ten Wortes } u_m.\}$$
$$H_{\text{speziell}} = \{n \mid M \text{ stoppt bei Eingabe des } n\text{-ten Wortes } u_n.\}$$
$$H_{\square} = \{n \mid M \text{ stoppt bei Eingabe des leeren Bandes.}\}$$
$$H_n = \{m \mid M \text{ stoppt bei Eingabe von } u_m.\}$$

4.5 Reduzierbarkeit

Bemerkung 126 Wir beschränken uns in diesem Abschnitt auf die Betrachtung von Zahlenmengen:

- Gödelnummern von Turingmaschinen
- Nummer der Eingaben in kanonischer Ordnung

Definition 61 Seien $A, B \subseteq \mathbb{N}$ zwei Zahlenmengen. A ist *reduzierbar* auf B ($A \leq B$) genau dann, wenn es eine total-definierte und Turing-berechenbare Funktion $f : \mathbb{N} \mapsto \mathbb{N}$ mit der Eigenschaft $n \in A \iff f(n) \in B$ gibt.

Lemma 7

- Sei $A \leq B$ und $B \in \mathcal{REC}$, dann ist auch $A \in \mathcal{REC}$.
- Sei $A \leq B$ und $B \in \mathcal{RE}$, dann ist auch $A \in \mathcal{RE}$.

Beweis:

- Sei $A \leq B$, d. h. es gibt eine total-definierte Turing-berechenbare reduzierende Funktion f mit $x \in A \iff f(x) \in B$. $B \in \mathcal{REC}$ bedeutet χ_B ist Turing-berechenbar, das heißt, es gibt zwei Turingmaschinen M_f und M_χ, die f bzw. χ_B berechnen. Wir konstruieren M' als Komposition von M_f und M_χ. Die Eingabe ist $x \in \mathbb{N}$.
  1. Fall $x \in A$: $x \overset{M_f}{\mapsto} f(x) \overset{M_\chi}{\mapsto} \chi_B(f(x))$. Dann ist $f(x) \in B$ und das Resultat ist 1.
  2. Fall $x \notin A$: $x \mapsto f(x) \mapsto \chi_B(f(x))$. Dann ist $f(x) \notin B$ und das Resultat ist 0.

  M' berechnet χ_A als $\chi_A = f \circ \chi_B$.

Bemerkung 127 Folgerung aus Lemma 7:

- Sei $A \leq B$ und $A \notin \mathcal{REC}$, dann ist $B \notin \mathcal{REC}$.
- Sei $A \leq B$ und $A \notin \mathcal{RE}$, dann ist $B \notin \mathcal{RE}$.

Mit anderen Worten: Die Reduzierbarkeit „vererbt“

- die Turing-Entscheidbarkeit (bzw. Turing-Aufzählbarkeit) nach unten und
- die Nicht-Entscheidbarkeit (bzw. Nicht-Aufzählbarkeit) nach oben.

$A \leq B$ bedeutet „A ist nicht schwerer als B" und „B ist mindestens so schwer wie A".

Bemerkung 128 Eigenschaften der Reduzierbarkeit: $\leq$ ist eine binäre Relation mit folgenden Eigenschaften:

- $\leq$ ist reflexiv, d. h. stets gilt $A \leq A$.
- $\leq$ ist transitiv, d. h. $A \leq B \land B \leq C \implies A \leq C$.
- $\leq$ ist nicht antisymmetrisch, d. h. $H' \leq H_{\text{spezi}} \land H_{\text{spezi}} \leq H'$, aber $H' \neq H_{\text{spezi}}$.

Damit ist $\leq$ eine *Quasihalbordnung* und definiert eine Äquivalenzrelation $A \sim B \iff A \leq B \land B \leq A$.

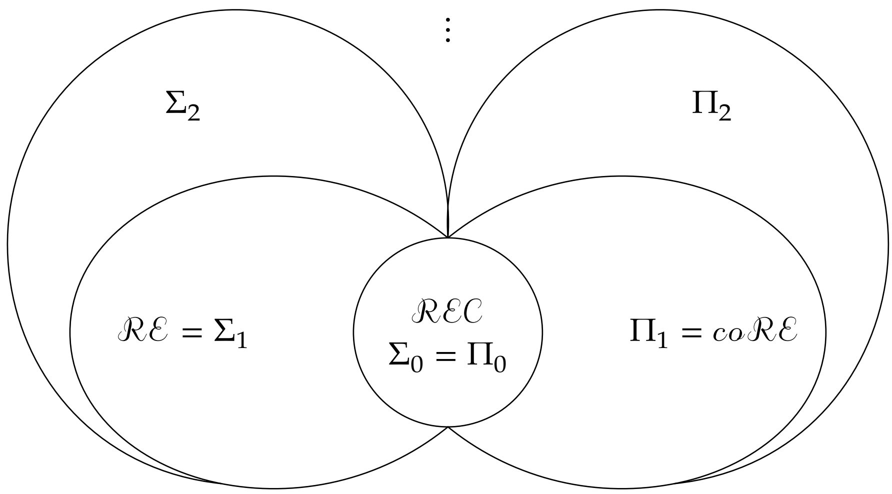
Bemerkung 129 Fakt: Es gibt eine unendliche Folge von Mengen $A_0, A_1, \ldots$ mit der Eigenschaft

- $A_0 \in \mathcal{REC}$,
- $A_i \leq A_{i+1}$ und
- $\neg(A_{i+1} \leq A_i)$.

Die letzten beiden Bedingungen besagen umgangssprachlich, A_{i+1} ist „echt schwerer" als A_i.

Beispiel: $A_0 = \text{Prim}$, $A_1 = H_{\text{spezi}}$, $A_2 = \text{Äqui}$.

Der Effekt dieses Faktes ist: Es gibt eine unendliche Folge von „Unentscheidbarkeitsgraden". Das ist die „Arithmetische Hierarchie" aus der „Rekursionstheorie".



Äqui liegt zum Beispiel in Σ_2.

Bemerkung 130 Das eindimensionale Halteproblem:

$$H_0 = \{n \mid \text{Turingmaschine } M_n \text{ terminiert auf dem leeren Band}\}$$

Wir zeigen $H \leq H_0$. Das bedeutet: H_0 ist nicht Turing-entscheidbar. Wir konstruieren eine Turingmaschine M mit Eingabe (n, m) wie folgt: M angesetzt auf das leere Band

- erzeugt das n-te Codewort $\langle M_n \rangle = w_{w_n}$,
- erzeugt das m-te Eingabe u_m und
- simuliert die Arbeit von M_n bei Eingabe u_m.

M ist eine Turingmaschine und hat eine Gödelnummer. Diese sei k, d. h. $\langle M \rangle = b_{w_k}$ und $M = M_k$. Wir definieren $f(n, m) = k$.

- f ist total definiert.
- Es gibt einen Algorithmus, der f berechnet, d. h. f ist Turing-berechenbar und es gilt

$$\begin{aligned}
(n, m) \in H_{\text{allg}} &\iff M_n \text{ terminiert bei Eingabe } u_m \\
&\iff M \text{ angesetzt auf das leere Band terminiert} \\
&\iff M_k \text{ angesetzt auf das leere Band terminiert} \\
&\iff M_{f(n,m)} \text{ angesetzt auf das leere Band terminiert} \\
&\iff f(n, m) \in H_0
\end{aligned}$$

Bemerkung 131 Beweis zum Satz von Rice (Satz 34): Sei $\emptyset \subsetneq R \subsetneq \mathcal{TM}$ und $I(R) = \{n \mid \exists \text{TM } M : \langle M \rangle = b_{w_n} \wedge \text{Res}_M \in R\}$, dann gilt $I(R)$ ist nicht Turing-entscheidbar.

1. Fall: Die nirgends definierte Funktion $\nu \in R$. Da $R \subsetneq \mathcal{TM}$ gibt es eine Turing-berechenbare Funktion $g \notin R$. M_g ist eine Turingmaschine, die g berechnet. Die Eingabe $n \in \mathbb{N}$: Wir konstruieren für jede solche Eingabe eine Turingmaschine M wie folgt: Für die Eingabe m arbeitet M,
   - indem es die Eingabe m ignoriert, aber
   - das n-te Codewort b_{w_n} erzeugt und
   - die Arbeit von M_n angesetzt auf das leere Band simuliert.
     1. Fall: M_n angesetzt auf das leere Band terminiert. Dann simuliert M die Arbeit von M_g bei Eingabe m.
     2. Fall: Sonst terminiert M für jede Eingabe m nicht.

Für die von M berechnete Funktion h gilt

$$h = \begin{cases} g & M_n \text{ angesetzt auf das leere Band terminiert} \\ \nu & \text{sonst} \end{cases}$$

Das heißt

$$h = \begin{cases} g & \text{falls } n \in H_0 \\ \nu & \text{sonst} \end{cases}$$

1. Fall: $n \in H_0$. Das heißt, M_n angesetzt auf das leere Band terminiert, das heißt M berechnet g. Sei $f(n) = \langle M \rangle = b_{w_{f(n)}}$. Das heißt, $\text{Res}_{M_{f(n)}} = g \notin R$, das heißt $f(n) \notin I(R)$.
2. Fall: $n \notin H_0$. Das heißt, M_n angesetzt auf das leere Band terminiert nicht. Das heißt M berechnet ν, das heißt $\text{Res}_{M_{f(n)}} = \nu \in R$, das heißt $f(n) \in I(R)$.

Zusammen: $n \in H_0 \iff f(n) \notin I(R)$, d. h. f ist reduzierende Funktion und es gilt $\overline{H_0} \leq I(R)$. Das heißt $I(R)$ ist nicht Turing-entscheidbar.

2. Fall: entsprechend (Rollen von ν und g „tauschen“). □

Bemerkung 132 Folgerung aus dem Satz von Rice (Satz 34): Sei

$$R = \{f \mid f : \mathbb{N} \to \mathbb{N}, f \text{ ist Turing-berechenbar}, D_f = \mathbb{N}\}$$

die Klasse der total-definierten und Turing-berechenbaren Zahlenfunktionen. Also ist $I(R)$ nicht Turing-entscheidbar. Das heißt, es gibt keinen Algorithmus, der bei Eingabe eines Turing-Programms P entscheidet, ob P für alle Eingaben terminiert.

Beispiel 107 Ein anderes Beispiel für Nicht-Entscheidbarkeit im richtigen Leben. Gegeben: Zwei kontextfreie Grammatiken G_1 und G_2 mit $L_1 = L(G_1)$ und $L_2 = L(G_2)$. Fragen:

- Ist $L_1 \cap L_2 = \emptyset$?
- Ist $L_1 \cap L_2$ endlich?
- Ist $L_1 \cap L_2$ regulär?

5 Einschübe und Exkurse

5.1 Ein Wörterbuch der Algorithmen

- ohne formale Definition
- trotzdem sehen wir Grenzen von Berechenbarkeit
- nicht der Formalismus ist entscheidend, sondern die Konstruktion

Algorithmus:

- intuitiv: Handlungsanweisungen
  - endlich beschreibbar
  - endliche Eingaben
  - endliche Menge von Grundoperationen
  - schrittweise Ausführung
  - Bestimmtheit
  - Terminierung
- Wir beschränken uns im Folgenden auf Handlungsanweisungen zur Berechnung von Zahlenfunktionen $f : \mathbb{N} \to \mathbb{N}$.

Beispiel 108 Gegeben sei eine natürliche Zahl n. Addiere zu dieser Zahl 1. Diese Zahl ist das Resultat.

Es wird die Nachfolgerfunktion $\operatorname{succ}(n) = n + 1$ für $n \in \mathbb{N}$ berechnet.

Beispiel 109 Gegeben sei eine natürliche Zahl n. Wenn $n = 0$ ist, dann ist das Resultat 1. Wenn $n = 1$, dann ist das Resultat 0. Anderenfalls wende die Vorschrift auf die Zahl $n - 2$ an.

Es wird die Paritätsfunktion

$$\operatorname{par}(n) = \begin{cases} 1 & \text{falls } n \text{ gerade} \\ 0 & \text{falls } n \text{ ungerade} \end{cases}$$

für $n \in \mathbb{N}$ berechnet.

Beispiel 110 Gegeben sei eine natürliche Zahl n. Setze $x = 1$. Wiederhole n-mal folgendes Verfahren: Setze $x = x + x$. Die zuletzt erhaltene Zahl ist das Resultat.

Es wird die Exponentialfunktion $\exp(n) = 2^n$ für $n \in \mathbb{N}$ berechnet.

Bemerkung 133 Ansatz: Wir erzeugen eine Liste aller derartigen Texte in kanonischer Ordnung. Das geht mittels Normierung der Texte. Es ist entscheidbar, ob ein beliebiger deutscher Text eine Vorschrift zur Berechnung einer Zahlenfunktion ist.

Auf diese Weise erhalten wir eine Liste aller Vorschriften, die Zahlenfunktionen berechnen: $(T_0, T_1, T_2, \ldots)$ (*ein* „Wörterbuch der Algorithmen").

Übereinkunft: Eine Zahlenfunktion $f : \mathbb{N} \to \mathbb{N}$ ist *text-berechenbar* genau dann, wenn es eine Vorschrift T in der Liste gibt, die diese Funktion f berechnet. Das ist äquivalent dazu, dass es eine Platznummer k gibt, sodass T_k eine Vorschrift zur Berechnung von f ist. Dann gilt $f = f_k$.

Auf diese Weise erhalten wir eine Liste $(f_0, f_2, \ldots)$ aller text-berechenbaren Funktionen.

Beispiel 111 Konstruktion: Gegeben sei eine natürliche Zahl n. Bestimme den Text T_n mit der Nummer n in der Liste. Wende diesen Text T_n auf die Eingabe n an. Addiere zu diesem Ergebnis 1. Die erhaltene Zahl ist das Resultat.

Welche Zahlenfunktion $g : \mathbb{N} \to \mathbb{N}$ wird berechnet? $g(n) = f(n) + 1$ für alle $n \in \mathbb{N}$ (Diagonalisierungsfunktion).

Bemerkung 134 Ist der Text aus Beispiel 111 gültig? Ja. Also kommt dieser Text in der Liste aller Texte vor. Damit hat er eine Platznummer. Diese sei k (also ist der Text T_k). Also gilt weiter $g = f_k$.

Frage: Welchen Wert hat die Funktion g an der Stelle k. Das ist die Frage nach der Selbstanwendung $g(k)$. Einserseits ist $g(k) = f_k(k) + 0$. Andererseits (nach Definition) ist $g(k) = f_k(k) + 1$. Die Selbstanwendung erzeugt einen Widerspruch.

Bemerkung 135 Bisher haben wir stillschweigend vorausgesetzt, dass alle text-berechenbaren Funktionen immer überall definiert sind, d. h. alle f_n sind total-definiert ($D_{f_n} = \mathbb{N}$). Dem ist aber nicht so.

Beispiele hierfür sind unendliche Schleifen oder nirgends definierte Funktionen (durch 0 teilen, Definitionsbereich leere Menge, ...).

Beispiel 112 Text, der nicht überall definiert ist: Es gibt ein x mit $D_{f_x} = \emptyset$. Stets ist $\emptyset \subseteq D_{f_y} \subseteq \mathbb{N}$ (partiell definiert).

Bemerkung 136 Auflösung des Widerspruchs aus Bemerkung 134: g ist an der Stelle k nicht definiert ($k \notin D_g$).

Bemerkung 137 Wir erzeugen eine Liste aller derartiger Texte in kanonischer Ordnung, die total-definierte Zahlenfolge $h : \mathbb{N} \mapsto \mathbb{N}$ berechnen. $(R_0, R_1, R_2, \ldots)$.

Dies liefert $(h_0, h_2, h_2, ...)$, die Liste aller total-definierten text-berechenbaren Zahlenfunktionen.

Bemerkung 138 Neue Diagonalisierungsfunktion: $d(n) = h(n) + 1$ für alle $n \in \mathbb{N}$ (mit anderen Worten $D_d = \mathbb{N}$).

d hat eine Platznummer l. Wir prüfen auf Widerspruch durch Selbstanwendung $(d(l))$. Einerseits $d = h_l$, also ist $d(l) = h_l(l)$. Andererseits (nach Definition) ist $d(l) = h_l(l) + 1$. Widerspruch.

Bemerkung 139 Auflösung des Widerspruchs aus Bemerkung 138: Eine solche Liste aller Texte zur Berechnung von total-definierten Zahlenfunktionen gibt es nicht.

Eine Liste aller Texte zur Berechnung von Zahlenfunktionen gibt es.

Ergebnis: Es gibt keinen Algorithmus, der für jeden beliebigen Text zur Berechnung einer Zahlenfunktion entscheidet, ob diese Zahlenfunktion total-definiert ist oder nicht.

Satz 35 Es ist algorithmisch nicht entscheidbar, ob eine berechenbare Funktion total definiert ist oder nicht.

5.2 Literaturhinweise

book: [author: Christel Baier and Alexander Asteroth] [isbn: 9783827370334] [publisher: Pearson Studium] [title: Theoretische Informatik] [year: 2002]

book: [author: Erk, Katrin and Priese, Lutz] [isbn: 9783540426240] [publisher: Springer] [title: Theoretische Informatik] [year: 2002]

book: [author: Hoffmann, Dirk] [isbn: 9783446445307] [publisher: Hanser] [title: Theoretische Informatik] [year: 2015]

book: [author: Hopcroft, John E. and Motwani, Rajeev and Ullman, Jeffrey D.] [isbn: 9783868940824] [publisher: Pearson Deutschland] [title: Einführung in die Automatentheorie, formale Sprachen und Berechenbarkeit] [year: 2011]

book: [author: Hromkovic, Juraj] [isbn: 9783834898531] [publisher: Vieweg+Teubner Verlag] [title: Theoretische Informatik: Formale Sprachen, Berechenbarkeit, Komplexitätstheorie, Algorithmik, Kommunikation und Kryptographie] [year: 2011] book: [author: Schöning, Uwe] [isbn: 9783827418241] [publisher: Spektrum Akademischer Verlag] [title: Theoretische Informatik - kurz gefasst] [year: 2008]

book: [author: Socher, Rolf] [isbn: 9783446412606] [publisher: Hanser] [title: Theoretische Grundlagen der Informatik] [year: 2008]

Index

a
Ableitungsschritt *46*
Alphabet *4, 10*
Anhalte-Satz *104*
A-Äquivalenz *22*
ablehnend *20*
ablehnende Finalkonfiguration *19*
ableitbar *46*
äquivalent *47*
akzeptierend *20*
akzeptierende Finalkonfiguration *19*
allgemeine Halteproblem *104*
aufzählende Funktion *93*

b
Berechnung *20, 75*
Binärdarstellung *13*
Boole *24, 26, 31*
Buchstabe *4*
binäre Nummer *12*
binäre Standardcodierung *102*

c
Cantor-Nummerierung *95*
Cantor *96*
Church *88, 103, 104, 117*
Cocke *63*
charakteristische Funktion *88*
co-Turing-aufzählbar *107*

d
Dyck-Sprache *38*
dyadische Darstellung *14*

e
Effizient *8*
endlicher Automat *16*
erweiterte Überführungsfunktion *20*
erzeugte Sprache *46*

f
Finalkonfiguration *75*
formale Grammatik *46*
formale Sprache *15*
freie Worthalbgruppe *11*

g
Gödel *103*
Gödelisierung *103*
Gödelnummer *103*
Graph *97*

h
Haltebereich *94*
Hauptsatz der Algorithmentheorie *88*

i
Index *40*
Indexmenge *106*

k
Kasani *63*
Knotenüberdeckung *90*
Konfiguration *18, 74*
Konkatenation *11*
Kontraposition des Pumping-Lemmas *38*
kanonische Ordnung *13*
kontextfreie Grammatik *55*
kontextsensitiv *52*

l
Länge *4, 11*
L-Äquivalenz *23*

linkslinear *71*

m
MYHILL *1, 40, 41, 43*
m-adische Darstellung *15*

n
Nachfolgekonfiguration *20*
NERODE *1, 40, 41, 43*
Nichtterminal *5*
Nichtterminale *46*
Normalform-Grammatik *47*
NP-vollständig *8*
nichtdeterministischer endlicher Automat *27*
normierte Finalkonfiguration *75*
n-stellige Wortfunktion *75*
n-stellige Zahlenfunktion *76*

p
k-te Potenz *12*
Potenzmengen-Automat *33*
Präfix *12*
Produktautomat *25*
Produktionen *46*
Pumping-Lemma *34*
Pumping-Lemma für kontextfreie Sprachen *60*
partiell-charakteristische Funktion *88*

q
quasilexikografische Ordnung *13*

r
Resultat *94*
RICE *106, 110*
reduzierbar *108*
regulär *21*
reguläre Operationen *31*
regulären Sprachen *21*

s
Semantik der regulären Ausdrücke *32*
Σ^* *10*
Spiegelgrammatik *72*
Spiegelwort *11*
Standardcodierung *102*
Standard-Turingmaschine *81*
Startkonfiguration *19, 75*
Startsymbol *46*
Suffix *12*
Syntax der regulären Ausdrücke *31*
spezielle Halteproblem *106*

t
Takt *19*
Teilwort *11*
Terminal *5*
Terminale *46*
These von CHURCH *88*
Turingmaschine *74*
Turing-aufzählbar *93, 97, 98*
Turing-berechenbar *76, 97*
Turing-entscheidbar *90*
Turing-semientscheidbar *91*
Typ-1-Grammatik *49*
Typ-1-Normalform-Grammatik *50*
Typ-2-Grammatik *55*
Typ-2-Normalform-Grammatik *56*
Typ-3-Grammatik *68*
Typ-3-Normalform-Grammatik *68*
text-berechenbar *113*
triadische Darstellung *15*

u
Überführungsrelation 74
Übergangsrelation 19
universelle Turingmaschine 105
unmittelbare Nachfolgekonfiguration 19, 75

w
Wertebereich 94
Wort 4, 10
Wortinduktion 26

y
YOUNGER 63

z
Zustandsfunktion 21
Zustandsklassen 21