

EINFÜHRUNG IN DIE THEORETISCHE INFORMATIK - SCRIPT

<https://github.com/C0d3Crush/ITH-Script>
Lukas.Dzielski@stud.uni-heidelberg.de



21. Juli 2023

Inhaltsverzeichnis

1	Grundlagen	5
1.1	Notationen und begriffe	5
1.2	Alphabet, Wörter und Sprachen	5
1.2.1	Definition (Alphabet)	6
1.2.2	Definition (Wörter)	6
1.2.3	Definition (Binäraphabet, Binärwörter)	6
1.2.4	Definitio (Sprache)	6
1.2.5	Definition(Wortmengen-Notation)*	6
1.2.6	Definition (Verkettung)	6
1.2.7	Definition (Präfix, Infix, Suffix)	6
1.2.8	Definition (präfixfrei)	7
1.2.9	Definition (Homomorphismus)	7
1.2.10	Definition (Längenlexikographische Ordnung)	7
1.2.11	Bemerkung(Endliche längenlexikographische Reihenfolge)*	7
1.2.12	Definition(Binärwort-Funktion in längenlexikographischer Reihenfolge)*	7
1.2.13	Bemerkung(Beziehung zwischen Binärworten und Dezimalzahlen)*	7
2	Turingmaschine	8
2.1	Definition (Turingmaschine, Alan Tuing, 1936)	9
2.2	Definition (Konfiguration)	10
2.3	Definition (Nachfolgekonfiguration)	10
2.4	Definition (Rechnung)	10
2.5	Bemerkung (Eindeutige Rechnungen bei k-DTM.)*	10
2.6	Definition (total)	10
2.7	Definition (akzeptierte Sprache)	10
2.8	Definition(rekursiv aufzählbar)	11
2.9	Bemerkung (Entscheidbare Sprache und rekursiv aufzählbare Sprache)*	11
2.10	Bemerkung (Endliche Sprachen sind entscheidbar.)*	11
2.11	Bemerkung (Entscheidbarkeit und Aufzählbarkeit.)*	11
2.12	Definition (Ausgabe)	11
2.13	Definition (berechnete Funktion)	11
2.14	Definition (partiell berechenbar)	12
2.15	Definition (charackteristische Funktion, partielle charakteristische Funktion)	12
2.16	Bemerkung (Entscheidbarkeit und Berechenbarkeit)*	12
2.17	Bemerkung (normiert)	13
2.18	Bemerkung (Entscheidbarkeit, Aufzählbarkeit, Berechenbarkeit.)*	14
3	Berechenbarkeit	15
3.1	Definition (Code)	16
3.2	Definition (standardaufzählung)	16
3.3	Bemerkung(Unendlich viele Indizes)*	16
3.4	Definition (U)	16
3.5	Definition (Universell)	17
3.6	Bemerkung(Universelle Turingmaschine U)*	17
3.7	Satz (s_n^m - Theorem)	17
3.8	Definition (diagonales Halteproblem)	18
3.9	Proposition(Diagonales Halteproblem)*	18
3.10	Satz(Diagonalhalteproblem - Nicht entscheidbar)*	18
3.11	Definition (m-Reduktion)	18
3.12	Bemerkung(Entscheidbarkeit unter Reduzibilität)*	18
3.13	Satz(Nicht-Entscheidbarkeit des initialen Halteproblems)*	19

3.14	Definition (Postisches Korrespondenzproblem, Emil Post, 1946)	19
3.15	Lemma(PCP-Reduktion: $\Gamma \rightarrow \Sigma$)*	20
3.16	Lemma(Reduktion von MPCP über Σ auf PCP über Σ)*	20
3.17	Lemma(Reduktion von H_{init} auf $MPCP_{\square,0,1,*,6,+}$)*	21
3.18	Beispiel(TM M_e mit $e \in \mathbb{N}_0$))*	22
3.19	Satz(Unentscheidbarkeit des Post'schen Korrespondenzproblems)*	22
3.20	Fixpunktsatz, Rekursionstheorem und Satz von Rice	22
3.20.1	Definition (Fixpunktsatz)	22
3.20.2	Satz (Fixpunktsatz, Hartley Rogers jr., 1967)	22
3.20.3	Satz (Rekursionstheorem, Stephen Cole Kleen, 1938)	23
3.20.4	Korollar(Existenz eines fixierten Punktes in der berechenbaren Funktionenfamilie)*	23
3.20.5	Definition (Indexmenge)	23
3.20.6	Satz (Satz von Rice, Henry Horden Rice, 1951)	23
4	Automaten	24
4.1	Definition (Endliche Automaten)	25
4.2	Definition (Übergangsfunktion eines EA)	25
4.3	Bemerkung (Eigenschaften von endlichen Automaten)*	26
4.4	Definition (Übergangsfunktion eines DEA)	26
4.5	Bemerkung (Folgerungen für DEA)*	26
4.6	Bemerkung(Eindeutigkeit endlicher Automaten)*	26
4.7	Definition (akzeptierte Sprache)	26
4.8	Definition (regulär)	26
4.9	Beispiel (Endlicher Automat A)*	27
4.10	Definition (Potenzautomaten)	28
4.11	Satz(Charakterisierung regulärer Sprachen)*	28
5	Reguläre Sprachen	29
5.1	Definition (Äquivalenzrelation)	30
5.2	Definition (A-Äquivalenz)	30
5.3	Bemerkung	30
5.4	Definition (Rechtskongruenz)	30
5.5	Proposition	30
5.6	Definition(DEA-Konstruktion für Äquivalenzklassen)	31
5.7	Lemma	31
5.8	Satz	31
5.9	Korollar	31
5.10	Definition(erreichbar)	31
5.11	Definition(isomorph)	32
5.12	Satz	32
5.13	Definition (L-Äquivalenz)	32
5.14	Bemerkung	32
5.15	Definition (Partion)	33
5.16	Definition (Verfeinerung)	33
5.17	Bemerkung	33
5.18	Proposition	33
5.19	Definition (Minimalautomat)	33
5.20	Satz(Charakterisierung regulärer Sprachen)*	33
5.21	Satz (Satz von Myhill und Nerode)	34
5.22	Satz (Pumping-Lemma)	34

6	Formale Grammatiken	36
6.1	Definition (Grammatiken)	37
6.2	Definition (Ableitung)	37
6.3	Definition (Erzeugte Sprache)	37
6.4	Lemma	38
6.5	Satz	38
6.6	Definition (Rechtslinear)	38
6.7	Satz	38
6.8	Beispiel	39
6.9	Definition (kontextfrei)	39
6.10	Definition (lexikographische Ordnung)	39
6.11	Definition (Baum)	41
6.12	Lemma	41
6.13	Lemma	42
6.14	Definition (Beschriftung)	42
6.15	Definition (Blattwort)	43
6.16	Definition (Ableitungsbaum)	44
6.17	Beispiel	44
6.18	Lemma	44
6.19	Lemma	45
6.20	Satz (Pumping-Lemma für kontextfreie Sprachen)	46
6.21	Beispiel	47
6.22	Beschreibung (Kellerautomat)	47
6.23	Satz	47
6.24	Beispiel	47
6.25	Definition (Kontextsensitiv)	47
6.26	Definition (nicht verkürzend)	48
6.27	Satz	48
6.28	Beschreibung (linear beschränkter Automat)	48
6.29	Satz	48
6.30	Beispiel	48
6.31	Definition	48
6.32	Satz	49
6.33	Beispiel	49
7	Zeitkomplexität	50
7.1	Definition (Rechenzeit)	51
7.2	Definition (zeitbeschränkt)	51
7.3	Satz (lineare Beschleunigung)	51
7.4	Satz (Alphabetwechsel)	52
7.5	Definition(Landau-Symbole)*	52
7.6	Bemerkung(Eigenschaften Landau-Symbole)*	52
7.7	Bemerkung(Eigenschaften Landau-Symbole)*	52
7.8	Proposition(Binärpalindrome erkennen)*	52
7.9	Satz (Zeitbeschränkung einer Turingmaschine)*	53
7.10	Satz(Reduktion auf 1-Band-TM)*	53
7.11	Satz(Codekonstruktion und Simulation)*	54
7.12	Satz (Universelle TM-Simulation mit Zeitbeschränkung)*	56
7.13	Definition (Zeitbeschränkte Funktionen und Sprachen)*	56
7.14	Definition (Zeitkonsturierbar)	56
7.15	Bemerkung (Polynomielle Zeitkonstruierbarkeit)*	56
7.16	Satz (Zeithirarchiesatz für deterministische TM)	57
7.17	Proposition(Zeitbeschränkte TM-Äquivalenz)*	57
7.18	Satz (Zeithirarchiesatz für nichtdeterministische TM)	57

7.19	Definition(abstraktes Komplexitätsmaß)	58
7.20	Definition(T-Beschränkte TM und DCOM)*	58
7.21	Satz (Lückensatz)	58
8	Komplexitätsklassen P und NP	59
8.1	Definition(Komplexitätsklassen P, FP und NP)*	60
8.2	Definition (p-m-Reduktion)	60
8.3	Bemerkung(Eigenschaften der p-m-Reduktion)*	60
8.4	Definition (NP -schwer, NP -vollständig)	60
8.5	Definition (aussagenlogische Formel)	60
8.6	Definition(Grundbegriffe der aussagenlogischen Formeln)*	60
8.7	Definition (konjunktive Normalform)	61
8.8	Definition (Wahrheitswert)	61
8.9	Definition (logisch äquivalent)	61
8.10	Definition (SAT)	61
8.11	Bemerkung (Das Erfüllbarkeitsproblem SAT in der Komplexitätsklasse NP)*	61
8.12	Satz (Satz von Cock)	62
8.13	Definition (k-konjunktive Normalform)	64
8.14	Definition (k-SAT)	64
8.15	Bemerkung(k-SAT als Entscheidungsproblem in der Komplexitätsklasse NP)*	64
8.16	Satz (NP-Vollständigkeit von k-SAT für $k \leq 3$)*	64
8.17	Definition (Graphen (V, E))*	64
8.18	Definition (k-Färbung)	64
8.19	Definition (k-COLORING)	65
8.20	Bemerkung(NP-Eigenschaft von COLORING und k-COLORING)*	65
8.21	Satz(NP-Vollständigkeit von 3-Coloring)*	65
8.22	Korollar (k-COLORING NP-vollständig für $k \leq 3$)*	65
8.23	Korollar(COLORING NP-vollständig)*	65
8.24	Definition (EXACTCOVER)	65
8.25	Bemerkung (EXACTCOVER NP Eigenschaft)*	65
8.26	Satz (EXACTCOVER NP-vollständig)*	65
8.27	Definition (SUBSETSUM)	65
8.28	Bemerkung(SUBSETSUM NP Eigenschaft)*	65
8.29	Satz(SUBSETSUM NP-vollständig)*	66
9	Platzkomplexität	67
9.1	Definition (TM vom Offline-Typ)	68
9.2	Definition(Platzbedarf)	68
9.3	Definition (platzbeschränkt)	68
9.4	Satz(lineare Kompression)	68
9.5	Satz(Alphabetwechsel)	68
9.6	Satz(Abstraktes Komplexitätsmaß)*	68
9.7	Satz (Spurentechnik und DTM)*	69
9.8	Definition(Klassen der Platzbeschränkten TM)*	69
9.9	Definition (Platzkonstruierbar)	69
9.10	Bemerkung (Platzkonstruierbare Funktionen: Eigenschaften)*	69
9.11	Satz (Platzhierarchiesatz für DTM)	69

¹Überschriften mit (*) Ed. sug.

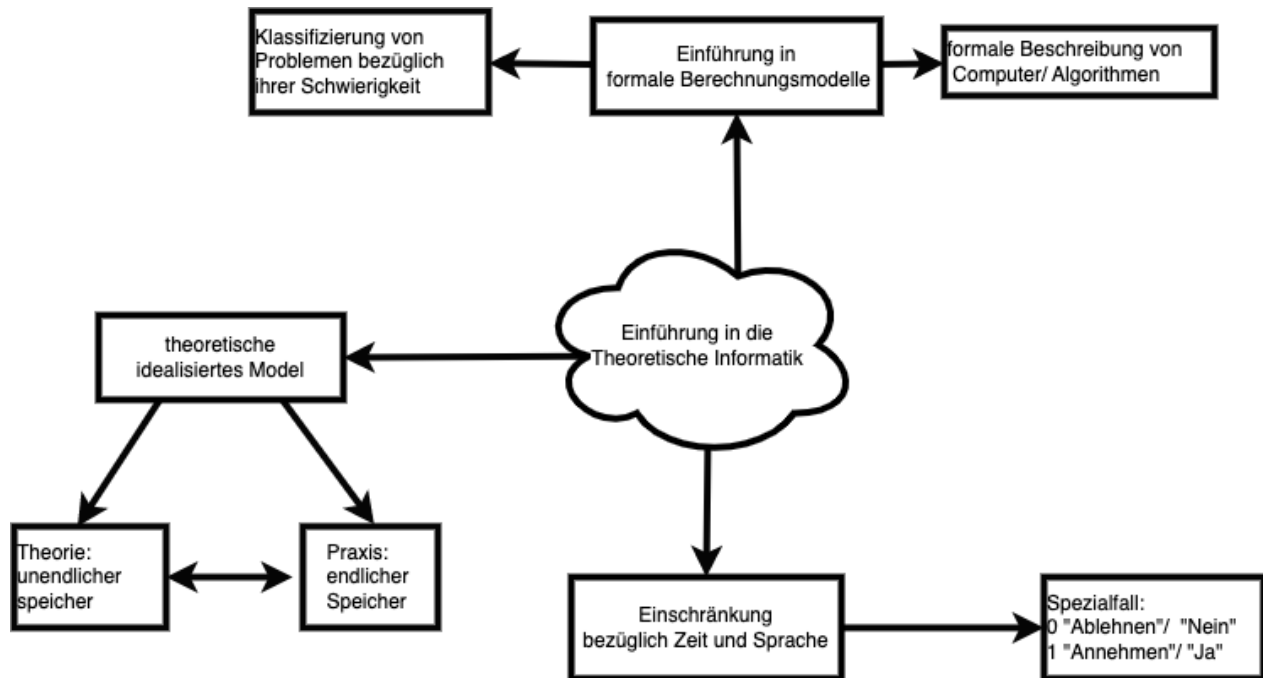


Abbildung 1: Überblick theoretische Informatik

Grundlagen

1.1 Notationen und begriffe

- \mathbb{N} bezeichnet die $\{1, 2, 3\}$
- \mathbb{N}_0 , sei $[n] = \{1, \dots, n\}$ und $[n]_0 = \{0, 1, \dots, n\}$
- Für eine Menge A und $n \in \mathbb{N}$ ist $A^n = \{(a_1, \dots, a_n) : a_1, \dots, a_n \in A\}$
- Für $n \in \mathbb{N}$ ist eine n -äre partielle funktion $\varphi : A^n \rightsquigarrow B$ eine Funktion mit $\text{dom}(\varphi) \subseteq A^n$ und $\text{Im}(\varphi) \subseteq B$. Für $a_1, \dots, a_n \in A$ bedeutet $\varphi(a_1, \dots, a_n) \downarrow$, dass $(a_1, \dots, a_n) \in \text{dom}(\varphi)$ gilt und $\varphi(a_1, \dots, a_n) \uparrow$ bedeutet, dass $(a_1, \dots, a_n) \notin \text{dom}(\varphi)$. Statt $\varphi(a_1, \dots, a_n) \uparrow$ schreiben wir auch $\varphi(a_1, \dots, a_n) = \uparrow$. Die partielle Funktion φ ist total, wenn $\text{dom}(\varphi) = A^n$ gilt.
- Eine lineare Ordnung, auch totale Ordnung, auf einer Menge A ist eine Relation $\leq \subseteq A^n$ m sodass die folgende Eigenschaften erfüllt sind. (wie für Relationen üblich verwenden wir hier Infixnotation, schreiben also für $a, b \in A$ den Ausdruck $a \leq b$ anstatt $(a, b) \in \leq$):
 - (i) $a \leq a \forall a \in A$ (Reflexivität)
 - (ii) $a \leq b \wedge b \leq a \Rightarrow a = b \forall a, b \in A$ (Antisymmetrie)
 - (iii) $a \leq b, b \leq c \Rightarrow a \leq c \text{ for all } a, b, c \in A$ (Transitivität)
 - (iv) $a \leq b \vee b \leq a \forall a, b \in A$ (Totalität)

1.2 Alphabet, Wörter und Sprachen

Eingaben und Ausgaben in unseren Berechnungsmodellen werden wörter genannt, wobei wir beliebige Zeichenketten als Wörter zulassen.

1.2.1 Definition (Alphabet)

Ein Alphabet ist eine nichtleere endliche Menge Σ . Das Alphabet Σ wird $|\Sigma|$ -är bezeichnet. Die Elemente von Σ heißen Buchstaben oder Symbole.

1.2.2 Definition (Wörter)

Ein Wort über einem Alphabet Σ ist eine endliche Folge von Symbolen aus Σ . Die Länge eines Wortes w ist $|w|$. Für $i \in |w|$ bezeichnet $w(i)$ das i -te Element von w und für Symbole $a_1, \dots, a_n \in \Sigma$ bezeichnet a_1, \dots, a_n das Wort w der Länge n mit $w(i) = a_i \forall i \in [n]$. Das Wort der Länge 0 heißt leeres Wort und wird λ bezeichnet. Ein Wort der Länge 1 wird mit dem Symbol $w(1)$ identifiziert.

1.2.3 Definition (Binäralphabet, Binärwörter)

Das Alphabet $\{0, 1\}$ heißt Binäralphabet. Die Wörter über dem Binäralphabet heißen Binärwörter.

1.2.4 Definition (Sprache)

Eine **Sprache** ist eine Menge von Wörter über einem gemeinsamen Alphabet Σ . Einige einfache grundlegenden Sprachen sind die folgenden.

1.2.5 Definition (Wortmengen-Notation)*

Die Menge aller Wörter über Σ wird mit Σ^* bezeichnet. Für $n \in \mathbb{N}_0$ setzen wir:

$$\Sigma^{\leq n} := \{w \in \Sigma^* : |w| \leq n\}$$

$$\Sigma^n := \{w \in \Sigma^* : |w| = n\}$$

$$\Sigma^{\geq n} := \{w \in \Sigma^* : |w| \geq n\}$$

$$\Sigma^+ := \Sigma^{\leq 1}$$

1.2.6 Definition (Verkettung)

Für Wörter w_1, w_2 ist die Verkettung $w_1 \circ w_2$, auch $w_1 w_2$, von w_1 und w_2 ist definiert durch:

$$w_1 \circ w_2 := w_1 \cdots w_1(|w_1|) w_2 \cdots w_2(|w_2|)$$

Für ein Wort w und $n \in \mathbb{N}_0$ ist w^n induktiv definiert durch $w^n := \lambda$ falls $n = 0$ und $w^n := w^{n-1} \circ w$ falls $n \geq 1$. Für eine Sprache L sei durch $L^0 := \{\lambda\}$, auch L^1 , definiert durch

$$L^1 \circ L^1 := \{w_1 w_2 : w_1 \in L^1, w_2 \in L^1\}$$

Für eine Sprache L und $n \in \mathbb{N}_0$ ist L^n induktiv definiert durch $L^n = \{\lambda\}$ falls $n = 0$ und $L^n := L \cdot L^{n-1}$ falls $n \geq 1$. Zudem sei $L^* := \bigcup_{n \in \mathbb{N}_0} L^n$. Für ein Wort w und eine Sprache L sei $wL := \{w\} \circ L$ und $Lw := L \circ \{w\}$.

Wir folgen der Konvention, dass \bullet^n und \bullet^* stärker binden als \circ ; für Wörter u, v gilt also $uv = u \circ (v^n)$. Insbesondere gilt auch $ab^n = a(b^n)$ für Symbole a, b eines Alphabets Σ .

1.2.7 Definition (Präfix, Infix, Suffix)

Seien u, v Wörter.

- (i) u ist Präfix von v , kurz $u \sqsubseteq v$, falls es ein Wort w gibt, sodass $uw = v$.
- (ii) u ist Infix von v falls es Wörter w_1, w_2 gibt sodass $v = w_1 u w_2$
- (iii) u ist Suffix von v , falls es ein Wort w gibt, sodass $v = wu$.

²Kleene-Stern, nach Stephen Cole Kleene, 1909-1994

1.2.8 Definition (präfixfrei)

Eine Sprache heißt **präfixfrei**, wenn $u \sqsubseteq v \Rightarrow u = v \quad \forall u, v \in L$.

1.2.9 Definition (Homomorphismus)

Für Sprache L und M heißt eine Funktion $\varphi : L \rightarrow M$ **Homomorphismus von Sprachen**, wenn $\varphi(uv) = \varphi(u)\varphi(v) \forall u, v \in L$ gilt.

1.2.10 Definition (Längenlexikographische Ordnung)

Ist Σ ein Alphabet und \leq eine lineare Ordnung auf Σ , so ist die zu \leq gehörige **längenlexikographische Ordnung** \leq_{lex} auf Σ^* die lineare Ordnung für die $u \leq_{lex} v$ genau dann für zwei verschiedene $u, v \in \Sigma^*$ gilt, wenn eine der folgenden Bedingungen gilt:

- $|u| < |v|$
- $|u| = |v|$ und ist $i \in [|u|]$ minimal mit $u(i) \neq v(i)$ so gilt $u(i) < v(i)$.

Bemerkung: Oft gehen wir von einer impliziten Ordnung auf Σ aus. Ist $\Sigma = a_1, \dots, a_n$ so gilt $a_1 < \dots < a_n$

1.2.11 Bemerkung (Endliche längenlexikographische Reihenfolge)*

Sei Σ ein Alphabet $\forall w \in \Sigma^*$ ist $v \in \Sigma^* : v \leq_{lex} w$ endlich. Dies erlaubt es uns für ein Alphabet Σ die Wörter über Σ in längenlexikographischen Reihenfolge w_1, w_2, \dots zu betrachten, wobei wir w_i für $i \in \mathbb{N}$ als kleinstes Element von $\Sigma^* / w_1, \dots, w_{i-1}$ gewählt sei. Wir identifizieren oft \mathbb{N}_0 mit $\{0, 1\}^*$ indem wir $i \in \mathbb{N}_0$ mit in die längenlexikographische Reihenfolge $(i+1)$ -ten Wort $w_{i+1} \in \{0, 1\}^*$ identifizieren.

\mathbb{N}_0	0	1	2	3
$\{0, 1\}^*$	λ	0	1	00

1.2.12 Definition (Binärwort-Funktion in längenlexikographischer Reihenfolge)*

Es bezeichnet $\text{bin} : \mathbb{N}_0 \rightarrow \{0, 1\}^*$ die Funktion, für die $\text{bin}(i)$ das in längenlexikographischer Reihenfolge $(i+1)$ -te Binärwort ist $\forall i \in \mathbb{N}_0$

1.2.13 Bemerkung (Beziehung zwischen Binärworten und Dezimalzahlen)*

$\forall i \in \mathbb{N}_0$ ist i $\text{bin}(i)$ die Binärdarstellung von $i+1$. Umgekehrt ist $\forall w \in \{0, 1\}^*$ das $(2^{|w|} + \sum_{i \in [|w|]} w(i) 2^{|w|-i})$ -te Binärwort.

Turingmachine

A Turing machine is like a wise old person, sitting at an endless table, playing a complex game. They have a magical pen that reads and writes on the game board. They follow strict rules, do not move from their spot, but the table mysteriously moves back and forth. Their concentration is deep and calm as they perform a complex ballet of reading, writing, and state-changing.

- ChatGPT

Wir Betrachte das folgende, sehr bekannt, berechnunsmodell. Anschaulich lässt es sich wie folgt beschreiben.

- Es gibt einen 'Speicher' \rightsquigarrow k unendlich lange Arrays (**Bänder**)
- Es gibt einen 'Arbeitsspeicher' \rightsquigarrow eine endliche Menge von Zuständen, die die Maschine einnehmen kann
- Für jedes Band gibt es einen Schreib- und Lesekopf
- Jeder Schritt ist wie folgt:
Abhängig von Zustand und gelesene Symbol, Schreiben die Köpfe genau ein Symbol, bewegen sich nun maximal eine Position und der Zustand der Maschine wird geändert.
- Stellt die Maschine ihr schrittweises Arbeiten ein, so wird die Ausgabe entweder den Zustand entnommen oder von einem der Bänder in geeigneter Weise abgelesen.

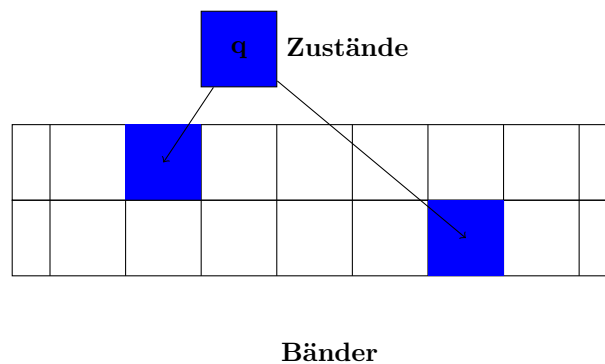


Abbildung 2: Turingmaschine

2.1 Definition (Turingmaschine, Alan Turing, 1936)

Sei $k \in \mathbb{N}$ eine **k-Band-Turingmaschine** kurz k-TM, ist ein Tupel $M = (Q, \Sigma, \Gamma, \Delta, s, F)$. Dabei ist:

- Q eine endliche Menge, **Zustandmenge**
- Σ das **Eingabealphabet**, ein Alphabet $\square \notin \Sigma$
- Γ das **Bandalphabet**, ein Alphabet mit $\Sigma \subseteq \Gamma$ und $\square \in \Gamma/\Sigma$
- $\Delta \subseteq Q \times \Gamma^k \rightarrow \subseteq Q \times \Gamma^k \times L, S, R^k$ die **Übergangsrelation**
- $s \in Q$ der **Startzustand**
- $F \subseteq Q$ die Menge der **akzeptierenden Zustände**

Das Symbol \square heißt **Blank**. Die Elemente von Δ heißen **Instruktionen**. Für eine Instruktion $(q_1, a_1, \dots, a_k, q', a'_1, \dots, a'_k, B_1, \dots, B_k)$ heißt (q, a_1, \dots, a_k) der **Bedingungsteil** und $(q', a'_1, \dots, a'_k, B_1, \dots, B_k)$ der **Anweisungsteil**. Die TM M ist eine **deterministische k-Band Turingmaschine**, kurz k-DTM, wenn es $\forall b \in Q \times \Gamma^k$ höchstens eine Instruktion $i \in \Delta$ mit Bedingungsteil b .

2.2 Definition (Konfiguration)

Sei $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ eine k-TM. Eine **Konfiguration** von M ist ein Tupel

$$C = (q, w_1, \dots, w_k, p_1, \dots, p_k) \in Q \times (p^*)^k \times \mathbb{N}^k$$

Die **Startkonfiguration** von M zur Eingabe $(u_1, \dots, u_n) \in (\Sigma^*)^n$, wobei $n \in \mathbb{N}$, ist die Konfiguration

$$Start_M(u_1, \dots, u_n) = (s, u_1 \square u_2 \square \dots \square u_n, \square, \dots, 1, \dots, 1)$$

Die Konfiguration C ist eine **Stoppkonfiguration** von M, wenn es keine Instruktion $i \in \Delta$ mit Bedingungsteil $(q, w_1(p_1), \dots, w_k(p_k))$ gibt.

2.3 Definition (Nachfolgekonfiguration)

Sei $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ eine k-DTM. Für Konfiguration $C = q_1, w_1, \dots, w_k, p_1, \dots, p_k$ und $C' = q'_1, w'_1, \dots, w'_k, p'_1, \dots, p'_k$ von M ist die Konfiguration C' Nachfolgekonfiguration von C, wenn es eine Instruktion

$$(q, w_1(p_1), \dots, w_k(p_k), a'_1, a'_k, B_1, \dots, B_k) \in \Delta$$

gibt, sodass

$$w'_i = \begin{cases} \square a'_i w_i(2) \dots w_i(|w_i|), & \text{falls } p_i = 1 \text{ und } B_i = L \\ w_i \dots w_i(|w_i| - 1) a'_i \square, & \text{falls } p_i = |w_i| \text{ und } B_i = R \\ w_i \dots w_i(p_i - 1) a'_i w_i(p_i + 1) \dots w_i(|w_i|), & \text{sonst} \end{cases}$$

und

$$p'_i = \begin{cases} 1, & \text{falls } p_i = 1 \text{ und } B_i = L \\ p_i - 1, & \text{falls } p_i \geq 2 \text{ und } B_i = L \\ p_i, & \text{falls } B_i = S \\ p_i + 1, & \text{falls } B_i = R \end{cases}$$

$\forall i \in [k]$ gelten.

Es bezeichnen \rightarrow_M die Relation auf der Menge der Konfiguration von M, sodass $C \rightarrow_M C'$ falls C, C' Konfig von M sind wobei C' eine Nachfolgekonfiguration von C ist.

2.4 Definition (Rechnung)

Sei $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ eine k-DTM. Eine **endliche partielle Rechnung** von M ist eine endliche Folge C_1, \dots, C_n von Konfig von M mit $C_i \rightarrow_M C_{i+1} \forall i \in [n-1]$. Eine **unendliche partielle Rechnung** von M ist eine unendliche Folge C_1, C_2, \dots von Konfiguration von M mit $C_1 \rightarrow_M C_{1+i} \forall i \in \mathbb{N}$. Eine **Rechnung von M zur Eingabe** $(w_1, \dots, w_n) \in (\Sigma^*)^n$ (mit $n \in \mathbb{N}$) ist eine endliche partielle Rechnung $start_M = C_1, \dots, C_m$ bei der C_m eine Stoppkonfiguration von M oder eine unendliche partielle Rechnung $start_M(w_1, \dots, w_n) = C_1, C_2, \dots$

2.5 Bemerkung (Eindeutige Rechnungen bei k-DTM.)*

Ist M eine k-DTM, so gilt es $\forall n \in \mathbb{N}$ und $(w_1, \dots, w_n) \in (\Sigma^*)^n$ genau eine Rechnung zur Eingabe (w_1, \dots, w_n) .

2.6 Definition (total)

Eine k-DTM $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ **terminiert** bei Eingabe $(w_1, \dots, w_n) \in (\Sigma^*)^n$ wenn die Rechnung von M zur Eingabe (w_1, \dots, w_n) endlich ist. Eine k-TM $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ ist **total**, wenn $\forall n \in \mathbb{N}$ und $(w_1, \dots, w_n) \in (\Sigma^*)^n$ alle Rechnungen von M zur Eingabe (w_1, \dots, w_n) endlich sind.

2.7 Definition (akzeptierte Sprache)

Sei $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ eine k-TM. Eine Stoppkonfiguration $(q, w_1, \dots, w_k, p_1, \dots, p_k)$ von M ist **akzeptierend**, wenn $q \in F$. Die **akzeptierte Sprache** $L(M)$ von M ist die Sprache über dem Alphabet Σ so dass $w \in L(M)$ gilt, wenn es eine endliche Rechnung C_1, \dots, C_n von M zur Eingabe w gibt, bei der C_n eine akzeptierende Stoppkonfiguration von M ist.

Hinweis: Für nicht deterministische TM heißt das insbesondere, dass es für die Wörter w in der akzeptierten Sprache nur mindestens **eine** im einer akzeptierten Stoppkonfiguration endende endliche Rechnung zur Eingabe w geben muss. Für Wörter w , die nicht in $L(M)$ sind, sind **alle** rechnungen von M zur Eingabe am Ende nicht in einer akzeptierten Stoppkonfiguration oder unendlich.

Zeitbeschränkte Funktionen und Sprachen

2.8 Definition(rekursiv aufzählbar)

Eine Sprache L ist genau dann **rekursiv aufzählbar**, wenn es eine k -TM mit akzeptierten Sprache L gibt. Wir schreiben **RE** für die Klasse der rekursiv aufzählbaren Sprachen. Die Aufzählbarkeit leitet sich daraus ab, dass es für eine rekursiv aufzählbare Sprache L über einem Alphabet Σ möglich ist effektive Verfahren anzugeben, die die Wörter von L aufzählen, also dass eine endlich oder unendliche Aufzählung von $A = w_1, w_2, \dots$ mit $w_1, w_2, \dots = L$ existiert.

Eine Menge wird beschreiben, die wir mit einem Computerprogramm oder Algorithmus "auflisten" können. Stellen Sie sich vor, Sie haben eine Box mit nummerierten Bällen, und Sie haben ein Programm, das Bälle aus der Box zieht. Wenn Sie sicherstellen können, dass Sie jeden Ball in der Box mindestens einmal ziehen, egal wie lange es dauert, dann ist die Menge der Bälle in der Box "rekursiv aufzählbar" ³

2.9 Bemerkung (Entscheidbare Sprache und rekursiv aufzählbare Sprache)*

Jede entscheidbare Sprache ist rekursiv aufzählbar.

2.10 Bemerkung (Endliche Sprachen sind entscheidbar.)*

Alle endlichen Sprachen sind entscheidbar.

2.11 Bemerkung (Entscheidbarkeit und Aufzählbarkeit.)*

Eine Sprache L über einem Alphabet Σ ist genau dann entscheidbar, wenn L und $L^c := (\Sigma^*)/L$ rekursiv aufzählbar sind.

2.12 Definition (Ausgabe)

Sei $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ ein k -TM und $C = (q, w_1, \dots, w_k, p_1, \dots, p_k)$ eine Konfiguration von M . Die Ausgabe $out_M(C)$ von M bei Konfiguration C ist das Präfix $w \sqsubseteq w_1(p_1) \dots w_1(|w_1|)$ maximale Länge mit $w \in (\Gamma/\{\square\})^*$.

2.13 Definition (berechnete Funktion)

Sei $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ eine k -DTM und $n \in \mathbb{N}$. Die von M berechnete **n -äre partielle Funktion** Φ_M ist die partielle Funktion $\Phi_M : (\Sigma^*)^n \rightsquigarrow (\Gamma/\square)^*$, so dass $\forall (w_1, \dots, w_n) \in (\Sigma^*)^n$ folgendes gilt:

1. Ist die Rechnung von M zur Eingabe (w_1, \dots, w_n) die endliche Rechnung C_1, \dots, C_M , so gilt $\Phi_M(w_1, \dots, w_n) = out_M(C_M)$.
2. Ist die Rechnung von M zur Eingabe (w_1, \dots, w_n) unendlich, so gilt $\Phi_M(w_1, \dots, w_n) \uparrow$

Für $w_1, \dots, w_n \in \Sigma^*$ schreiben wir statt $\Phi_M(w_1, \dots, w_n)$ auch $M(w_1, \dots, w_n)$.

³Ed. sug. text

2.14 Definition (partiell berechenbar)

Für Alphabet Σ, Γ und eine partielle Funktion $\Phi : \Sigma^* \rightsquigarrow \Gamma^*$ ist Φ **partiell berechenbar**, wenn es eine $k \in \mathbb{N}$ gibt und eine k-DTM M mit $\Phi_M = \Phi$ gibt. Ist Φ total und partiell berechenbar, so ist Φ berechenbar.

Mittels der Induktivität von \mathbb{N}_0 und $\{0,1\}^*$ können so auch partielle Funktionen, die von oder nach \mathbb{N}_0 abbilden als (partielle) berechenbare Funktion bezeichnet werden. Beispielsweise ist eine partielle Funktion $\Phi : \mathbb{N}_0 \rightsquigarrow \mathbb{N}_0$ dennoch genau dann partiell berechenbar, wenn die partielle Funktion $\text{bin} \circ \Phi \circ \text{bin}^{-1}$ partiell berechenbar ist. Gewissermaßen verfügen die hier definierten TM über zwei Ausgabemechanismen. Die Ausgabe im engeren Sinne in [Definition 2.13 \(Ausgabe\)](#) und das Ablesen von Akzeptanz anhand des schließlich erreichten Zustands in [Definition 2.7 \(Akzeptierte Sprache\)](#). Im Sinne der folgenden Bemerkung wäre der zweiten Fall nicht notwendig, allerdings ist dies ein wichtiger Spezialfall.

Wir schreiben **RF** für die Klasse der partiellen Funktionen

2.15 Definition (charakteristische Funktion, partielle charakteristische Funktion)

Sei L eine Sprache über dem Alphabet Σ

- (i) Die **charakteristische Funktion** von L als Sprache über Σ ist die Funktion $\mathbb{1}_L : \Sigma \rightarrow \{0,1\}$ mit $\mathbb{1}_L(w) = 1 \quad \forall w \in L$ und $\mathbb{1}_L(u) = 0 \quad \forall w \in \Sigma^*/L$.
- (ii) Die **partielle charakteristische Funktion** von L als Sprache über Σ ist die partielle Funktion $x_L : \Sigma^* \rightsquigarrow \{1\}$ mit $x_L(w) = 1 \quad \forall w \in L$ und $x_L(w) \uparrow \quad \forall w \in \Sigma^*/L$.

2.16 Bemerkung (Entscheidbarkeit und Berechenbarkeit)*

Sei L eine Sprache über einem Alphabet Σ .

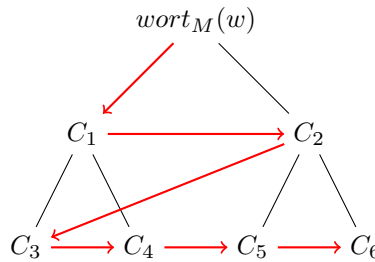
- (i) L ist genau dann entscheidbar, wenn $\mathbb{1}_L$ berechenbar ist.
- (ii) L ist genau dann rekursiv aufzählbar, wenn x_L partiell berechenbar ist.

2.17 Bemerkung (normiert)

Eine 1-DTM $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ heißt **normiert**, wenn $Q = 0, \dots, n$ für eine $n \in \mathbb{N}_0$, $\Sigma = 0, 1$, $\Gamma = \square, 0, 1$, $s = 0$, $F = s$. Alle TMs mit Eingabealphabet $0, 1$ lassen sich mit folgenden Schritten in eine normierte TM mit gleicher erkannter Sprache und gleicher berechneter Funktion umwandeln.

Von Nichtdeterminismus zu Determinismus: Eine DTM kann die Rechnungen einer nichtdeterministischen TM parallel im Sinne von abweichend schrittweise durchführen um schließlich das Verhalten der simulierten TM zu ?? Dies entspricht einer **Breitensuche im Rechnungsbaum**.

Abbildung 3: Breitensuche



Von mehreren Bändern zu einem Band : Intuitiv können k Bänder auf ein Band simuliert werden, indem die Felder des einen Bandes in k -teilerfelder unterteilt werden, die jeweils die gleiche Bandalphabetbuchstaben wie zuvor als Beschreibung zulassen und es zudem erlaubt zu markieren, dass der simulierte Kopf des simulierten Bandes dort steht. Eine dieser Idee folgende Konstruktion wird als **Spurenteknik** bezeichnet. Formal: Übergang vom Bandalphabet Γ zu

$$((\Gamma \cup \underline{a} : a \in \Gamma)^k / \square^k) \cup \square$$

wobei $\underline{a} \notin \Gamma$ für $a \in \Gamma$. Hierbei bedeutet \underline{a} , dass das simulierte Feld mit a beschriftet ist und dass dort der simulierte Kopf steht. Weiter spielt \square die Rolle des k -Tupels $(\square, \dots, \square)$ um der Tatsache gerecht zu werden, dass alle Felder zu Beginn mit \square beschriftet sind.

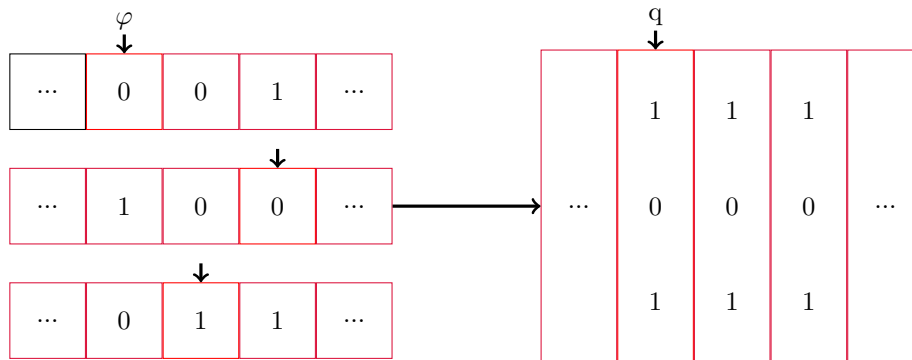


Abbildung 4: Spurenteknik

Von beliebigen bandalphabet zu $\{\square, 0, 1\}$: Andere bandalphabete können bei einem **Alphabetwechsel** zum Bandalphabet $\{\square, 0, 1\}$ simuliert werden um ein Symbol des vorherigen Bandalphabets durch ein Binärwort zu beschreiben. Die TM liest stets nur ein Feld, es wird dabei also nötig sein die Zustandsmenge so zu erweitern, dass angrenzende Felder im Zustand gespeichert werden können.

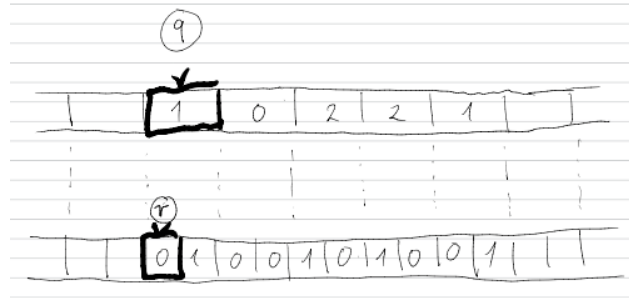


Abbildung 5: Alphabetwechsel

2.18 Bemerkung (Entscheidbarkeit, Aufzählbarkeit, Berechenbarkeit.)*

Sei $L \subseteq \{0,1\}^*$ eine Sprache und sei $\Phi : \{0,1\}^* \rightsquigarrow \{0,1\}^*$ eine partielle Funktion.

- (i) L ist genau dann entscheidbar, wenn L akzeptierte Sprache einer totalen normierten TM ist.
- (ii) L ist genau dann rekursiv aufzählbar, wenn L akzeptierte Sprache einer normierten TM ist.
- (iii) Φ ist genau dann partiell berechenbar, wenn Φ berechnete Funktion einer normierten TM ist.

Church- Turing- These Berechenbarkeit auf einer Turingmaschine entspricht intuitiver Berechenbarkeit.

Berechenbarkeit

Predictability is like knowing the path a river takes. The river starts at its source and flows down to the sea. Along the way, it may turn, twist, and divide, but it always follows the path of least resistance due to gravity. Knowing the terrain allows us to predict where the river will go.
- ChatGPT

Konvention: Sprechen wir von einer $e \in \mathbb{N}_0$ oder $(e_1, \dots, e_n) \in \mathbb{N}_0^n$ wobei $n \in \mathbb{N}$ als Eingabe für eine TM oder Ausgabe einer TM, so bedeutet dies, dass die Eingabe bzw. Ausgabe $\text{bin}(e)$ bzw. $(\text{bin}(e_1), \dots, \text{bin}(e_n))$ ist. Dies erlaubt es über partiell berechenbare Funktionen $\Phi : \mathbb{N}_0^n \rightsquigarrow \mathbb{N}_0$ wobei $n \in \mathbb{N}$ zu sprechen und $L \subseteq \mathbb{N}_0$ als Sprache über $\{0, 1\}$ aufzufassen.

3.1 Definition (Code)

Wir betrachten die Funktion code (mit geeignetem Definitionsbereich) und Zielmenge $\{0, 1\}^*$, für die folgendes gilt. Zunächst gelte

$$\text{code}(L) = 10 \quad \text{code}(S) = 00 \quad \text{code}(R) = 01$$

Für eine Instruktion $I = (q, a, q', a', B) \in \mathbb{N}_0 \times \{0, 1\} \rightarrow \mathbb{N}_0 \times \{0, 1\} \times \{L, S, R\}$ einer normierten TM sei

$$\text{code}(I) = 0^{|\text{bin}(q)|} 1 \text{bin}(q) a 0^{|\text{bin}(q')|} 1 \text{bin}(q') a' \text{code}(B)$$

Für eine endliche Menge $\Delta \subseteq \mathbb{N}_0 \times \{0, 1\} \rightarrow \mathbb{N}_0 \times \{0, 1\} \times \{L, S, R\}$ von Instruktionen einer normierten TM und $i \in [|\Delta|]$ sein $\text{code}_i(\Delta)$ dann ein längenlexikographische Ordnung i -te Wort in $\{\text{code}(I) : I \in \Delta\}$ und sei

$$\text{code}(\Delta) = \text{code}_1(\Delta), \dots, \text{code}_{|\Delta|}(\Delta)$$

Für eine normierte TM $M = (\{0, \dots, n\}, \{0, 1\}, \{\square, 0, 1\}, \Delta, 0, \{0\})$ sei

$$\text{code}(M) = 0^{|\text{bin}(n)|} 1 \text{bin}(n) \text{code}(\Delta)$$

der **Code** von M . Relevant ist hierbei dass es eine geeignete effektive Codierung von Turingmaschinen durch Binärwörter gibt, so dass folgendes gilt

- Jede normierte TM hat einen Code
- Keine zwei verschiedene normierten TMs haben den gleichen Code.
- Die Sprache der Codes von Turingmaschinen ist entscheidbar
- Codes können eine geeignete Repräsentation der durch sie codierten TMs umgewandelt werden, die es insbesondere erlauben die codierten TMs effektiv zu simulieren.
- geeignete Repräsentationen von TMs können effektiv in ihre Codes umgewandelt werden.

3.2 Definition (standardaufzählung)

Sei $\hat{w}_0, \hat{w}_1, \dots$ die Aufzählung aller Codes normierter TMs in längenlexikographischer Ordnung. Für $e \in \mathbb{N}_0$ sei M_e die durch \hat{w}_e codierte TM und für $n \in \mathbb{N}$ sei $\Phi_e^n : \mathbb{N}_0^n \rightarrow \mathbb{N}_0$ die von M_e berechnete n -äre partielle Funktion. Für $n \in \mathbb{N}$ heißt die Folge (Φ_e^n) mit $e \in \mathbb{N}$ **standardaufzählung** der n -ären partiell berechenbaren Funktion. Für $n \in \mathbb{N}$ und eine partiell berechenbare n -äre Funktion $\varphi : \mathbb{N}_0^n \rightarrow \mathbb{N}_0$ heißt jede Zahl $e \in \mathbb{N}_0$ mit $\Phi_e^n = \varphi$ **Index** von φ .

Konvention: Ergibt sich n aus dem Kontext, so schreiben wir auch Φ_e statt Φ_e^n

3.3 Bemerkung(Unendlich viele Indizes)*

Für $n \in \mathbb{N}$ und eine partiell berechnbare n -äre partielle Funktion $\Phi : \mathbb{N}_0^n \rightarrow \mathbb{N}_0$ gibt es unendlich viele Indizes von φ .

3.4 Definition (U)

Es bezeichnet U die normierte TM, die bei Eingabe $(e, x_1, \dots, x_n) \in \mathbb{N}_0^{n+1}$ wobei $n \in \mathbb{N}$ die normierte TM \mathcal{M}_e bei Eingabe (x_1, \dots, x_n) simuliert und falls diese terminiert die Ausgabe der Simulierten ausgibt.

3.5 Definition (Universell)

Eine DTM U heißt **Universell**, wenn es für alle $n \in \mathbb{N}$ und alle partiell berechenbaren Funktionen $\varphi : \mathbb{N}_0^n \rightsquigarrow \mathbb{N}_0$ eine $e \in \mathbb{N}$, so dass

$$U(e, x_1, \dots, x_n) = \varphi(x_1, \dots, x_n)$$

$\forall x_1, \dots, x_n \in \mathbb{N}_0$ gilt.

3.6 Bemerkung(Universelle Turingmaschine U)*

Die TM U ist universell, denn für $e \in \mathbb{N}_0$, $n \in \mathbb{N}$ und $x_1, \dots, x_n \in \mathbb{N}_0$ gilt

$$U(e, x_1, \dots, x_n) = \Phi_e(x_1, \dots, x_n)$$

$$(x, y) \mapsto x^y$$

$$y \mapsto 2^y$$

$$(x_1, \dots, x_m, y_1, \dots, y_n) \mapsto \varphi(x_1, \dots, x_m, y_1, \dots, y_n) \text{ partiell berechenbar}$$

$$\rightsquigarrow (y_1, \dots, y_m) \mapsto \varphi(x_1, \dots, x_m, y_1, \dots, y_n) \text{ partiell berechenbar}$$

3.7 Satz (s_n^m - Theorem)

$\forall m, n \in \mathbb{N}$ existiert eine berechenbare Funktion $s_n^m : \mathbb{N}_0^{m+1} \rightarrow \mathbb{N}_0$ mit

$$\Phi_e^{m+1}(x_1, \dots, x_m, y_1, \dots, y_n) = \Phi_{s_n^m(e, x_1, \dots, x_m)}^n(y_1, \dots, y_n)$$

$\forall e, x_1, \dots, x_m, y_1, \dots, y_n \in \mathbb{N}_0$

Beweis. Fixiere $m \in \mathbb{N}$. Betrachte die DTM S , die bei Eingabe $(e, x_1, \dots, x_m) \in \mathbb{N}_0^{m+1}$ wie folgt vorgeht.

- Zunächst bestimmt S den Code von \mathcal{M}_e
- der Code von \mathcal{M}_1 wird dann in einen Code einer normierten TM \mathcal{M} umgewandelt, die zunächst $x_1 \square \dots \square x_m \square$ neben die Eingabe schreibt, dann den Kopf auf das erste Feld des beschriebenen Bandteilsbewegt und dann wie \mathcal{M}_1 arbeitet.
- Es wird bestimmt an welcher Stelle der Standardaufzählung der Code von \mathcal{M}_e auftaucht und diese Stelle wird ausgegeben.

Sei s_n^m die von S berechnete $(m+1)$ -äre partielle Funktion. Dann ist s_n^m eine Funktion wie gewünscht. Es gibt überabzählbar viele Binärsprachen, denn: Betrachte Aufzählung von Binärsprachen L_1, L_2, \dots

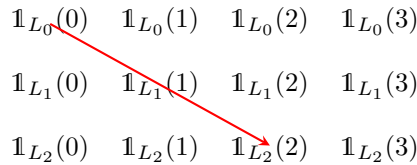
$$\begin{array}{cccc} \mathbb{1}_{L_0}(0) & \mathbb{1}_{L_0}(1) & \mathbb{1}_{L_0}(2) & \mathbb{1}_{L_0}(3) \\ \mathbb{1}_{L_1}(0) & \mathbb{1}_{L_1}(1) & \mathbb{1}_{L_1}(2) & \mathbb{1}_{L_1}(3) \\ \mathbb{1}_{L_2}(0) & \mathbb{1}_{L_2}(1) & \mathbb{1}_{L_2}(2) & \mathbb{1}_{L_2}(3) \end{array}$$


Abbildung 6: Standardaufzählung

$$L \text{ mit } \mathbb{1}_L(i) = \begin{cases} 0, & \text{wenn } \mathbb{1}_{L_i}(i) = 1 \\ 1, & \text{wenn } \mathbb{1}_{L_i}(i) = 0 \end{cases}$$

□

3.8 Definition (diagonales Halteproblem)

Die Menge $H_{diag} := \{e \in \mathbb{N}_0 : \Phi_e(e) \downarrow\}$ heißt **diagonales Halteproblem**.

3.9 Proposition(Diagonales Halteproblem)*

Das diagonale Halteproblem ist rekursiv aufzählbar.

Beweis. Die DTM, die bei Eingabe $e \in \mathbb{N}_0$ wie U bei Eingabe (e, e) arbeitet, aber bei terminieren 1 statt der Ausgabe von U ausgibt berechnet die partielle charakteristische Funktion von H_{diag} . Die partielle Funktion $x_{H_{diag}}$ ist also partiell berechenbar. Die partielle Funktion $x_{H_{diag}^c}$ ist nicht partiell berechenbar, dann: Betrachte Standardaufzählung \square

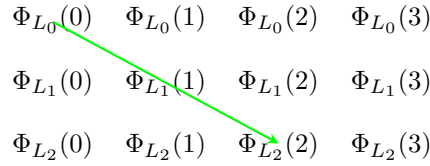


Abbildung 7: Standardaufzählung

φ mit $\varphi(i) = \begin{cases} \uparrow, & \text{wenn } \Phi_i(i) \downarrow \\ \downarrow, & \text{wenn } \Phi_i(i) \uparrow \end{cases}$ Wird nicht aufgezählt.

3.10 Satz(Diagonalhalteproblem - Nicht entscheidbar)*

Das diagonale Halteproblem ist nicht entscheidbar.

Beweis. Angenommen H_{diag} wäre entscheidbar. Dann wäre die partielle charakteristische Funktion φ von $H_{diag}^c = \mathbb{N}_0 / H_{diag}$ partiell berechenbar, es gäbe also ein Index $e \in \mathbb{N}_0$ von φ . Es folge

$$e \in H_{diag}^c \Leftrightarrow \varphi(e) \downarrow \Leftrightarrow \Phi_e(e) \downarrow \Leftrightarrow e \in H_{diag} \Leftrightarrow e \notin H_{diag}^c$$

Das ist ein Widerspruch. \square

3.11 Definition (m-Reduktion)

Für eine Sprache A über einem Alphabet Σ und eine Sprache B über einem Alphabet Γ ist A genau dann **many-one-reduzierbar**, auch **m-reduzierbar**, auf B , kurz $A \leq_m B$, wenn es eine berechenbare Funktion $f : \Sigma^* \rightarrow \Gamma^*$ gibt so dass

$$w \in A \Leftrightarrow f(w) \in B$$

$\forall w \in \Sigma^*$ gilt. Gelten $A \leq_m B$ und $B \leq_m A$, so sind A und B **many-one-äquivalent** auch **m-äquivalent**, kurz $A =_m B$.

3.12 Bemerkung(Entscheidbarkeit unter Reduzibilität)*

- (i) \leq_m ist transitiv.
- (ii) Gilt $A \leq_m B$ für Sprachen A und B und ist B entscheidbar, so ist auch A entscheidbar.
- (iii) Alle entscheidbaren Sprachen L mit $\emptyset \neq L \neq \mathbb{N}_0$ und m-äquivalent.

3.13 Satz(Nicht-Entscheidbarkeit des initialen Halteproblems)*

Das **initiale Halteproblem** $H_{init} = e \in \mathbb{N}_0 = \Phi_e(0) \downarrow$ ist nicht entscheidbar.

Idee: suche $f : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ mit $\Phi_e(e) \downarrow \Leftrightarrow \Phi_{f(e)}(0) \downarrow$ Wähle f so dass $\Phi_{f(e)}(x) = \Phi_e(e) \forall x \in \mathbb{N}_0$

Beweis. Sei $\psi : \mathbb{N}_0^2 \rightsquigarrow \mathbb{N}_0$ mit $\psi(e, x) = \Phi_e(e) \forall e, x \in \mathbb{N}_0$. Dann ist ψ partiell berechenbar. Sei e_0 ein Index von ψ und $s : \mathbb{N}_0^2 \rightarrow \mathbb{N}_0$ gilt. Sei $f : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ mit $f(e) = s(e_0, e) \forall e \in \mathbb{N}_0$. Dann ist f berechenbar. $\forall e \in \mathbb{N}_0$ gilt.

$$e \in H_{diag} \Leftrightarrow \Phi_e(e) \downarrow \Leftrightarrow \psi(e, 0) \downarrow \Leftrightarrow \Phi_{e_0}(e, 0) \downarrow \Leftrightarrow \Phi_s(e_0, e)(0) \downarrow \Leftrightarrow \Phi_{f(e)}(0) \downarrow \Leftrightarrow f(e) \in H_{init}$$

Es gilt also $H_{diag} \leq_m H_{init}$, da H_{diag} nicht entscheidbar ist, ist damit H_{init} nicht entscheidbar. \square

Dominosteinspiel!

Gegeben: Endlich viele typen von Spielsteinen mit jeweils zwei beschrifteten Feldern: „oberes Feld, unteres Feld“. Beschriftungen sind nichtleere Wörter über einem Alphabet. Spielsteine sind vom gleichen Typ, wenn die beiden oberen Felder gleich beschriftet sind und die beiden unteren Felder gleich beschriftet sind. Es gibt von jedem Typ beliebig viele steine.

Gesucht: Können ein oder mehrere (aber endlich viele) Spielsteine so nebeneinander gelegt werden, dass sich oben und unten von links nach rechts gelesen das gleiche Wort ergibt?

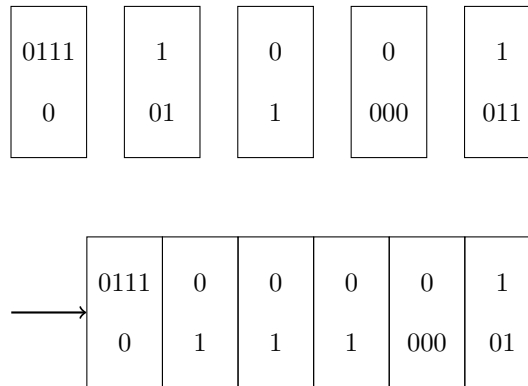


Abbildung 8: Dominosteinspiel

3.14 Definition (Postsches Korrespondenzproblem, Emil Post, 1946)

Für ein Alphabet Σ sei eine Instanz des Postschen Korrespondenzproblems über Σ eine endliche Teilmenge $I \subseteq (\Sigma^+)^2$. Eine Lösung für eine solche Instanz ist eine endliche Folge $(u_1, v_1), \dots, (u_n, v_n)$ von Paaren in I mit $n \geq 1$, so dass

$$u_1 \cdots u_n = v_1 \cdots v_n$$

Gibt es eine Lösung für eine Instanz des Postschen Korrespondenzproblems, so heißt diese Instanz lösbar. Das **Postsche Korrespondenzproblem** über einem Alphabet Σ , kurz PCP_Σ ist die Menge aller lösbaren Instanzen des Postschen Korrespondenzproblems über Σ .

Für ein Alphabet Σ sei eine Instanz des modifizierten Postschen Korrespondenzproblems über Σ ein Paar (p, I) , wobei $I \subseteq (\Sigma^+)^2$ eine endliche Teilmenge und $p \in I$ ein Paar von Wörtern ist. Eine Lösung für eine solche Instanz ist eine endliche Folge $(u_1, v_1), \dots, (u_n, v_n)$ von Paaren in I , so dass

$$p = (u_1, v_1) \text{ und } u_1 \cdots u_n = v_1 \cdots v_n$$

Gibt es eine Lösung für eine Instanz des modifizierten Postschen Korrespondenzproblems so heißt diese Instanz lösbar. Das **modifizierte Postsche Korrespondenzproblem** über einem Alphabet Σ , kurz $MPCP_\Sigma$ ist die Menge aller lösbaren Instanzen des modifizierten Postschen Korrespondenzproblems über Σ .

Plan: Für Alphabet mit $|\Sigma| \geq 2$:

$$H_{init} \stackrel{(3)}{\leq_m} MPCP_\Gamma \stackrel{(2)}{\leq_m} PCP_\Gamma \stackrel{(1)}{\leq_m} PCP_\Sigma$$

3.15 Lemma(PCP-Reduktion: $\Gamma \rightarrow \Sigma$)*

Für ein Alphabet Σ und Γ mit $|\Sigma| \geq w$ gilt $PCP_\Gamma \leq_m PCP_\Sigma$

Beweis. Wir suchen eine effektive Transformation, die jede Instanz I des Postschen Korrespondenzproblems über Γ in eine Instanz I' des postschen Korrespondenzproblems über Σ transformiert, so dass I genau dann lösbar ist, wenn I' lösbar ist. Seien $a_1, a_2 \in \Sigma$ verschieden und sein $b_1, \dots, b_{|\Gamma|}$ die Elemente von Γ . Es bezeichne $\varphi : \Gamma^* \rightarrow \Gamma^*$ den eindeutigen Homomorphismus von Sprachen mit $\varphi(b_i) = a_1^i a_2 \forall i \in [|\Gamma|]$. Gegeben eine solche Instanz I wie oben sei $I' := \{(\varphi(u), \varphi(v)) : (u, v) \in I\}$. Die Funktion, die geeignete Codes von Instanzen I auf geeignete Codes von Instanzen I' abbildet ist berechenbar. Ist $(u_1, v_1), \dots, (u_n, v_n)$ eine lösung I , so gilt

$$\varphi(u_1) \cdots \varphi(v_1) = \varphi(u_1, \dots, \varphi(v_n)) = \varphi(v_1, \dots, v_n) = \varphi(v_1) \cdots \varphi(v_n)$$

und somit ist $(\varphi(u_1), \varphi(v_1), \dots, (\varphi(u_n), \varphi(v_n)))$ eine Lösung von I' . Die Instanz I' ist also lösbar wenn I lösbar ist. Ist $(u'_1, v'_1), \dots, (u'_n, v'_n)$ eine Lösung von I' , so gibt es eine Folge $(u_1, v_1) \cdots (u_n, v_n)$ von Paaren in I mit $\varphi(u'_i)$ und $\varphi(v_i) = v'_i \forall i \in [n]$, also mit

$$\varphi(u_1, \dots, u_n) = u'_1, \dots, u'_n = u'_1, \dots, u'_n = \varphi(u_1, \dots, u_n)$$

Da $\varphi|_\Sigma$ injektiv und $\varphi(\Sigma)$ präfixfrei ist, ist φ injektiv (siehe Übung), folglich gilt $u_1, \dots, u_n = v_1, \dots, v_n$ und somit ist $(u_1, v_1), \dots, (u_n, v_n)$ eine Lösung von I . Die Instanz I ist also lösbar wenn I' Lösbar ist. \square

3.16 Lemma(Reduktion von MPCP über Σ auf PCP über Σ)*

Für Jedes alphabet Σ mit $|\Sigma| \leq w$ gitl $MPCP_\Sigma \leq_m PCP_\Sigma$.

Beweis. Sei Σ ein Alphabet mit $|\Sigma| \geq 2$. Nach Lemma 3.14 genügt es ein Alphabet Γ zu finden, so das $MPCP_\Sigma \leq_m PCP_\Gamma$ gilt.

Wir suchen eine effektive Transformation, die jede instanz (p, I) des modifizierten Postschen Korrespondenzproblems über Σ in eine Instanz I' des Postschen Korrespondenzproblems über einem geeignetem Alphabet Γ transformiert, so dass (p, I) genau dann lösbar ist, wenn I' lösbar ist. \square

Idee:

HIERABBEINFÜGEN

$* \in \Sigma$

HIERABBEINFÜGEN

Betrachte die Homomorphismus von Sprachen $\delta_{\leftarrow}, \delta_{\rightarrow} : \Sigma^* \rightarrow (\Sigma \cup *)^*$ mit $\delta_a = a*$ und $\delta_{\leftarrow}(a) = *a \forall a \in \Sigma$. Für jede Instanz $(p, I) = ((u_1, v_1), I)$ wie oben sei

$$I' = \{(\delta_{\leftarrow}(u_1), * \delta_{\rightarrow}(v_1))\} \cup \{\delta_{\leftarrowarrow}(u), \delta_{\rightarrowarrow}(v) : (u, v) \in I\} \cup \{\delta_{\leftarrow}(u)*, \delta_{\rightarrow}(v) : (u, v) \in I\}$$

Die Funktion die geeignete Codes von Instanzen (p, I) auf geeignete Codes der zugehörigen Instanzen I' abbildet ist berechenbar. Gibt es eine Lösung $(u_1, v_1), \dots, (u_n, v_n)$ von (p, I) dann ist

$$\begin{aligned} \delta_{\leftarrow}(u_1) \cdots \delta_{\leftarrow}(u_n)* &= \delta_{\leftarrow}(u_1 \cdots u_n)* \\ &= \delta_{\leftarrow}(v_1 \cdots v_n)* \end{aligned}$$

$$\begin{aligned}
&= * \delta_{\rightarrow}(v_1 \cdots v_n) \\
&= * \delta_{\rightarrow}(v_1) \cdots \delta_{\rightarrow}(v_n)
\end{aligned}$$

und folglich ist

$$(\delta_{\leftarrow}(u_1), * \delta_{\rightarrow}(v_1)), (\delta_{\leftarrow}(u_2), \delta_{\rightarrow}(v_2)), \dots, (\delta_{\leftarrow}(u_{n-1}), \delta_{\rightarrow}(v_{n-1})), (\delta_{\leftarrow}(u_n), \delta_{\rightarrow}(v_n))$$

eine Lösung von I' . Es bleibt zu zeigen das (p, I) lösbar ist, wenn I' lösbar ist. Sei $\tau : (\Sigma \cup \{*\})^* \rightarrow \Sigma^*$ der Homomorphismus von Sprachen mit $\tau|_{\Sigma} = id_{\Sigma}$ und $\tau(*) = \lambda$. Für $(u', v') \in I'$ gilt $(\tau(u'), \tau(v')) \in I$. Sei $(u'_1, v'_1), \dots, (u'_n, v'_n)$ eine Lösung von I' und $(u'_i, v'_i) = (\tau(u'_i), \tau(v'_i))$ für $i \in [n]$. Es gilt

$$\tau(u'_1) \cdots \tau(u'_n) = \tau(u'_1 \cdots u'_n) = \tau(v'_1 \cdots v'_n) = \tau(v'_1) \cdots \tau(v'_n)$$

und somit ist $(u_1, v_1), \dots, (u_n, v_n)$ eine Lösung von I als Instanz des Postschen Korrespondenzproblems über Σ . Es genügt aber zu zeigen, dass $(u_1, v_1) = p$ gilt. Sei $p' = (\delta_{\leftarrow}(u_1), \delta_{\rightarrow}(v_1))$. Für $(u', v') \in I' / \{p'\}$ gilt $u'(1) \neq v'(1)$, da $(u'_1, v'_1), \dots, (u'_n, v'_n)$ eine Lösung von I' ist gilt also $(u'_1, v'_1) = p'$ und damit $(u_1, v_1) = (\tau(u'_1), \tau(v'_1)) = p$.

3.17 Lemma(Reduktion von H_{init} auf $MPCP_{\square, 0, 1, *, 6, +}$)*

Für jedes Alphabet Σ mit $|\Sigma| \geq 2$ gilt $H_{init} \leq_m MPCP_{\square, 0, 1, *, 6, +}$

Beweis. Wir suchen eine effektive Transformation, die jede natürliche Zahl e auf eine Instanz (p_e, I_e) des modifizierten Postschen Korrespondenzproblems über $\{\square, 0, 1, *, +\}$ abbildet, so dass $\mathcal{M}_e(\lambda) \downarrow$ genau dann gilt, wenn (p_e, I_e) lösbar ist. Sei $e \in \mathbb{N}_0$. Sei Q Die Zustandsmenge und Δ die Übergangsrelation von \mathcal{M}_e . Es gelte also $\mathcal{M}_e = (Q, \Sigma, \Gamma, \Delta, s, F)$ für $\Sigma = \{0, 1\}$, $\Gamma = \{\square, 0, 1\}$, $S = 0$, $F = \{0\}$

Für eine Instanz (p, I) des modifizierten Postschen Korrespondenzproblems über einem Alphabet bezeichnen wir eine Folge $p = (u_1, v_1), \dots, (u_n, v_n)$ für die $u_1 \cdots u_n \sqsubseteq v_1 \cdots v_n$ oder $v_1 \cdots v_n \sqsubseteq u_1 \cdots u_n$ gilt als **partielle Lösung** von (p, I) . Wir wollen (p_e, I_e) so wählen, dass partielle Lösungen von (p_e, I_e) partielle Rechnungen von \mathcal{M}_e entsprechen. Dabei codieren wir eine Konfiguration $(p, w, p) \in Q \times (\Gamma^*)^* \mathbb{N}_0$ von \mathcal{M}_e durch das Wort

$$code(q, w, p) := \#w(1) \cdots w(p-q) * bin(q) * w(p) \cdots w(|w|)\#$$

Im wesentlichen wollen wir erreichen, dass es genau dann für ein Wort w eine partielle Lösung $(u_1, v_1), \dots, (u_n, v_n)$ von (p_e, I_e) mit $w = v_1 \cdots v_n$ gibt, wenn w Präfix der Konkation $code(C_1) \cdots code(C_n)$ der Code der Konfiguration einer partiellen Rechnung C_1, \dots, C_n von \mathcal{M}_e bei Eingabe λ ist. Eine solche partielle Lösung soll genau dann zu einer Lösung von (p_e, I_e) vervollständigt werden können, wenn die durch w beschriebene partielle Rechnung mit einer Stoppkonfiguration endet, also eine Rechnung ist. Dann ist (p_e, I_e) genau dann lösbar, wenn die Rechnung von \mathcal{M}_e zur Eingabe λ endlich ist.

Für $q \in Q$ sei $\hat{q} : *bin(q)$

Als Startpaar sehen wir

$$p_e = (0, 0\# * *\square\#)$$

(die 0en sind nur dafür da da, damit im?"komplment nicht leer ist.) Wir beschreiben nun die Konstruktion von I_e . Für jede Instruktion $(q, a, q', a', L) \in \Delta$ fügen wir folgende Paare ein

$$(\# \hat{q} a, \# \hat{q}' \square a'), (\square \hat{q} a, \hat{q}' \square a'), (0 \hat{q} a, \hat{q}' 0 a') (1 \hat{q} a, \hat{q}' 1 a')$$

ein. Weiter, um unveränderte Infixe kopieren zu können fügen wir die Paare

$$(\#, \#), (0, 0), (1, 1), (\square, \square)$$

ein. Nun brauchen wir noch Paare, die bei Terminierung der TM zu einer validen Instanz der $MPCP$ - Instanz führen.

$\rightsquigarrow \forall q \in Q \forall a \in \{\square, 0, 1\}$ für die es keine Instruktion (q, a, q', a', B) fügen wir das Paar $(\hat{q} a, \dagger a)$ hinzu und auch

$$(\dagger \square, \dagger), (\dagger 0, \dagger), (\dagger 1, \dagger)$$

$$(\square \dagger, \dagger), (0 \dagger, \dagger), (1 \dagger, \dagger) \\ (\# \dagger \# 0, 0)$$

Dies beschreibt die Konstruktion von (p_e, I_e) . Wir verzichten auf die einfache aber aufwändige Verifikation, dass \mathcal{M}_e genau dann bei Eingabe λ terminiert, wenn (p_e, I_e) lösbar ist. \square

3.18 Beispiel(TM M_e mit $e \in \mathbb{N}_0$))*

Sei $e \in \mathbb{N}_0$ mit $\mathcal{M}_e = (\{0, 1\}, \{0, 1\}, \{\square, 0, 1\}, \Delta, 0, \{0\})$, wobei $\Delta = \{(0, \square, 1, 1, R), (1, \square, 1, 1, L)\}$. Mit der Notation aus dem Beweis aus Lemma 3.17 gilt dann [hier bild einfügen!]

3.19 Satz(Unentscheidbarkeit des Post'schen Korrespondenzproblems)*

Für jedes Alphabet Σ mit $|\Sigma| \geq 2$ ist PCP_Σ nicht entscheidbar.

Beweis. Mit Lemma 3.16, Lemma 3.17 und Lemma 3.18 folgt

$$H_{init} \leq_m MPCP_{\square, 0, 1, *, \#, \dagger} \leq_m PCP_{\square, 0, 1, *, \#, \dagger} \leq_m PCP_\Sigma$$

und damit $H_{init} \leq_m PCP_\Sigma$. Folglich ist PCP_Σ nicht entscheidbar, da H_{init} nicht entscheidbar ist. \square

3.20 Fixpunktsatz, Rekursionstheorem und Satz von Rice

Wir beschäftigen uns nun mit weiteren Konsequenzen der Standardaufzählung von TM.

$$\Phi_0, \Phi_1, \Phi_2, \dots$$

Standardaufzählung

$$\Phi_{\Phi_e(0)}, \Phi_{\Phi_e(1)}, \Phi_{\Phi_e(2)}, \dots$$

andere Aufzählung \Rightarrow

$$\Phi_{f(0)}, \Phi_{f(1)}, \Phi_{f(2)}$$

3.20.1 Definition (Fixpunktsatz)

Ein **Fixpunkt** einer berechenbaren Funktion $f : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ ist ein $e \in \mathbb{N}_0$ mit $\Phi_{f(e)} = \Phi_e$.

3.20.2 Satz (Fixpunktsatz, Hartley Rogers jr., 1967)

Alle berechenbaren Funktionen $f : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ haben einen Fixpunkt.

Beweis. $\forall e, x \in \mathbb{N}_0$ mit $\Phi_e(x) \uparrow$ sei $\Phi_{\Phi_e(x)} : \mathbb{N}_0 \rightsquigarrow \mathbb{N}_0$ die apptiell berechenbare partielle Funktion mit $\text{dom}(\Phi_{\Phi_e(x)}) = \emptyset$. Sei e_ψ ein Index von ψ . Gemäß S_n^m -Theorem (Satz 3.7) existiert eine berechenbare Funktion $s_1^1 : \mathbb{N}_0^2 \rightarrow \mathbb{N}_0$ mit $\Phi_{s_1^1(e_\psi, e)}(x) = \psi(e, x)$. $\forall e, x \in \mathbb{N}_0$. Sei $\eta : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ die berechenbare Funktion mit $\eta(e) := s_1^1(e_\psi, e)$. Dann gilt

$$\psi_{\eta(e)}(x) = \psi_{s_1^1(e_\psi, e)}(x) = \psi(e, x) = \Phi_{\Phi_e(e)}(x) \forall x \in \mathbb{N}_0$$

also gilt

$$\Phi_{\eta(e)} = \Phi_{\Phi_e(e)}(*)$$

Sei $e_{f \circ h}$ ein Index der berechneten Funktion $f \circ h$ und $e_{fix} := \eta(e_{f \circ h})$.

$$\Phi_{f(e_{fix})} = \Phi_{f(\eta(e_{f \circ h}))} = \Phi_{\Phi_{e_{f \circ h}}(e_{f \circ h})} \stackrel{(*)}{=} \Phi_{\eta(e_{f \circ h})} = \Phi_{e_\eta}$$

Folglich ist e_{fix} ein Fixpunkt von f . \square

Solche Fixpunkte wie oben sind β emandische"Fixpunkte und kein β yntaktischen"Fixpunkte. Aus dem Fixpunktsatz kann man leicht das Rekursionstheorem folgen, dsa es anschaulich erlaubt während der Konstruktion einer partiell berechenbaren Funktion anzunehmen den Index der fertig definierten Funktion zu kennen. Auf Programmebene bedeutet das, dass es möglich ist ein Programm so zu schreiben, dass der fertige Quellcode im Programm zur Verfügung steht (ohne diesen irgendwo, zum Beispiel vom Speicher des Quellcodes, einzulesen)

3.20.3 Satz (Rekursionstheorem, Stephen Cole Kleen, 1938)

Für alle partielle Funktionen $\varphi : \mathbb{N}_0^2 \rightsquigarrow \mathbb{N}_0$ gibt es ein $e \in \mathbb{N}_0$ mit $\Phi_e(x) = \varphi(e, x) \quad \forall x \in \mathbb{N}_0$

Beweis. Sei e_φ ein index von φ . Gemäß s_n^m - Theorem gibt es eine berechenbare Funktion $s_1^1 : \mathbb{N}_0^2 \rightarrow \mathbb{N}_0$ mit $\Phi_{s_1^1(e_\varphi, e)}(x) = \varphi(e, x) \quad \forall x \in \mathbb{N}_0$ \square

Für Programme bedeutet dies die Existenz von sogenannten **Quines**. Dies sind Programme, die ihren eigenen Quellcode ausgeben (ohne diesen vom Speicher zu lesen). Unsere Resultate zeigen, dass für hinreichend komplexe Programmiersprachen immer Quines existieren. Eine weitere Konsequenz aus dem Fixpunktsatz ist die Einsicht, dass jede nicht triviale Programmiereigenschaft unentscheidbar ist.

3.20.4 Korollar (Existenz eines fixierten Punktes in der berechenbaren Funktionenfamilie)*

Es gibt ein $e \in \mathbb{N}_0$ mit $\Phi_e(x) = e \quad \forall x \in \mathbb{N}_0$.

Beweis. Sei $\psi : \mathbb{N}_0^2 \rightsquigarrow \mathbb{N}_0$ die partiell berechenbare Funktion mit $\psi(e, x) = e \quad \forall e, x \in \mathbb{N}_0$. Gemäß [Satz 3.20.2](#) gibt es nun ein $e \in \mathbb{N}_0$ mit $\Phi_e(x) = \psi(e, x) = e \quad \forall x \in \mathbb{N}_0$ \square

3.20.5 Definition (Indexmenge)

Eine Teilmenge $I \subseteq \mathbb{N}_0$ heißt Indexmenge, wenn $e \in I \Leftrightarrow e' \in I \quad \forall e, e' \in \mathbb{N}_0$ mit $\Phi_e = \Phi_{e'}$ gilt.

3.20.6 Satz (Satz von Rice, Henry Horden Rice, 1951)

Ist I eine Indexmenge $\emptyset \neq I \neq \mathbb{N}_0$, so ist I nicht entscheidbar.

Beweis. Sei $e_0 \notin I, e_1 \in I$ und sei $f : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ die Funktion mit $f(e) = e_0 \quad \forall e \in I$ und $f(e) = e_1 \quad \forall e \in \mathbb{N}_0/I$. (Ist I entscheidbar dann ist f offensichtlich berechenbar.) $\forall e \in \mathbb{N}_0$ gilt $f(e) \in I \Leftrightarrow e \notin I$ und da I eine Indexmenge ist ist somit $\Phi_{f(e)} \neq \Phi_e$. Die Funktion f hat also keinen Fixpunkt. Wäre I entscheidbar, so hätte f aber einen Fixpunkt nach dem [Fixpunktsatz](#). \square

Automaten

Imagine you're in a city with a limited number of locations (like a park, library, cafe, etc.). You can move from one place to another following specific paths (like roads). The paths you take depend on some rules, like the time of the day, or the type of ticket you have. The places you can reach with these rules represent different states in a finite automaton, and the rules themselves act like the transition function.

- ChatGPT

Wir wollen Turingmaschinen nun stark einschränken. Wir betrachten ein Modell, das im wesentlichen ohne Speicher zurechtkommt (=TM ohne Band \rightarrow brauchen es nur für die Eingabe). Der Ausgabemechanismus kennt nur Akzeptanz und Nichtakzeptanz. Als TM kann der wie folgt realisiert werden:

- Es ist nur ein Band erlaubt.
- Bei jedem Rechenschritt bewegt sich der Kopf nach rechts. Ob und wie die Felder des Bandes dabei überschreiben werden spielt dann keine Rolle, denn der Kopf kann nie zurück bewegt werden; wir legen aber fest, dass Symbole nicht überschrieben werden. Die Symbole des Bandalphabet Γ neben denen des Eingabealphabets Σ und des \square Symbols hat spielen keine Rolle. Wir legen hier $\Gamma = \Sigma \cup \{\square\}$ fest.
- Beim Einlesen des ersten \square Symbols muss die Rechnung der Maschine enden. Wir soll die Rechnung nicht vor dem Einlesen des ersten \square Symbols enden.

Dies bedeutet, dass wir TM $M = (Q, \Sigma, \Sigma \cup \{\square\}, \Delta, s, F)$ die nur Instruktionen der Form (q, a, q', a, R) mit $q \in Q$ und $a \in \sigma$ hat. Dies sind nun stark eingeschränkte TM. Wir wählen eine äquivalente Form, die als endliche Automaten bezeichnet werden.

Hinweis: Durch die Einschränkungen wird die Leistungsfähigkeit der Turingmaschine erheblich reduziert. Sie kann nun nicht mehr so allgemeine Berechnungen durchführen wie eine herkömmliche Turingmaschine.

4.1 Definition (Endliche Automaten)

Ein endlicher Automat, kurz EA, ist ein Tupel $A = (Q, \Sigma, \Delta, s, F)$. Dabei ist

- Q eine endliche Menge, der Zustandsmenge;
- Σ das Eingabealphabet;
- $\Delta \subseteq Q \times \Sigma \times Q$ die Übergangsrelation, eine relation, so dass es für alle $q \in Q$ und $a \in \Sigma$ ein $q' \in Q$ mit (q, a, q') ;
- $s \in Q$ der Startzustand;
- $F \subseteq Q$ die Menge der akzeptierten Zustände.

Der endliche Automat A ist ein deterministischer endlicher Automata, kurz DEA, wenn es $\forall (q, a) \in Q \times \Sigma$ genau ein q' gibt mit $(q, a, q') \in \Delta$. Im Sinne der obigen Betrachtung entspricht ein EA $A = (Q, \Sigma, \Delta, s, F)$ der 1-TM $M_A = (Q, \Sigma, \Sigma \cup \{\square\}, \{(q, a, q', a, R) : (q, a, q') \in \Delta\}, s, F)$.

\rightsquigarrow Band spielt keine wesentliche Rolle, Zustände mit gerade gelesenen Symbol bilden die Konfigurationen.

4.2 Definition (Übergangsfunktion eines EA)

Sei $A = (Q, \Sigma, \Delta, s, F)$ ein EA. Die **Übergangsfunktion** von A ist die Funktion $\delta_A : Q \times \Sigma \rightarrow 2^Q$ ⁴ mit

$$\delta_A(q, a) = \{q' \in Q : (q, a, q') \in \Delta\} \quad \forall q \in Q, a \in \Sigma$$

erweiterter **Übergangsfunktion** von A ist die Funktion

$$\delta_A^* : Q \times \Sigma^* \rightarrow 2^Q \quad \delta_A^*(q, \lambda) = \{q\}$$

und

$$\delta_A^*(q, aw) = \bigcup_{q' \in \delta_A(q, a)} \delta_A^*(q', w) \quad \forall q \in Q, a \in \Sigma$$

und $w \in \Sigma^*$. Für $Q_0 \subseteq Q$ und $w \in \Sigma^*$ schreiben wir $\delta_A^*(Q_0, w)$ statt $\bigcup_{q \in Q_0} \delta_A^*(q, w)$.

Für einen EA $A = (Q, \Sigma, \Delta, s, F)$, mit entsprechender TM $M_A = (Q, \Sigma, \Gamma, \Delta', s, F)$, $q \in Q$ und $w \in \Sigma^*$ ist $\delta_A^*(s, w)$ die Menge der Zustände, die sich als erst Komp.?? der letzten Konfig einer Rechnung von M_A zur Eingabe zu ergeben.

⁴(=Potenzmenge von Q)

4.3 Bemerkung (Eigenschaften von endlichen Automaten)*

Sei $A = (Q, \Sigma, \Delta, s, F)$ ein EA

- (i) $\forall q \in Q$ und $a \in \Sigma$ gilt $\delta_A^*(q, a) = \delta_A(q, a)$.
- (ii) Ist A ein DEA, $q \in Q$, $a \in \Sigma$ und $w \in \Sigma^*$, und $|\delta_A^*(q, w)| = 1$.
- (iii) Seien $u, v \in \Sigma^*$ $\forall q \in Q$ gilt $\delta_A^*(q, uv) = \delta_A^*(\delta_A^*(Q_0, u), v)$.

4.4 Definition (Übergangsfunktion eines DEA)

Sei $A = (Q, \Sigma, \Delta, s, F)$ eine DEA. Auch die Funktion $\delta_{det,A} : Q \times \Sigma \rightarrow Q$ mit $\delta_A(q, a) = \{\delta_{det,A}(q, a)\} \quad \forall q \in Q$ und $a \in \Sigma$ wird auch **Übergangsfunktion** von A genannt. Analoges gilt für $\delta_{det,A}^*(Q_0, w)$ statt $\bigcup_{q \in Q_0} \{\delta_{det,A}^*(q, w)\}$.

4.5 Bemerkung (Folgerungen für DEA)*

Ist $A = (Q, \Sigma, \Delta, s, F)$ ein DEA, so gelten [Bemerkung 4.3 \(i\)](#) und [\(iii\)](#) auch wenn δ_A durch $\delta_{det,A}$ und δ_A^* durch $\delta_{det,A}^*$ ersetzt wird.

4.6 Bemerkung (Eindeutigkeit endlicher Automaten)*

Sei Q eine endliche Menge, Σ ein Alphabet, $s \in Q$, und $F \subseteq Q$.

- (i) \forall Funktionen $\delta : Q \times \Sigma \rightarrow 2^Q$ gibt es genau einen EA $A = (Q, \Sigma, \Delta, s, F)$ mit $\delta_A = \delta$.
- (ii) \forall Funktionen $\delta : Q \times \Sigma \rightarrow Q$ gibt es genau einen $\delta_{det,A} = \delta$.

4.7 Definition (akzeptierte Sprache)

Sei $A = (Q, \Sigma, \Delta, s, F)$ ein EA. Die Sprache $L(A) := \{w \in \Sigma^* : \delta_A^*(s, w) \cap F \neq \emptyset\}$ ist die **akzeptierte Sprache** von A.

4.8 Definition (regulär)

Eine Sprache L heißt **regulär** wenn es einen EA A mit $L(A) = L$ gibt. Wir schreiben **REG** für die Klasse der regulären Sprachen. Zu jedem Zeitpunkt während der Verbindung der Eingabe durch einen endlichen Automaten hängt der restliche Bearbeitung immer nur vom gegenwärtigen Zustand und dem noch einzulesenden Teil der Eingabe ab, nicht aber wie bei TM im allgemeinen von vergangenen Bandmanipulation. Interpretiert man die Eingabe als von einer äußeren Quelle kommend, so ist der Zustand des Automaten also allein durch seinen Zustand gegeben und der nächste Zustand hängt nur vom Zugeführten Symbol ab. Daher bietet sich eine Darstellung eines EA durch ein Übergangsdiagramm oder eine sogenannte Übergangstabelle an.

4.9 Beispiel (Endlicher Automat A)*

Sei $A := (\{q_0, q_1\}, \{0, 1\}, \Delta, q_0, \{q_1\})$ mit $\Delta = \{(q_0)\}$. Das Übergangsdiagramm und die Übergangstabelle sehen wie folgt aus:

Zustand/Symbol	0 ¹	1 ²
\rightarrow ³ q_0	q_0	q_1
q_1 , ⁴ *	q_1 ⁵	q_0

1. die Elemente von Σ
2. die Elemente von Σ
3. Startzustand
4. Zustand $\in F$
5. $(q_1, 0, a_1) \in \Delta$ (wenn a in q_1 ist und 0 einliest, geht A in q_1 über)

Abbildung 9: Übergangstabelle

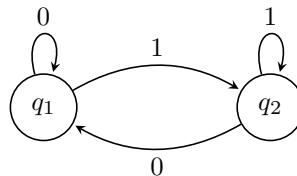


Abbildung 10: Übergangsdiagramm

Übergangsdiagramm: Für jeden Zustand gibt es einen Kreis. Zustände in F bekommen einen Doppelkreis. Für $(q, a, q') \in \Delta$ für einen Pfeil von dem Kreis von q zu dem Kreis von q' mit der Beschreibung a . Zusätzlich gibt es einen Pfeil (ohne Beschriftung) aus dem "Nichtsbus deom Kreis des Starzustandes. Ähnlich wie bei allgemeinen und normierten TM bleibt die Klasse der akzeptierten Sprachen gleich wenn man nur deterministisch endliche Automaten zulässt. Um dies zu beweisen führen wir den Potentautomaten ein.

4.10 Definition (Potenzautomaten)

Sei $A = (Q, \Sigma, \Delta, s, F)$ ein EA. der **Potenzautomat** von A ist der DEA $P_A = (2^Q, \Sigma, \Delta', \{s\}, \{P \subseteq Q : P \cap F \neq \emptyset\})$ mit

$$\delta_{det, P_A}(Q_0, a) = \bigcup_{q \in Q_0} \delta_A(q, a) \quad \forall Q_0 \subseteq Q \quad \forall a \in \Sigma$$

Anmerkung: Es gibt eine einfache Möglichkeit einen nicht Deterministischen Automaten in einen Deterministischen umzuwandeln. Das wird hier in Zukunft beschrieben. Siehe Tutoriumaufschrieb. (das wird hier in Zukunft angefügt)

4.11 Satz(Charakterisierung regulärer Sprachen)*

Eine Sprache L ist genau dann regulär, wenn es eine DEA A mit $L(A) = L$ gibt.

Beweis. Sei $A = (Q, \Sigma, \Delta, s, F)$ ein EA mit Potenzautomat P_A . Es genügt zu zeigen, dass $L(A) = L(P_A)$. Hierfür genügt es zu zeigen, dass:

$$\delta_{det, P}^* = \delta_A^*(s, w) \quad \forall w \in \Sigma^* \circledast$$

Denn damit folgt

$$\begin{aligned} w \in L(P_A) &\Leftrightarrow \delta_{P_A}^*(\{s\}, w) \cap \{P \subseteq Q : P \cap F \neq \emptyset\} \neq \emptyset \\ &\Leftrightarrow \delta_{det, P_A}^*(\{s\}, w) \cap F \neq \emptyset \\ &\Leftrightarrow \delta_A^*(\{s\}, w) \cap F \neq \emptyset \\ &\quad \circledast \\ &\Leftrightarrow w \in L(A) \end{aligned}$$

Wir zeigen \circledast mittels vollständiger Induktion über $|w|$. Es gilt $\delta_{det, P_A}^*(\{s\}, \lambda) = \delta_A^*(s, \lambda)$. Sei $w \in \Sigma^+$ mit $\delta_{det, A}^*(\{s\}, v) = \delta_A^*(s, v) \quad \forall v \in \Sigma^{\leq |w|-1}$. Nun zeigen wir \circledast Sei $va := w$ mit $a \in \Sigma$ und $|v| = |w| - 1$.

$$\begin{aligned} \delta_{det, P_A}^*(\{s\}, w) &\stackrel{\text{Bem 4.5}}{=} \delta_{det, P_A}^*(\delta_{det, P_A}^*(\{s\}, v), a) \\ &\stackrel{\text{Ind. hyp}}{=} \delta_{det, P_A}^*(\delta_{det, P_A}^*(\{s\}, v), a) \\ &= \bigcup_{q \in \delta_{det, A}^*(\{s\}, v)} \delta_A(q, a) \\ &= \delta_A^*(\delta_A^*(s, v), a) \\ &= \delta_A^*(s, va) \\ &= \delta_A^*(s, w) \end{aligned}$$

□

Reguläre Sprachen

A regular language can be thought of as a collection of sentences in a secret code. This secret code has a set of rules that determine which sentences are valid. You can think of it like a secret handshake, where only certain movements are allowed to be performed in a particular order.

- ChatGPT

5.1 Definition (Äquivalenzrelation)

Sei A eine Menge. Eine Äquivalenzrelation auf A ist eine Relation $\sim \subseteq A^2$, so dass die folgende Eigenschaft erfüllt sind. (wie bei Relationen üblich verwenden wir Infixnotation)

- (i) $a \sim a \quad \forall a \in A$ (Reflexivität)
- (ii) $a \sim b \Rightarrow b \sim a \quad \forall a, b \in A$ (Symmetrie)
- (iii) $a \sim b, b \sim c \rightarrow a \sim c \quad \forall a, b, c \in A$ (Transitivität)

Die **Äquivalenzklasse** eines Elements $a \in A$ bezüglich \sim ist die Menge $[a] := \{a' \in A : a' \sim a\}$. Der **Index** von \sim ist die Kardinalität der Menge $A/\sim := \{[a] : a \in A\}$ falls diese endlich ist und ∞ andernfalls.

Erklärung: Die Anzahl der Elemente in einer Äquivalenzklasse, und der Index von \sim gibt die Kardinalität der Menge A/\sim an, wenn sie endlich ist, oder ∞ , wenn sie unendlich ist. ⁵

Beispiel:

- In der Menge der ganzen Zahlen könnte die Äquivalenzrelation 'Teilbarkeit' sein, bei der $a \sim b$ bedeutet, dass a ohne Rest durch b teilbar ist.
- In der Menge der Personen könnte die Äquivalenzrelation "gleiches Alter" sein, bei der $a \sim b$ bedeutet, dass a und b das gleiche Alter haben. ⁶

5.2 Definition (A-Äquivalenz)

Sei $A = (Q, \Sigma, \Delta, s, F)$ ein DEA mit erweiterter Übergangsfunktion $\delta^* : Q \times \Sigma^* \rightarrow Q$. Die A-Äquivalenz ist die Relation \sim_A auf Σ^* mit

$$u \sim_A v \Leftrightarrow \delta^*(s, u) = \delta^*(s, v) \quad \forall u, v \in \Sigma^*$$

5.3 Bemerkung

Sei $A = (Q, \Sigma, \Delta, s, F)$ ein DEA.

- (i) Die A-Äquivalenz ist eine Äquivalenzrelation.
- (ii) Der Index von \sim_A ist höchstens $|Q|$.
- (iii) Es gilt $L(A) = \bigcup_{w \in L(A)} [w]_{\sim_A}$.

5.4 Definition (Rechtskongruenz)

Sei Σ ein Alphabet. Eine Rechtskongruenz auf Σ^* ist eine Äquivalenzrelation $\sim \subseteq (\Sigma^*)^2$ mit $u \sim v \Rightarrow uw \sim vw \quad \forall u, v, w \in \Sigma^*$.

5.5 Proposition

Sei $A = (Q, \Sigma, \Delta, s, F)$ ein DEA. Die A-Äquivalenz \sim_A ist eine Rechtskongruenz auf Σ^* .

Beweis. Seien $u, v, w \in \Sigma^*$ mit $u \sim_A v$. Dann gilt

$$\begin{aligned} \delta_{det,A}^*(s, uw) &= \delta_{det,A}^*(\delta_{det,A}^*(s, u), w) = \delta_{det,A}^*(\delta_{det,A}^*(s, v), w) \\ &= \delta_{det,A}^*(s, vw). \end{aligned}$$

(hier benutzen wir [Bemerkung 4.3](#) und [Bemerkung 4.5](#)) □

⁵exp. Ed. Sug.

⁶Textbsp. Ed. sug.

Dann gilt $uw \sim_A vw$. Zu jedem DEA A gibt es also eine dazugehörige Rechtskongruenz \sim auf Σ^* mit endlichem Index so dass $L(A)$ die Vereinigung von Äquivalenzklasse von \sim_A ist. Tatsächlich gilt auch die Umkehrung: Ist L die Vereinigung von Äquivalenzklasse einer Rechtskongruenz \sim mit endlichem Index, so gibt es einen DEA A mit $L(A) = L$

5.6 Definition(DEA-Konstruktion für Äquivalenzklassen)

Sei Σ eine Alphabet und L Vereinigung von Äquivalenzklasse einer Rechtskongruenz \sim auf Σ^* mit endlichem Index. Es bezeichne

$$A_{\sim,L} := (\Sigma_{/\sim}^*, \Sigma, \Delta, [\lambda]_{\sim}, [w]_{\sim} : w \in L)$$

den DEA mit $\delta_{det,A_{\sim,L}}([w]_{\sim}, a) = [wa]_{\sim} \forall w \in \Sigma^*$ und $a \in \Sigma$. Die Wohldefiniertheit von $\delta_{det,A_{\sim,L}}$ ergibt sich daraus, dass \sim eine Rechtskongruenz ist. Um uns davon zu überzeugen, dass $L(A_{\sim,L}) = L$ gilt betrachten wir zunächst die Arbeitsweise von $A_{\sim,L}$.

5.7 Lemma

Sei Σ ein Alphabet, L Vereinigung von Äquivalenzklassen einer Rechtskongruenz \sim auf Σ^* mit endlichem Index und sei $\delta^* : \Sigma_{/\sim}^* \times \Sigma^* \rightarrow \Sigma_{/\sim}^*$ die erweiterte Übergangsfunktion von $A_{\sim,L}$. Dann gilt $\delta^*([\lambda]_{\sim}, w) = [w]_{\sim} \forall w \in \Sigma^*$.

Beweis. Wir verwenden vollständige Induktion über $|w|$. Es gilt $\delta^*([\lambda]_{\sim}, \lambda) = [\lambda]_{\sim}$. Sei nun $w \in \Sigma^+ \dots$ □

5.8 Satz

Sei L die Vereinigung von Äquivalenzklasse einer Rechtskongruenz \sim mit endlichem Index. Es gibt $L(A_{\sim,L}) = L$

Beweis. Sei Σ das Alphabet, so dass \sim eine Rechtskongruenz auf Σ^* ist. Sei $\delta^* : \Sigma_{/\sim}^* \times \Sigma^* \rightarrow \Sigma_{/\sim}^*$ die erweiterte Übergangsfunktion von $A_{\sim,L}$ und sei $w \in \Sigma^*$. Aus [Lemma 5.5](#) folgt.

$$\begin{aligned} w \in L(A_{\sim,L}) &\Leftrightarrow \delta^*([\lambda]_{\sim}, w) \in [v]_{\sim} : v \in L \\ &\Leftrightarrow [w]_{\sim} \in [v]_{\sim} : v \in L \\ &\Leftrightarrow \exists v \in L : [w]_{\sim} = [v]_{\sim} \\ &\Leftrightarrow \exists v \in L : w \sim v \\ &\Leftrightarrow w \in L \end{aligned}$$

□

5.9 Korollar

Eine Sprache L ist genau dann regulär, wenn sie die Vereinigung von Äquivalenzklasse einer Rechtskongruenz mit endlichem Index ist.

Beweis. Folgt aus [Bemerkung 5.3](#), [Proposition 5.5](#) und [Satz 5.8](#) □

Betrachten man nur deterministische endliche Automaten ohne unerreichbare Zustände, so entsprechen diese bis auf Unbenutzung von Zuständen sogar den Rechtskongruenz mit endlichem Index zusammen mit Vereinigung von Äquivalenzklassen dieser.

5.10 Definition(erreichbar)

Sei Σ ein Alphabet. Sei $A = (Q, \Sigma, \Delta, s, F)$ ein EA mit erweiterter Übergangsfunktion δ^* . Ein Zustand $q \in Q$ heißt erreichbar in A wenn es ein Wort $w \in \Sigma^*$ mit $q \in \delta^*(s, w)$ gilt.

5.11 Definition(isomorph)

Sei $A_i = (Q_i, \Sigma, \Delta_i, s_i, F_i)$ für $i \in 1, 2$ ein EA mit Übergangsfunktion δ_i . Die endliche Automaten A_1 und A_2 sind **isomorph**, kurz $A_1 \cong A_2$, wenn es eine Projektion $f : Q_1 \rightarrow Q_2$ gibt, sodass folgendes gilt:

- (i) $f(s_1) = s_2$
- (ii) $\delta_2(f(q_1), a) = f(\delta_1(q_1, a))$
- (iii) $f(F_1) = F_2$

5.12 Satz

- (i) Ist A eine DEA ohne un erreichbare Zustände, so gilt $A \cong A_{\sim A, L(A)}$
- (ii) Ist L die Vereinigung von Äquivalenzklasse einer Rechtskongruenz \sim mit endlichem Index, so gilt $(\sim, L) = (\sim_{A_{\sim, L}, L(A_{\sim, L})})$.

Beweis. (i) Sei $A = (Q, \Sigma, \Delta, s, F)$ eine DEA mit erweiterte Übergangsfunktion $\delta^* : Q \times \Sigma^* \rightarrow Q$ ohne un erreichbare Zustände, $\sim := \sim_A$, $A' := A_{\sim, L(A)}$ und sei $\delta' : \Sigma^* / \sim \times \Sigma^* \rightarrow \Sigma^* / \sim$ die erweiterte Übergangsfunktion von A' . Sei $f : Q \rightarrow \Sigma^* / \sim$ die Bijektive mit $f(q) := \{w \in \Sigma^* : \delta^*(s, w) = q\}$. Es gelte $f(s) = [\lambda]_{\sim}$ und $f(F) = \{[w]_{\sim} : w \in L(A)\}$. Es genügt somit zu zeigen, dass $\delta'(f(q), a) = f(\delta(q, a)) \forall q \in Q, a \in \Sigma$. Sei $q \in Q, a \in \Sigma^*$. Es genügt $w \in \delta'(f(q), a) \Leftrightarrow \delta^*(s, w) = \delta^*(q, a)$ zu zeigen. Sei $v \in \Sigma^*$ mit $\delta^*(s, v) = q$. Nun gilt $w \in \delta'(f(q), a) \Leftrightarrow w \in \delta'([v]_{\sim}, a) \Leftrightarrow w \sim va \Leftrightarrow \delta^*(s, w) = \delta^*(s, va) \Leftrightarrow \delta^*(s, w) = \delta^*(q, a)$

fügehierbeidemletzetnpfeilnoch" Bem4.3und4.5unterdenpfeilhinzu"

- (ii) Sei Σ ein Alphabet, \sim eine Rechtskongruenz auf Σ^* , L Vereinigung von Äquivalenzklassen von \sim , $A' := A_{\sim, L} = (\Sigma^* / \sim, \Sigma, A', [\lambda]_{\sim}, \uparrow, \delta' : \Sigma^* / \sim \times \Sigma^* \rightarrow \Sigma^* / \sim)$ die erweiterte Übergangsfunktion von. Nach [Satz 5.8](#) gilt $L = L(A')$, es genügt also $\sim = \sim'$ zu zeigen. Sei $u, v \in \Sigma^*$. Aus [Lemma 5.7](#) folgt $u \sim v \Leftrightarrow [u]_{\sim} = [v]_{\sim} \Leftrightarrow \delta'(\dots) \dots$

□

Satz 5.12 Bedeutet insbesondere folgendes: Ist $A_i, i \in \{1, 2\}$ ein DEA ohne un erreichbare Zustände, so gilt $A_1 \cong A_2 \Leftrightarrow (\sim_{A_1}, L(A_1)) = (\sim_{A_2}, L(A_2))$ und ist L_i für $i \in \{1, 2\}$. Vereinigung von Äquivalenzklassen einer Rechtskongruenz \sim_i mit endlichem Index, so gilt $(\sim_1, L_1) = (\sim_2, L_2) \Leftrightarrow A_{\sim_1, L_1} \cong A_{\sim_2, L_2}$.

Ist L eine reguläre Sprache, so gibt es verschiedene endliche Automaten (ohne un erreichbare Zustände) mit $L(A) = L$. Äquivalenzklassen verschiedener Rechtskongruenz mit endlichem Index. Für alle solche Rechtskongruenz \sim und $\forall u, v, w \in \Sigma^*$ mit $u \sim v$ gilt aber

$$uw \in L \Leftrightarrow \delta_{det, A}^*(s, uw) \in F \Leftrightarrow \delta_{det, A}^*(s, vw) \in F \Leftrightarrow vw \in L$$

Dies führt zum Begriff der L -Äquivalenz und zeigt, dass die Partition in die Äquivalenzklassen von \sim Vereinfacht der Partition in die Äquivalenzklasse der L -Äquivalenz ist.

5.13 Definition (L-Äquivalenz)

Sei L eine Sprache über einem Alphabet Σ . Die **L-Äquivalenz** von L als Sprache ist die Relation \sim_L auf Σ^* mit

$$u \sim_L v \Leftrightarrow (uw \in L \Leftrightarrow vw \in L \quad \forall w \in \Sigma^*)$$

5.14 Bemerkung

Sei L eine Sprache über Σ .

- (i) Die L -Äquivalenz ist eine Rechtskongruenz.
- (ii) Es gilt $L = \bigcup_{w \in L} [w]_{\sim_L}$.

5.15 Definition (Partion)

Sei A eine Menge. Eine Partion von A ist eine Menge $\mathcal{A} = A_1, \dots, A_n$ paarweise disjunkt nichtleere Teilmengen von A mit $\bigcup_{i \in [n]} A_i = A$.

5.16 Definition (Verfeinerung)

Seien \mathcal{A}_1 und \mathcal{A}_2 partionen einer Menge A . Die Partion \mathcal{A}_2 **Verfeinert** \mathcal{A}_1 (heißt Verfeinerung von \mathcal{A}_1), wenn es $\forall A_2 \in \mathcal{A}_2$ ein $A_1 \in \mathcal{A}_1$, mit $A_2 \subseteq A_1$ gibt.

5.17 Bemerkung

Seien \mathcal{A}_1 und \mathcal{A}_2 Partionen einer Menge A , so dass \mathcal{A}_2 die Partion \mathcal{A}_1 verfeinert.

- (i) $\forall A' \in \mathcal{A}_1$, gibt es eine Teilmengen $\mathcal{A}'_2 \subseteq \mathcal{A}_2$, die eine Partion von A' ist.
- (ii) Es gilt $|\mathcal{A}_1| \leq |\mathcal{A}_2|$
- (iii) Gilt $|\mathcal{A}_1| = |\mathcal{A}_2|$ dann ist $\mathcal{A}_1 = \mathcal{A}_2$

5.18 Proposition

Sei Σ eine Alphabet und L eine Sprache über Σ und \sim eine Rechtskongruenz auf Σ^* mit $L = \bigcup_{w \in L} [w]_{\sim}$. Die Partion Σ^* / \sim ist eine Verfeinerung der partition Σ^* / \sim_L .

Beweis. Seien $u, v \in \Sigma^*$ mit $u \sim v$. Es genügt zu zeigen, dass $u \sim_L v$. Sei $w \in \Sigma^*$. Es genügt $uw \in L \leftrightarrow vw \in L$ zu zeigen. Da \sim eine Rechtskongruenz ist folgt $uw \sim vw$. Ist $u, w \in L$, so folgt aus $L = \bigcup_{w' \in L} [w']_{\sim}$ auch $vw \in L$ (analog folgt auch $vw \in L \Rightarrow uw \in L$).
 $\Rightarrow u \sim_L v$. □

Das heißt \sim_L ist die größte Partion, die L darstellen kann.

5.19 Definition (Minimalautomat)

Sei L eine reguläre Sprache über Σ . Der Minimalautomat von L als Sprache über Σ ist der DEA $A_{\sim_L, L}$.

5.20 Satz(Charakterisierung regulärer Sprachen)*

Sei L eine reguläre Sprache über Σ und sei $M(Q, \Sigma, \Delta, s, F)$ der Minimalautomat von L . Dann gilt:

- (i) $L(M) = L$
- (ii) Ist A ein DEA mit Zustandsmenge Q_A und $L(A) = L$, so gilt $|Q_A| \geq |Q|$.
- (iii) Ist A ein DEA mit $|Q|$ Zuständen und $L(A) = L$, so gilt $A \cong M$.

Beweis. (i) Folgt direkt aus [Satz 5.8](#)

- (ii) Aus [Bemerkung 5.3 \(ii\)](#) folgt $|Q_A| \geq |\Sigma^* / \sim_A|$. Nach [Proposition 5.18](#) ist Σ^* / \sim_A eine Verfeinerung von Σ^* / \sim_L , nach [Bemerkung 5.17 \(ii\)](#) gilt also $|\Sigma^* / \sim_A| \geq |\Sigma^* / \sim_L|$. Wegen $|\Sigma^* / \sim_L| = |Q|$ folgt somit

$$|Q_A| \geq |\Sigma^* / \sim_A| \geq |\Sigma^* / \sim_L| = |Q|$$

- (iii) Sei A ein DEA mit $|Q|$ Zuständen und $L(A) = L$. Hätte A un erreichbare Zustände, so folgt $|\Sigma^* / \sim_A| < |Q| = |\Sigma^* / \sim_L|$ im Widerspruch zu [Bemerkung 5.17 \(ii\)](#) und [Proposition 5.18](#)

Nach [Satz 5.12](#) genügt es zu zeigen, dass $\sim_A = \sim_M$ zu zeigen. Die Relationen \sim_M und \sim_A sind nach [Bemerkung 5.3](#) und [Proposition 5.5](#) Rechtskongruent mit endlichem Index und L ist Verfeinerung von Äquivalenzklassen davon. Damit sind Σ^* / \sim_A und Σ^* / \sim_M nach [Proposition 5.18](#) Verfeinerungen von Σ^* / \sim_L . Somit sind die Indices von \sim_A und \sim_M mindestens so groß wie der Index von \sim_L . Weiter sind die Indices von \sim_A und \sim_M nach [Bemerkung 5.3 \(ii\)](#) aber auch höchstens so groß wie $|Q| = |\Sigma^* / \sim_L|$. Die Indices von \sim_A , \sim_M und \sim_L sind alle gleich groß. Da Σ^* / \sim_A und Σ^* / \sim_M Verfeinerungen von Σ^* / \sim_L sind, folgt mit [Bemerkung 5.17 \(ii\)](#) somit $\Sigma^* / \sim_A = \Sigma^* / \sim_M = \Sigma^* / \sim_L$ und $\sim_A = \sim_L = \sim_M$. \square

Unsere bisherigen Betrachtungen erlauben verschiedene Äquivalenten Charakterisierungen der Klasse der regulären Sprachen.

5.21 Satz (Satz von Myhill und Nerode)

Für eine Sprache L über einem Alphabet Σ sind die folgenden Aussagen äquivalent:

- (i) L ist regulär.
- (ii) Der Index von \sim_L ist endlich.
- (iii) L ist die Vereinigung von Äquivalenzklasse einer Rechtskongruenz mit endlichem Index.

Beweis. (i) \Leftrightarrow (iii) ist die Aussage von [Korollar 5.9](#). Die Relation \sim_L ist nach [Bemerkung 5.14](#) eine Rechtskongruenz und es gilt $L = \bigcup_{w \in L} [w]_{\sim_L}$. Somit folgt (i) \Rightarrow (iii). Die Implikation (iii) \Rightarrow (ii) folgt aus [Bemerkung 5.17\(ii\)](#) und [Proposition 5.19](#). \square

Wie wollen nun ein Kriterium beschreiben das hilft nicht reguläre Sprachen zu erkennen.

5.22 Satz (Pumping-Lemma)

Sei Σ ein Alphabet. Für jede reguläre Sprache $L \subseteq \Sigma^*$ gibt es eine Konstante $k \in \mathbb{N}$, so dass folgendes gilt: Ist $z \in L$ mit $|z| \geq k$, so gibt es Wörter u, v, w mit $z = uvw$, so dass folgendes gilt:

- (i) $v \neq \lambda$
- (ii) $|uv| \leq k$
- (iii) $uv^i w \in L \forall i \in \mathbb{N}_0$ (?ist das w noch in dem Wort oder ist es ausserhalb aber in L ?)

Beweis. Sei $L \subseteq \Sigma^*$ eine reguläre Sprache und $A = (Q, \Sigma, \Delta, s, F)$ ein DEA mit $L(A) = L$ und erweiterter Übergangsfunktion $\delta^* : Q \times \Sigma^* \rightarrow Q$. Sei $k := |Q|$. Sei $z \in L$ mit $|z| \geq k$. (Falls kein solches z existiert ist nichts zu zeigen). Die Funktion $f : \{0, \dots, k\} \rightarrow Q, i \mapsto \delta^*(s, z(1) \dots z(i))$ ist keine Injektion, denn es gibt $|\{0, \dots, k\}| = k + 1 > |Q|$. Seien $j_1, j_2 \in \{0, \dots, k\}$ mit $j_1 \neq j_2$ und $f(j_1) = f(j_2)$. Sei $u := z(1) \dots z(j_1)$, $v = z(j_1 + 1) \dots z(j_2)$, $w = z(j_2 + 1) \dots z(|z|)$. Dann gilt $z = uvw$. Aus $j_1 < j_2$ folgt $v \neq \lambda$. Aus $j_2 \leq k$ folgt $|uv| \leq k$. Es bleibt zu zeigen, dass $uv^i w \in L \forall i \in \mathbb{N}_0$ gilt. Dafür genügt es zu zeigen $\delta^*(s, uv^i) = \delta^*(s, u) \otimes$. Denn dann gilt mit [Bemerkung 4.3\(iii\)](#) und [Bemerkung 4.5](#)

$$\delta^*(s, uv^i w) = \delta^*(\delta^*(s, uv^i), w) = \delta^*(\delta^*(s, u), w) = \delta^*(\delta^*(s, uv), w) = \delta^*(s, uvw)$$

und damit $uv^i w \in L$. Wir zeigen \otimes mittels vollständiger Induktion über i . $i = 0$. Gelte nun $\delta^*(s, uv^{i-1}) = \delta^*(s, u)$ für ein $i \in \mathbb{N}$. Wieder folgt

$$\delta^*(s, uv^i) = \delta^*(\delta^*(s, uv^{i-1}), v) \stackrel{\text{I.H.}}{=} \delta^*(\delta^*(s, u), v) = \delta^*(s, uv) = f(j_2) = f(j_1) = \delta^*(s, u).$$

\square

Beispiel Die Sprache $L = \{0^n i^n : n \in \mathbb{N}_0\}$ ist nicht regulär. Dies lässt sich mit dem Pumping-Lemma wie folgt zeigen.

Beweis. Angenommen L wäre regulär.

$\Rightarrow \exists k \in \mathbb{N}_0 : \forall z \in L$ mit $|z| \geq k$ gilt, $\exists u, v, w \in \Sigma^*$ mit $z = uvw$ und (i) $v \neq \lambda$ (ii) $|uv| \leq k$ (iii) $uv^i w \in L \forall i \in \mathbb{N}_0$.
Sei $z := 0^k 1^k$.

Aus (i) und (ii) folgt, dass $v = 0^l$ für $l > 0$ und damit folgt $uv = 0^{k+l} 1^k$

□

Formale Grammatiken

- *ChatGPT*

Idee: Konstruktion aller Wörter einer Sprache.

Beispiel:

$$L = \{0^n\}_{n \in \mathbb{N}_0}$$

S Startsymbol

$$S \rightarrow \lambda, S \rightarrow 0S \text{ Regel } (S, \lambda), (S, 0S)$$

$$S \rightarrow 0S \rightarrow 00S \rightarrow 000S \rightarrow 000$$

6.1 Definition (Grammatiken)

Eine Grammatik ist ein Tupel $G = (N, T, P, S)$. Dabei ist

- N das Alphabet der **Nichtterminalsymbole/Variablen**
- T das Alphabet der **Terminalsymbole** mit $N \cup T = \emptyset$
- $P \subseteq ((N \cup T)^* \setminus T^*) \times (N \cup T)^*$ eine endliche Menge von **Regeln/Produktionen**, wobei wir für ein Paar $(u, v) \in P$ auch $u \rightarrow v$ schreiben.
- $S \in N$ das **Startsymbol**

Eine **Satzform** von G ist ein Wort $s \in (N \cup T)^*$ und eine **Terminalwort** von G ist ein Wort $t \in T^*$.

6.2 Definition (Ableitung)

Sei $G = (N, T, P, S)$ eine Grammatik. Eine Satzform w' von G ist in einem Schritt aus einer Satzform w von G **ableitbar**, wenn es Satzformen u, v, x, y von G gibt, so dass $w = xuy$, $u \rightarrow v \in P$ und $w' = xvy$ gelten. Es bezeichne \rightarrow_G die Relation auf der Menge der Satzformen von G , sodass $w \rightarrow_G w'$ genau dann für Satzformen von G gilt, wenn w' aus w in einem Schritt ableitbar ist.

Für Satzformen u, v von G ist eine **Ableitung** von v aus u eine Folge $u = w_1, \dots, w_n = v$ mit $w_i \rightarrow_G w_{i+1} \forall i \in [n-1]$ und eine Ableitung von v in G ist eine **Ableitung** von v aus S in G . Für $n \in \mathbb{N}$ schreiben wir $u \xrightarrow{G}_n v$ wenn es eine Ableitung von v aus u der Länge n gibt und wir schreiben $u \xrightarrow{*}_G v$ wenn eine Ableitung von v aus u in G existiert.

Erklärung: Stell dir vor, du möchtest ein Rezept zum Backen von Kuchen haben. Das Rezept besteht aus verschiedenen Schritten, wie zum Beispiel das Mischen der Zutaten und das Backen im Ofen. In ähnlicher Weise kann man sich eine Grammatik vorstellen, die Regeln für den Aufbau von Sätzen in einer Sprache festlegt. Nehmen wir an, du hast eine Grammatik namens G , die aus Buchstaben (N) und Wörtern (T) besteht. Diese Grammatik hat auch Regeln (P) und einen Startpunkt (S). Eine Satzform ist ein Satz, der in der Grammatik G gebildet werden kann. Jetzt stellen wir uns vor, du hast einen Satz, den wir als " w " bezeichnen. Du möchtest einen anderen Satz, " w' ", in einem Schritt aus dem Satz " w " ableiten. Das bedeutet, dass es bestimmte Regeln gibt, die angewendet werden können, um von " w " zu " w' " zu gelangen. Man kann sich das wie einen Schritt in einem Rezept vorstellen, bei dem man eine Zutat durch eine andere ersetzt oder sie anders kombiniert. Eine Ableitung ist eine Folge von Schritten, bei der man von einem Satz zu einem anderen Satz " v " gelangt. Jeder Schritt in der Ableitung wird durch eine Regel aus der Grammatik G dargestellt. Eine Ableitung von " v " in G ist eine Ableitung von " v " ausgehend vom Startpunkt S in G .

6.3 Definition (Erzeugte Sprache)

Sei $G = (N, T, P, S)$ eine Grammatik. Die von G erzeugte Sprache $L(G)$ ist die Menge aller Wörter $w \in T^*$ für die es eine Ableitung von w in G gibt.

6.4 Lemma

Sei $G = (N, T, P, S)$ eine Grammatik und seien u, v, x, y Satzformen von G und seien $n, m \in \mathbb{N}$ mit $u \rightarrow_G^n v$ und $w \rightarrow_u^n xuy$.

Dann gilt $w \rightarrow_u^{m+n-1} xvy$.

Beweis. Sei $\alpha_1, \dots, \alpha_n$ eine Ableitung von xuy aus w und β_1, \dots, β_m eine Ableitung von v aus u . Dann ist $\alpha_1, \dots, \alpha_{n-1}, x\beta_1y, \dots, x\beta_my = xvy$ eine Ableitung von xvy aus w in G der Länge $n+m-1$.

Im folgenden beschäftigen wir uns mit dem Thema welche Sprache Grammatiken verschiedener Komplexitätsstufen erzeugen können. \square

6.5 Satz

Eine Sprache ist genau dann rekursiv aufzählbar, wenn sie von einer Grammatik erzeugt wird.

Beweisidee Wird eine Sprache L von einer Grammatik erzeugt, so ist L die erkannte Sprache einer TM, die in geeigneter Weise Ableitungen von G erzeugt, prüft ob diese Ableitung dem Wort der Eingabe entspricht und gegebenenfalls akzeptiert. Wenn eine Ableitung der Eingabe gefunden ist.

Gegeben eine rekursiv aufzählbare Sprache L und eine TM, die L erkennt. So konstruieren wir ähnlich dem Postschen Korrespondenzproblems Regeln und Symbole, sodass wir die Arbeitsweise der TM modellieren können und entsprechend mit einem Terminalwort enden wenn dies von der TM erkannt wird.

6.6 Definition (Rechtslinear)

Eine Grammatik $G = (N, T, P, S)$ ist rechtslinear, wenn alle Regeln von der Form

$$X \in uy \text{ oder } X \rightarrow u$$

mit $X, y \in N$ und $u \in T^*$ sind.

Hier ist es sinnvoll endliche Automaten zu betrachten bei denen es nicht \forall Zustände q und Eingabesymbole a ein Tripel (q, a, q') in der Übergangsrelation geben muss. Solche Automaten sind zwangsläufig nicht deterministisch.

6.7 Satz

Eine Sprache ist genau dann regulär, wenn sie von einer rechtslinearen Grammatik erzeugt wird.

Beweisidee Zunächst überzeugt man sich davon, dass eine Sprache L genau dann von einer rechtslinearen Grammatik erzeugt wird, wenn sie von einer Grammatik $G = (N, T, P, S)$ erzeugt wird bei der alle Regeln von der Form $X \rightarrow ay$ oder $X \rightarrow \lambda$ mit $x, y \in N$ und $a \in T$ sind. Eine Solche Grammatik wird als Grammatik in Simulationsform bezeichnet.

Die Sprache L die von einer rechtslinearen Grammatik (in Simulationsform) gebildet wird von dem EA

$$A = (N, T, \Delta, S, \{X \in N : X \rightarrow \lambda \in P\})$$

mit

$$\Delta = \{(X, a, y) \in N \times T \times N : X \rightarrow ay \in P\}$$

erkannt.

Umgekehrt ist es einfach zu sehen, dass jede Reguläre Sprache von einer rechtslinearen Grammatik erzeugt wird.

6.8 Beispiel

Die Sprache $\{0\}^*$ wird von der rechtlinearen Grammatik

$$G = (\{S\}, \{0, 1\}^*, \{S \rightarrow 0S, S \rightarrow \lambda\}, S)$$

erzeugt von EA

$$A = (\{S\}, \{0, 1\}^*, \{(S, 0, S)\}, S, \{S\})$$

erkannt. Sowohl auf der Seite der Maschinenmodelle als auch auf der Seite der Grammatiken gibt es weitere wichtige Sprachklassen, die sich ergeben wenn der Maschinenarbeitsweise oder Menge der zulässigen Regeln weniger stark eingeschränkt wird bei EA und rechtslinear Grammatiken.

6.9 Definition (kontextfrei)

Eine Grammatik $G = (N, T, P, S)$ ist **kontextfrei**, wenn alle Regeln von G von der Form

$$X \rightarrow w$$

mit $X \in N$ und $w \in (N \cup T)^*$ sind. Eine Sprache ist **kontextfrei**, wenn sie von einer Kontextfreien Grammatik erzeugt wird. Die Menge aller kontextfreien Sprachen bezeichnen wir mit **CF**.

6.10 Definition (lexikographische Ordnung)

Sei Σ ein Alphabet. Die **lexikographische Ordnung** auf Σ^* ist die lineare Ordnung leq auf Σ^* für die $u \leq v$ für $u, v \in \Sigma^*$ genau dann gilt wenn eine der folgenden Bedingungen erfüllt ist.

(L1) $u \sqsubseteq v$.

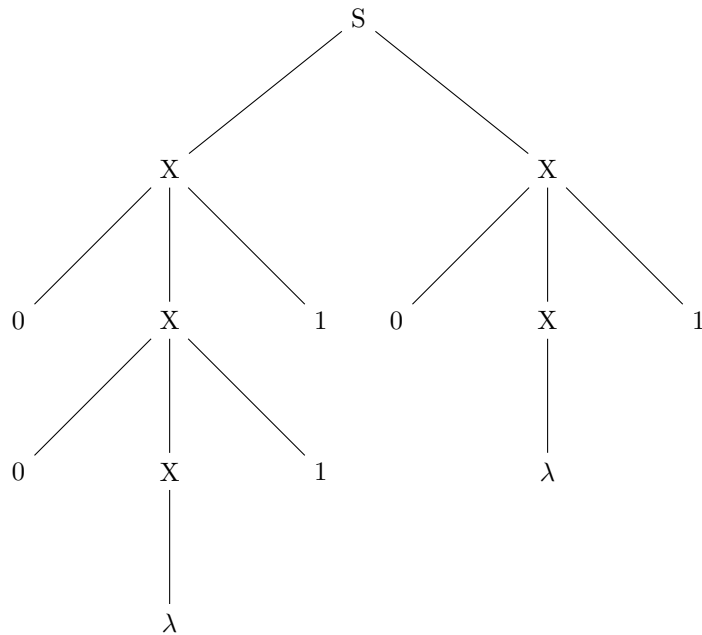
(L2) Es gibt ein $i \in [\min\{|u|, |v|\}]$ mit $u(j) = v(j) \quad \forall j \in [i-1]$ und $u(i) \neq v(i)$ und $u(i) \leq v(i)$ für eine gebildete, lineare Ordnung \leq auf Σ

Schreiben wir \leq für eine lineare Ordnung, so bedeutet $u < v$ für Elemente u, v , dass $u \leq v$ und $u \neq v$ gelten.

Betrachte $G = (\{S, X\}, \{0, 1\}, P, S)$ mit

$$P = \{S \rightarrow XX, X \rightarrow 0X1, X \rightarrow \lambda\}$$

Betrachte Ableitung $S, XX, X0X1, 0x10x1, 0X101, 00X1101, 001101$



- oben Wurzel
- die Ordnung der Knoten/ Ecken ist wichtig
- Knoten/ Ecken haben Beschriftungen

Abbildung 11

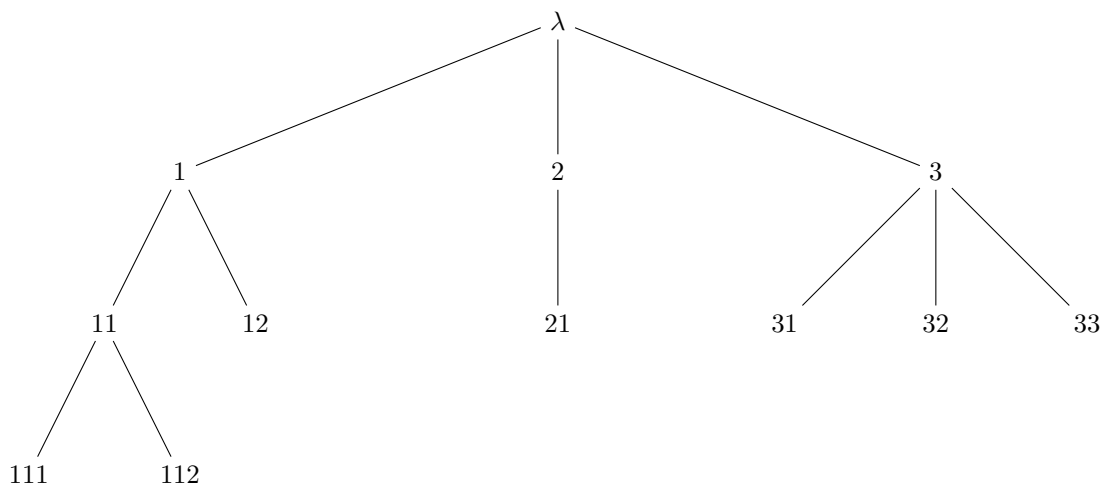


Abbildung 12

- Abbildung 12 repräsentiert die Sprache $T = \{\lambda, 1, 2, 3, 11, 12, 21, 31, 32, 33, 111, 112\}$.

- Der Wurzelknoten ist λ .
- Die Reihenfolge ist lexikographisch.

6.11 Definition (Baum)

Eine endliche nicht leere Sprache T heißt **Baum**, wenn sie unter Präfixbildung abgeschlossen ist, also wenn für alle $w \in T$ und $p \sqsubseteq w$ auch $p \in T$ gilt. Ein Wort $w \in T$ heißt **Blatt** von T , wenn w bezüglich \sqsubseteq maximal in T ist, also wenn $w = w' \quad \forall w' \in T$ mit $w \sqsubseteq w'$ gilt. Wörter $w \in T$, die keine Blätter von T sind heißen **innere Ecken** von T .

6.12 Lemma

Sei Σ eine Alphabet und \leq die lexikographische Ordnung auf Σ^* . Seien $u, w \in \Sigma^*$ mit $u \not\leq w$ und $u \leq w$. Seien $v_1 < \dots < v_n \in \Sigma^*$ mit $v_1 \neq \lambda$. Dann gilt $u < uv_1 < \dots < uv_n < w$

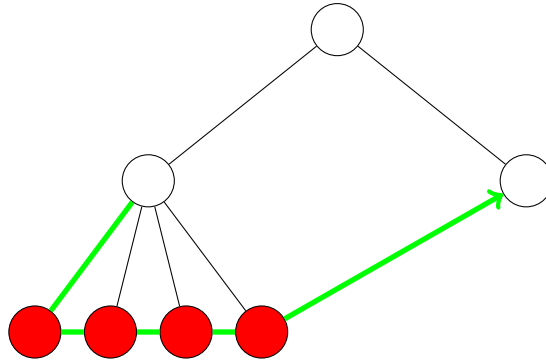


Abbildung 13: Baum und Präfixbildung

Beweis. Für $v, v' \in \Sigma^*$ mit $v \leq v'$ gilt offenbar $uv \leq uv'$, es genügt also zu zeigen, dass $uv \leq w \quad \forall v \in \Sigma^*$ gilt. Wegen $u \not\leq w$ existiert $i \in [\min\{|u|, |w|\}]$ mit $u(j) = w(j) \quad \forall j \in [i-1]$ und $u(i) < w(i)$. Für $v \in \Sigma^*$ gilt dann $(uv)(j) = w(j) \quad \forall j \in [i-1]$ und $(uv)(i) < w(i)$, also $uv < w$. \square

6.13 Lemma

Sei T ein Baum, \leq die lexikographische Ordnung auf T , $p \in T$ und $Q := \{w \in T : p \sqsubseteq w\}$. Es gelten $p = \min Q$ und $Q = \{w \in T : \min Q \leq w \leq \max Q\}$.

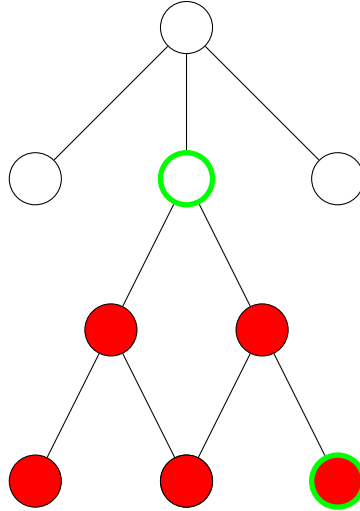


Abbildung 14

Beweis. Offensichtlich gilt $p = \min Q$. Für $w \in Q$ gilt $\min Q \leq w \leq \max Q$ nach Definition von $\min Q$ und $\max Q$. Für $w \notin Q$ mit $w \sqsubseteq w' \quad \forall w' \in Q$, also $w < \min Q$. Für $w \notin Q$ mit $w \not\sqsubseteq p \quad \exists i \in [\min|p|, |w|]$ mit $w(i) \neq p(i)$, insbesondere existiert also ein minimales solches i und nach Definition von \leq folgt damit $w \leq w' \quad \forall w' \in Q$ oder $w' \leq w \quad w' \in Q$, insbesondere also wegen $w \notin Q$ somit $w < \min Q$ oder $\max Q < w$. \square

6.14 Definition (Beschriftung)

Eine **Beschriftung** eines Baumes T mit Elementen einer Menge X ist eine Funktion $b : T \rightarrow X$.

6.15 Definition (Blattwort)

Sei Σ eine Alphabet, sei (T, b) ein Paar aus einem Baum T und einer Beschriftung $b: T \rightarrow \Sigma \cup \{\lambda\}$ und sei t_1, \dots, t_n die Folge der Blätter von T in lexikographischer Reihenfolge. Das **Blattwort** von (T, b) ist $b(t_1) \cdots b(t_n)$. Blattwort: 001101

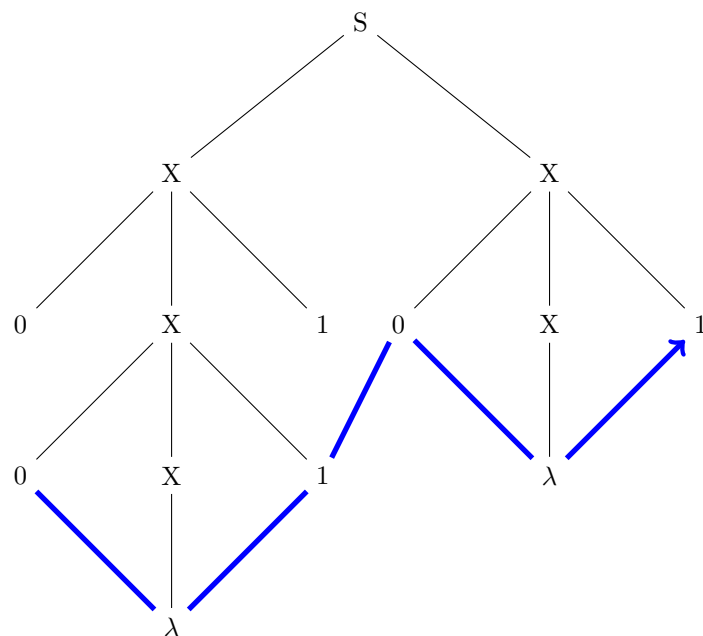


Abbildung 15: Blattwort

6.16 Definition (Ableitungsbaum)

In einer kontextfreien Grammatik $G = (N, T, P, S)$ ist ein **Ableitungsbaum** für eine Satzform $w \in (N \cup T)^*$ aus $A \in N$ ein paar (T, b) , bestehend aus einem Baum T und einer Beschriftung $b : T \rightarrow N \cup T \cup \{\lambda\}$. Der Baum T ist eine Teilmenge von $[d]^*$, wobei $d := \max(\{|w| : x \rightarrow w \in P\} \cup \{1\})$. Die Wurzel des Baumes ist mit dem Symbol A beschriftet und das Blattwort entspricht der Satzform w . Für jede innere Ecke t im Baum T existiert eine Regel $X \rightarrow v \in P$, sodass Folgendes gilt.⁷

- (i) $b(t) = X$
- (ii) $\{t' \in T : t \sqsubseteq t' \text{ und } |t'| = |t| + 1\} = \{ta : a \in \max\{|v|, 1\}\}$
- (iii) $b(ta) = v(a) \quad \forall a \in [|v|]$
- (iv) $b(t_1) = \lambda$ falls $v = \lambda$

Die **Blätter** von (T, b) sind die Blätter von T und die **inneren Ecken** von (T, b) sind die inneren Ecken von T .

6.17 Beispiel

Sei $G = (\{S\}, \{0, 1\}, \{S \rightarrow 0S1, S \rightarrow \lambda\}, S)$, $T = \{\lambda, 1, 2, 3, 21, 22, 23, 221\}$ und sei $b : T \rightarrow \{S, 0, 1, \lambda\}$ die Beschriftung mit $b(\lambda) = b(2) = b(22) = S$, $b(1) = b(21) = 0$ und $b(221) = \lambda$. Dann ist (T, b) ein Ableitungsbaum von 0011 aus S in G . Die **Darstellung** von T sieht wie folgt aus:

HIERABBEINFÜGEN

Abbildung 16: Darstellung von T

Visualisierung 1 + 2

Abbildung 17: Colorization

6.18 Lemma

Sei $G = (N, T, P, S)$ eine kontextfreie Grammatik und w_1, \dots, w_l eine Ableitung von w aus S in G . Dann gibt es einen Ableitungsbaum (T, b) von w aus S in G mit $l - 1$ innere Ecken.

Beweis. Induktion über l .

L = 1 $S = (T, b)$

Sei nun $l \geq 2$ und die Aussage wahr $\forall l' < l$. Sei (T', b') ein Ableitungsbaum von w_{l-1} aus S in G mit $l - 2$ innere Ecken (existiert per Induktions Hypothese). Sei $w' := w_{l-1}$. Sei $i \in [|w'|]$ und $v \in (N \cup T)^*$ mit $w'(i) \rightarrow v \in P$ und $w = w'(1) \cdot w'(i - 1)vw'(i + 1) \cdots w'(|w'|)$. Sei t' das in lexikographischer Reihenfolge i -te Blatt von T' und sei (T, b) der Ableitungsbaum mit $T = T' \cup \{t'a : a \in [|v|]\}$, $b(\tilde{w}) = b'(\tilde{w}) \quad \forall \tilde{w} \in T'$ und $b(t'a) = v(a) \quad \forall a \in [|v|]$. Aus Lemma 6.12 folgt dann, dass w das Blattwort von (T, b) ist. Außerdem ist die Anzahl der inneren Ecken von T durch $l - 2 + 1 = l - 1$ gegeben. \square

⁷Tex. Ed. sug.

6.19 Lemma

Sei $G = (N, T, P, S)$ eine kontextfreie Grammatik und (T, b) ein Ableitungsbaum von w aus S in G mit $l-1$ inneren Ecken. Dann gibt es eine Ableitung $leq = w_1, \dots$

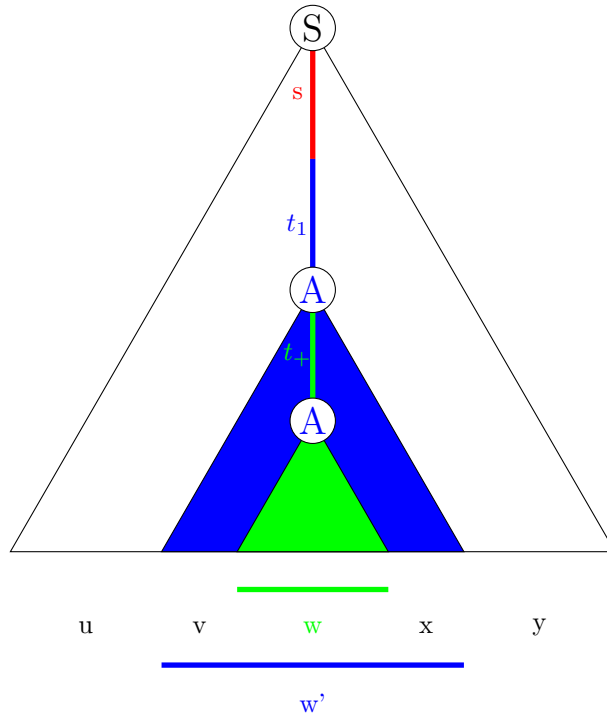
ohnebeweis

6.20 Satz (Pumping-Lemma für kontextfreie Sprachen)

Sei Σ ein Alphabet. Für jede kontextfreie Sprache $L \subseteq \Sigma^*$ gibt es eine Konstante $\in \mathbb{N}$, so dass folgendes gilt: Ist $z \in L$ mit $|z| \geq k$, so gibt es Wörter $u, v, w, x, y \in \Sigma^*$ mit $z = uvwxy$, so dass folgendes gilt.

- (i) $vx \neq \lambda$
- (ii) $|vwx| \leq k$
- (iii) $uv'wx'y \in L \quad \forall i \in \mathbb{N}_0$

Beweis. Sei $G = (N, T, P, S)$ eine kontextfreie Grammatik mit $L(G) = L$, sei $d := \max (\{|v| : X \rightarrow v \in P\} \cup \{1\})$, sei $k := |[d]|^{\geq |N|+2}$. Sei $z \in L$ mit $|z| \leq k$. Sei (T, b) ein Ableitungsbaum von z aus S in G mit minimaler Anzahl innerer Ecken. Sei $r \in T$ ein Wort maximaler Länge. Da z Blattwort von T ist folgt insbesondere $|T| \leq |z| \leq k < |[d]|^{|N|+1}$, also $T \not\subseteq [d]^{\geq |N|+1}$ und somit gilt $|r| \geq |N| + 2$. Seien $s, t \in [d]^*$ mit $r = st$ und $|t| = |N| + 2$. Nach Schubfachprinzip existieren Präfixe t_1, t_2 von t mit $b(st_1) = b(st_2) \in N$ und $|t_1| < |t_2|$. Dann ist auch t_1 Präfix



Abbildungung 18: Ableitungsäume

Wir betrachten vier Ableitungsäume.

- (1) Sei (T_1, b_1) der Ableitungsbaum mit $T_1 = \{q \in [d]^* : s, t, q \in T\}$. Sei w' das Blattwort von $(t_1, b_1) \cdot (T_1, b_1)$ ist ein Ableitungsbaum von w' aus A in G .
- (2) Sei T_2, b_2 der Ableitungsbaum mit $T_2 = \{q \in [d]^* : st_2q \in T\}$ und $b_2(q) = b(st_2q) \quad \forall q \in T_2$. Sei w das Blattwort von T_2, b_2 . Der Ableitungsbaum T_2, b_2 ist ein Ableitungsbaum von w aus A in G .
- (3) Sei (U_0, c_0) der Ableitungsbaum mit $U_0 = T / (\{st_1\}(T \setminus \{\lambda\}))$ und $c_0(q) = b(q) \quad \forall q \in U_0$ und sei z_- das Blattwort von U_0, c_0 . Aus Lemma 6.13 folgt, dass es $u, y \in (N \cup T)^*$ gibt, sodass $z = uw'y$ und $z_- = uAy$.
- (4) Sei (U_1, c_1) der Ableitungsbaum mit $U_1 = T_1 / (\{t_1\}(T_2 \setminus \{\lambda\}))$ und $c_1(q) = b_1(q) \quad \forall q \in U_1$ und sei w'_- aus A in G . Aus Lemma 6.13 folgt, dass es $v, x \in (N \cup T)^*$ gibt, so dass $w' = vwxundw'_- = vAx$ gelte.

Insgesamt existiert somit Wörter u, v, w, x, y mit $z = uvwx y = uw' y$ und

$$\begin{array}{ll} & \text{bezeugt durch} \\ S \rightarrow_G^* uAy & (U_0, c_0) \\ \circledast \quad A \rightarrow_G^* vAx & (U_1, c_1) \\ A \rightarrow_G^* w & (T_2, b_2) \end{array}$$

Da T minimal ist gilt $vx \neq \lambda$. Da $|t| = |N| + 2$, ist die Anzahl der Ecken in T_2, b_2 höchstens $k \Rightarrow |vwx| \leq k$. Aus \circledast folgt $uv^i wx^i y \in L \quad \forall i \in \mathbb{N}_0$

6.21 Beispiel

$L = \{0^n 1^n 0^n : n \in \mathbb{N}_0\} \notin \mathbf{CF}$ lässt sich mit dem Pumping-Lemma für kontextfreie Sprachen wie folgt zeigen:

Beweis. Angenommen $L \in \mathbf{CF}$. Sei $k \in \mathbb{N}$ für L wie im Satz 6.20 gewählt. Sei $z := 0^k 1^k 0^k$. Seien $u, v, w, x, y \in \{0, 1\}^*$ gemäß der Wahl von k Wörter mit $uvwx y = z$, $vx \neq \lambda$, $|vwx| \leq k$ und $uv^i wx^i y \in L \quad \forall i \in \mathbb{N}_0$. Aus $uvwx y = z$ und $|vwx| \leq k$ folgt, dass $r, s \leq k$ mit $u = 0^r \wedge y = 1^s 0^k$ oder $u = 0^k 1^r \wedge y = 0^s$ existieren. Wegen $vx \neq \lambda$ existieren also $r, s \leq k$ mit $r \leq k-1$ oder $s \leq k-1$ so dass $uw y = 0^r 1^s 0^k$ oder $uw y = 0^k 1^r 0^s$ gilt. In jedem Fall gilt also $uw y \notin L$ im Widerspruch dazu, dass $uv^i wx^i y \in L \quad \forall i \in \mathbb{N}_0$ gilt. \square

6.22 Beschreibung (Kellerautomat)

Die Konstruktion von **Kellerautomaten** folgt den folgenden Ideen:

- Es handelt sich im wesentlichen um endliche Automaten, die nicht wie TM zusätzlich ein Band zur freien Verfügung haben, sondern stattdessen einen Stack/ Keller.
- Zustandsübergänge hängen nicht nur vom Zustand und eingelesenen Symbol ab, sondern auch vom obersten Kellersymbol.
- Bei jedem Zustandsübergang wird das oberste Kellersymbol entfernt und es werden nacheinander Symbole eines Wortes über dem Kelleralphabet oben auf den Keller gelegt.

Bei geeigneter Formalisierung von Kellerautomaten lässt sich folgendes zeigen.

6.23 Satz

Eine Sprache wird genau dann von Kellerautomaten erkannt, wenn sie kontextfrei ist.

6.24 Beispiel

Die Sprache $\{0^n 1^n : n \in \mathbb{N}_0\}$ wird von der kontextfreien Grammatik

$$G = (\{S\}, \{0, 1\}, \{S \rightarrow 0S1, S \rightarrow \lambda\}, S)$$

erzeugt. Ein Kellerautomat könnte prüfen ob die Eingabe von der Form $0^m 1^n$ ist und den Keller benutzen um zu erkennen ob $\#_0(w) = \#_1(w)$ für eine Eingabe w ist.

6.25 Definition (Kontextsensitiv)

Eine Grammatik $G = (N, T, P, S)$ heißt **kontextsensitiv**, wenn alle Regeln von G von der Form

$$uXv \rightarrow uvv$$

mit $x \in N, u, v \in (N \cup T)^+$. Eine Sprache L heißt **kontextsensitiv**, wenn L/λ von einer kontextfreien Grammatik erzeugt wird. Die Menge aller kontextsensitiven Sprachen bezeichnen wir mit **CS**.

6.26 Definition (nicht verkürzend)

Eine Grammatik $G = (N, T, P, S)$ ist **nichtverkürzend**, wenn alle Regeln von G von der Form $u \rightarrow v$ mit $u, v \in (N \cup T)^*$ mit $|u| \leq |v|$ sind.

6.27 Satz

Eine Sprache L ist genau dann kontextsensitiv, wenn $L/\{\lambda\}$ von einer nichtverkürzten grammatik erzeugt wird.

Beweis. Beweisidee: Kontextsensitive grammatiken sind immer nichtverkürzend. Es genügt also nichtverkürzende Grammatiken in kontextsensitive Grammatiken umzuwandeln. Jede nichtverkürzende Grammatik kan in eine nichtverkürzende Grammatik umgewandeln, die dieselbe Sprache erzeugt, sodass alle Regeln von der Form

$$X_1 \cdots X_m \rightarrow Y_1 \cdots Y_n \text{ oder } X \rightarrow a$$

mit $m \leq n$, $X, X, \dots, X_m, Y_1, \dots, Y_n \in N$ und $a \in T$ sind. (Das man Nichtterminalsymbol A einführt für jedes $a \in T$ und in allen Regeln a durch A ersetzt und die Regeln $A \rightarrow a \quad \forall a \in T$ hinzufügt.) Eine solche grammatik heißt **separiert** und kann in eine kontextsensitive Grammatik umgewandelt werden. Dazu betrachten wir jede regel $p = X_1 \cdots X_m \rightarrow Y_1 \cdots Y_n$ mit $X_1, \dots, X_m, Y_1, \dots, Y_n \in N$. Wir ersetzen p durch die Regeln

$$\begin{aligned} X_1 &\rightarrow Z_1^p \\ Z_1^p X_{i+1} &\rightarrow Z_i^p Z_{i+1}^p \text{ mit } i \in [m-2] \\ Z_{m-1}^p X_{i+1} &\rightarrow Z_i^p \tilde{Z}_m^p Y_{m+1} \cdots Y_n \\ Z_i^p \tilde{Z}_{i+1}^p &\rightarrow \tilde{Z}_i^p \tilde{Z}_{i+1}^p + 1^p \text{ mit } i \in [m-1] \\ \tilde{Z}_i^p \tilde{Z}_{i+1}^p &\rightarrow \tilde{Z}_i^p Y_{i+1} \text{ mit } i \in [m-i] \\ \tilde{Z}_1^p &\rightarrow Y_1 \end{aligned}$$

Dabei sind $Z_1^p, \dots, Z_{m-1}^p, \tilde{Z}_1^p, \dots, \tilde{Z}_m^p$ neue Nichtterminalsymbole. □

6.28 Beschreibung (linear beschränkter Automat)

Ein linear beschränkter Automat, kurz LBA, ist ein l-TM M , die \forall Wörter w über dem Eingabealphabet in allen Rechnungen von M zur Eingabe w nur die durch den Eingabemechanismus mit den Symbolen von w beschränkten Feldern und die an diese angrenzenden Felder besucht.

6.29 Satz

Eine Sprache ist genau dann kontextsensitiv, wenn sie von einem LBA erkannt wird.

ohnebeweis

6.30 Beispiel

Die Sprache $\{0^n 1^n 0^n : n \in \mathbb{N}\}$ wird von einer nicht verkürzenden Grammatik $G = (N, \{0, 1\}, P, S)$ erzeugt mit $N = \{S, A, B, C, \tilde{A}, \tilde{B}, \tilde{C}\}$ und

$$P = S \rightarrow \tilde{A}B\tilde{C}, S \rightarrow 010, \tilde{C} \rightarrow CAB\tilde{C}, BA \rightarrow AB, CA \rightarrow AC, CB \rightarrow BC, \tilde{A}A \rightarrow 0\tilde{A}, \tilde{A}B \rightarrow 0\tilde{B}, \tilde{B}B$$

Diese Sprachklassen bilden die **Chomsky-Hirarchie**

6.31 Definition

Die Klassen der Chomsky-Hirarchie sind $\mathbf{CH}(0) := \mathbf{RE}$, $\mathbf{CH}(1) := \mathbf{CS}$, $\mathbf{CH}(2) := \mathbf{CF}$, $\mathbf{CH}(3) := \mathbf{REG}$

6.32 Satz

Es gilt

$$\mathbf{REG} \not\subseteq \mathbf{CF} \not\subseteq \mathbf{CS} \not\subseteq \mathbf{REC} \not\subseteq \mathbf{RE}$$

Beweis. Beweisidee: Für die Echtheit der Inklusionen siehe Beispiel 6.33. Für die Gültigkeit der Inklusionen bleiben $\mathbf{CF} \leq \mathbf{CS}$ und $\mathbf{CD} \leq \mathbf{REC}$ zu zeigen. Ist $G = (N, T, P, S)$ eine kontextfreie Grammatik, so erhält man die Menge P' der Regeln einer kontextsensitiven Grammatik $G' = (N, T, P', S)$ mit $L(G) = L(G')/\{\lambda\}$ wenn ausgehend von P in zwei Schnitten wie folgt verfahren wird. Zunächst werden alle Regeln $X \rightarrow u \in P$ die Regeln $X \rightarrow u'$ hinzugefügt, bei denen u' aus u entsteht indem ein oder mehrere Symbole aus u entfernt werden für die es eine Ableitung von $\lambda \text{ aus } Y$ gibt. Dann werden alle Regeln der Form $X \rightarrow \lambda$ entfernt. $\Rightarrow \mathbf{CF} \subseteq \mathbf{CS}$

Jede kontextfreie Sprache ist entscheidbar, denn zu einem geeignet gesetzten LBA M kann wegen der endlichen Anzahl von Konfigurationen von M effektiv von einer TM entschieden ob dieser eine gegebene Eingabe akzeptiert oder nicht. \square

6.33 Beispiel

- (i) Die Sprache $L_1 = \{1^n : n \in \mathbb{N}\}$ ist offensichtlich regulär.
- (ii) Die Sprache $L_2 = \{0^n 1^n : n \in \mathbb{N}\}$ ist kontextfrei (Beispiel 6.24), aber nicht regulär \rightarrow Pumping Lemma.
- (iii) Die Sprache $L_3 = \{0^n 1^n 0^n : n \in \mathbb{N}\}$ ist kontextsensitiv (Beispiel 6.30), aber nicht kontextfrei (Beispiel 6.21, Pumping Lemma für kontextfreie Sprachen)
- (iv) Sei M_0, M_1, \dots eine geeignete Aufzählung aller LBAs mit Zustandsmenge $\{0, \dots, n\}$ für ein $n \in \mathbb{N}$, Eingabealphabet $\{1\}^*$ und Bandalphabet $\{0, \dots, m, \square\}$ für ein $m \in \mathbb{N}$.
Die Sprache $L_4 = \{1^e \notin L(M_e)\}$ ist entscheidbar aber nicht kontextsensitiv.
- (v) Die Sprache $\{1^e : e \in H_{diag}\}$ ist rekursiv aufzählbar, aber nicht entscheidbar.
- (vi) Die Sprache $\{1^e : e \notin H_{diag}\}$ ist nicht rekursiv aufzählbar.

Zeitkomplexität

- *ChatGPT*

Komplexität Einschränkung von TM bezüglich Zeit (Anzahl Rechenschritte einer TM) und Platz (Anzahl besuchte Felder einer TM)

- Worst case
- Average case
- Best case

7.1 Definition (Rechenzeit)

Sei $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ eine DTM. Es bezeichne $time_M : \Sigma^* \rightsquigarrow \mathbb{N}_0$ die partielle Funktion mit $dom(time_M) = dom(\varphi_M)$, so dass $time_M(w)$ für alle $w \in dom(\varphi_M)$ die Länge der Rechnung von M zur Eingabe w ist. Die **rechenzeit** von M zur Eingabe $w \in dom(\varphi_M)$ ist $time_M(w)$.

Da im Folgenden nahezu alle Aussagen nur im Hinblick auf die Betrachtung Eingabelänge $\rightarrow \infty$ sinnvoll sind, lassen wir in der nächsten Definition **endlich** viele Ausnahmen zu.

7.2 Definition (zeitbeschränkt)

Eine **Zeitschranke** ist eine berechenbare Funktion $t : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ mit $t(n) \leq n \quad \forall n \in \mathbb{N}_0$.

Sei $t : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ eine Funktion. Eine TM M ist **t-zeitbeschränkt**, wenn M total ist und es ein $n_0 \in \mathbb{N}_0$ gibt, so dass $\forall w \in \Sigma^{\leq n_0}$ alle Rechnungen von M zur Eingabe w höchstens Länge $t(|w|)$ haben.

Ist n ein Variablensymbol und t ein Term, der eine Funktion $f : n \mapsto t$ festlegt, so verwendet man auch die Zeichnung t-zeitbeschränkt an Stelle von f-zeitbeschränkt. Man spricht also zum Beispiel von n^2 -zeitbeschränkt der $3x^3$ -zeitbeschränkt oder auch von $f(n)$ -zeitbeschränkt für eine Funktion $f : \mathbb{N}_0 \rightarrow \mathbb{N}_0$. Betrachten wir eine Funktion $t : \mathbb{N}_0 \rightarrow \mathbb{R}_{\geq 0}$ als Zeitschranke, so ist damit die Funktion $t' : \mathbb{N}_0 \rightarrow \mathbb{N}_0, n \mapsto \lfloor t(n) \rfloor$ gemeint.

7.3 Satz (lineare Beschleunigung)

Sei $\tilde{c} > 1$, $\epsilon > 0$ und $t : \mathbb{N}_0 \rightarrow \mathbb{R}_{\geq 0}$ mit $t(n) \leq (1 + \epsilon)n \quad \forall n \in \mathbb{N}_0$. Ist M eine $\tilde{c} \cdot t(n)$ -zeitbeschränkt k-DTM, $k \leq 2$, so gibt es auch eine $t(n)$ -zeitbeschränkte k-TM M' mit $L(M) = L(M')$.

Beweisskizze: Die Idee ist es sei $c \leq 2$ wir wollen c Felder in einen Vektor der Länge c zusammenfassen und damit das Bandalphabet vergrößern und andererseits ungefähr "c Schritte auf einmal machen". Mit einem neuen Bandalphabet

$$((\Gamma \cup \{\underline{a} : a \in \Gamma\})^c / \{\square\}^c) \cup \{\square\} \cup \Sigma$$

dabei übernimmt \square die Rolle von $\{\square\}^c$ und die Markierung ist für den Kopf der ursprünglichen TM. Konstruktion einer neuen TM M', die schneller ist als eine gegebene TM M.

- 1.Schritt: Transformation der Eingabe in die "komprimierte" Form auf dem 2.Band ($|w|+1$ Schritte)
- 2.Schritt: Simulation von M bandweise (1. und 2. Band vertauscht). Wir benutzen Vektoren der Länge c um c Felder von M darzustellen. Zeichen mit einer Markierung $_$, markieren den Köpfen den simulierten TM M.
- 3.Schritt: Die TM M' speichert in ihren Zuständen die Beiden Felder rechts und links des Feldes wo die simulierten Köpfe von M sind.
 - Simulation von c Schritten von M.
 - Update der Felder, die von M verändert wurden 6 Schritte notwendig

Insgesamt macht M' also $|w| + 1 + 6 \cdot \frac{\tilde{c}t(|w|)}{c} + 10 \approx \frac{6\tilde{c}}{c}t(|w|) = t(|w|)$ für $c := 6\tilde{c}$. In gewissem Sinne ist die obige Konstruktion invers zum [Alphabetwechsel des bandalphabets bei der Konstruktion normierter TM](#). Entsprechend steigt die Laufzeit auch nur um einen konstanten Faktor wenn wir die TM normieren

7.4 Satz (Alphabetwechsel)

Ist M eine $t(n)$ -zeitbeschränkte k -TM M mit Eingabealphabet $\{0, 1\}$, so gilt es eine Konstante $c \geq 1$ und eine $ct(n)$ -zeitbeschränkte k -TM M' mit Eingabealphabet $\{0, 1\}$, Bandalphabet $\{\square, 0, 1\}$ und $L(M') = L(M)$.
 \rightsquigarrow Konstanten spielen nur eine untergeordnete Rolle für uns.

7.5 Definition(Landau-Symbole)*

Für eine Funktion $f : \mathbb{N}_0 \rightarrow \mathbb{R}_{\geq 0}$ sei

$$o(f) = \{g : \mathbb{N}_0 \rightarrow \mathbb{R}_{\geq 0} : \forall \epsilon > 0 : \exists n_0 \in \mathbb{N}_0 : \forall n \geq n_0 : g(n) \leq \epsilon f(n)\}$$

$$O(f) = \{g : \mathbb{N}_0 \rightarrow \mathbb{R}_{\geq 0} : \exists c > 0 : \exists n_0 \in \mathbb{N}_0 : \forall n \geq n_0 : g(n) \leq cf(n)\}$$

$$\theta(f) = \{g : \mathbb{N}_0 \rightarrow \mathbb{R}_{\geq 0} : \exists c, \epsilon > 0 : \exists n_0 \in \mathbb{N}_0 : \forall n > n_0 : \epsilon f(n) \leq g(n) \leq cf(n)\}$$

$$\Omega(f) = \{g : \mathbb{N}_0 \rightarrow \mathbb{R}_{\geq 0} : \exists \epsilon > 0 : \exists n_0 \in \mathbb{N}_0 : \forall n \geq n_0 : g(n) \geq \epsilon f(n)\}$$

$$w(f) = \{g : \mathbb{N}_0 \rightarrow \mathbb{R}_{\geq 0} : \forall c > 0 : \exists n_0 \in \mathbb{N}_0 : \forall n \geq n_0 : g(n) \leq cf(n)\}$$

7.6 Bemerkung(Eigenschaften Landau-Symbole)*

Seien $f, g : \mathbb{N}_0 \rightarrow \mathbb{R}_{\geq 0}$ Funktion dann gilt:

- (i) $g \in o(f) \Leftrightarrow f \in w(g)$
- (ii) $g \in O(f) \Leftrightarrow f \in \Omega(g)$
- (iii) $g \in \theta(f) \Leftrightarrow (g \in O(f) \text{ und } g \in \Omega(f))$

7.7 Bemerkung(Eigenschaften Landau-Symbole)*

Seien $f, g : \mathbb{N}_0 \rightarrow \mathbb{R}_+$ Funktion dann gilt:

- (i) $g \in o(f) \Leftrightarrow \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0$
- (ii) $g \in O(f) \Leftrightarrow \limsup_{n \rightarrow \infty} \frac{g(n)}{f(n)} < \infty$
- (iii) $g \in \Omega(f) \Leftrightarrow \liminf_{n \rightarrow \infty} \frac{g(n)}{f(n)} > 0$
- (iv) $g \in w(f) \Leftrightarrow \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = \infty$

Bei diesen Klassen ist es üblich " $=$ " statt " \in " zu verwenden, also beispielsweise $g = O(f)$ statt $g \in O(f)$ zu schreiben. Unsere Konstruktion für lineare Beschleunigung bedurfte mehrere Bänder. Tatsächlich kann es einen Unterschied machen ob ein oder mehrere Bänder zur Verfügung stehen. Als Beispiel betrachten wir Binärpalindrome:

Für $w \in \{0, 1\}^*$. Sei $w^R = w(|w| \cdots w(1))$. Die Sprache der Binärpalindrome ist folglich $\{w \in \{0, 1\}^* : w = w^R\}$.

7.8 Proposition(Binärpalindrome erkennen)*

Es gibt eine $t(n)$ -zeitbeschränkte DTM M mit $L(M) = \{w \in \{0, 1\}^* : w = w^R\}$ und $t(n) = O(n)$.

Beweis. Eine 2-DTM M die die Eingabe von links nach rechts durchläuft und dabei die Eingabe auf das 2. Band kopiert, dann auf dem ersten Band den Kopf zurück auf die Ausgangsposition bringt, dann gleichzeitig die Eingabe von links nach rechts und die Kopie symbolweise vergleicht und genau dann akzeptiert, wenn alle Vergleiche positiv waren, dann erkennt M die Binärpalindrome. \square

7.9 Satz (Zeitbeschränkung einer Turingmaschine)*

Ist M eine $t(n)$ -zeitbeschränkte 1-DTM mit $L(M) = \{w \in \{0,1\}^* : w = w^R\}$, so ist $t(n) = \Omega(n^2)$

Beweis. Sei M eine $t(n)$ -zeitbeschränkter 1-DTM mit Eingabealphabet $\{0, 1\}$, $L := \{ww^R : w \in \{0,1\}^*\} \subseteq L(M)$ und $t(n) \neq \Omega(n^2)$. Wir zeigen $L \neq L(M)$. Aus der Existenz von M folgt die Existenz einer $(2t(n) + 1)$ -zeitbeschränkte TM M' mit Zustandsmenge Q wobei $|Q| \leq 2$, Eingabealphabet $\{0, 1\}$ und $L(M) = L(M')$, so dass der Kopf von M' am Ende jeder Rechnung auf dem Feld steht auf dem er am Anfang stand. Wir betrachten die Felder auf dem Band von M' als beginnend mit dem Feld auf dem der Kopf zu Rechnungsbeginn steht mit den natürlichen Zahlen 1, 2, 3, ...

Für jede Rechnung R von M' sei $C_i(R)$ die Folge q_1, \dots, q_l von Zuständen von M' , so dass M' während der Rechnung R insgesamt l und die Grenze zwischen i und $i+1$ überschritt und so dass q_j für $j \in [l]$ der Zustand ist, in dem sich M' unmittelbar vor dem Aufführen der Anweisungen befindet, die das j -te übertreten auslöst. Die Folgen $C_i(R)$ werden Crossing-Sequenz von M' genannt.

Die wesentliche Eigenschaft dieser Crossing-Sequenzen, die wir benutzen, ist wie folgt:

Ist R eine akzeptierte Rechnung von M' zur Eingabe u und S eine Rechnung zur Eingabe v und sind $i \in [|u|]$ und $j \in [|v|]$ mit $C_i(R) = C_j(S)$, so gilt

$$u(1) \cdots u(i)v(j+1) \cdots v(|v|) \in L(M')$$

Wir nutzen die kurze Laufzeit von M' nun mittels Schubfachprinzip gleiche Crossing-Sequenzen zu Rechnungen von unterschiedlichen Eingaben zu finden, um zu folgern, dass M' ein Wort akzeptiert, das nicht in L liegt.

Sei $\epsilon := \frac{1}{300 \log_2 |Q|}$. Aus $t(n) \neq \Omega(n^2)$ folgt die Existenz eines n mit $\epsilon n \leq 1$. Sei $n' := \frac{n}{4}$.

Für jedes Wort $L' := \{w'0^{\frac{n}{2}}w'^R : w' \in \{0,1\}^{\frac{n}{4}}\} \subseteq L$ betrachten wir die kürzeste Crossing-Sequenz $C_w \in \{C_{n'}(R), \dots, C_{n-n'}(R)\}$ einer akzeptierten Rechnung R von M' zur Eingabe w . Für jede Rechnung von M' zu einer Eingabe $w \in \{0,1\}^n$ gilt

$$\sum_{i=n'}^{n-n'} |C_i(R)| \leq 2t(n) + 1$$

wobei $|C|$ die Länge von C bezeichnet. Für jedes Wort $w \in \{0,1\}^n$ folgt damit

$$|C_w| \leq \frac{2t(n) + 1}{n - 2n' + 1} \leq \frac{4cn^2}{\frac{n}{2} + 1} \leq 16\epsilon n$$

Die Anzahl der Crossing-Sequenzen von M' der Länge $16\epsilon n - 1$ ist höchstens

$$(|Q| + 1)^{16\epsilon n} \leq \frac{1}{2}(2|Q|)^{16\epsilon n} \leq \frac{1}{2}|Q|^{32\epsilon n} \leq 2^{n'-1}$$

mit $\Rightarrow |Q| = 2^{\log_2 |Q|}$

Die Anzahl der Wörter in L' ist aber $2^{n'}$. Es gibt also voneinander verschiedene Wörter in L' und $i \in [|u|]$ und $j \in [|v|]$ mit $C_i(R) = C_j(S)$ wobei R die Rechnung von u ist und S die Rechnung von v in M' . Es gibt also einen Präfix u' von u mit $|u'| \leq n'$ und einen Suffix v' von v mit $|v'| \leq n'$, sodass $u'v' \in L(M') = L(M)$ gilt. Da u und v verschieden sind gilt $u'v' \notin L$. Im Sinne von Satz 7.9 macht es zwar einen Unterschied ob TM mit einem oder mit mehreren Bändern betrachtet werden, der Anstieg der Laufzeit ist bei Simulation mehrerer Bänder auf einem Band mittels Spurentechnik aber nur quadratisch. \square

7.10 Satz(Reduktion auf 1-Band-TM)*

Ist M eine $t(n)$ -zeitbeschränkte TM, so gibt es eine $t'(n)$ -zeitbeschränkte 1-TM mit $t'(n) = O((t(n))^2)$, die dieselbe Sprache erkennt. Während der Laufzeitanstieg bei der Reduktion auf ein Band quadratisch sein kann, ist die Reduktion von k auf zwei Bänder effizienter möglich

7.11 Satz(Codekonstruktion und Simulation)*

Ist für alle deterministischen TM $M = (\{0, \dots, n\}, \{0, 1\}, \{0, 1, \square\}, \Delta, 0, \{0\})$ und für alle Wörter $w \in \{0, 1\}^*$ das Wort $\text{code}(M, w)$ ein geeigneter Code für (M, w) , so gibt es eine 2-DTM U , so dass folgendes gilt:

- (i) \forall DTM M wie oben und $\forall w \in \{0, 1\}^*$ akzeptiert U den Code $\text{code}(M, w)$ genau dann, wenn M das Binärwort w akzeptiert.
- (ii) Für alle Zeitschranken t , alle t -zeitbeschränkten DTM M wie oben, gibt es ein n_0 , sodass $\forall w \in \{0, 1\}^{\leq n_0}$ eine Konstante $c \in \mathbb{N}$ gibt, sodass die Rechnung von U zur Eingabe $\text{code}(M, w)$ Länge höchstens $ct(|w|)\log(t(|w|))$ hat.

Beweis. Idee:

1. Verwende [Spurentechnik](#) um die womöglich vielen Bänder von M auf einem Band zu simulieren.
2. Intelligentes Speichermanagement: Simulierte Köpfe bleiben in der Mitte simuliertes Band "bewegt" + geschicktes Lücken lassen im Speicher.

Bild für den Fall, dass M eine Band hat

ABBHIEREINFÜGEN

Genauer verfügt U bei Eingabe $\text{code}(M, w)$ wie folgt:

- Zu Beginn werden einige Initialisierungsschritte ausgeführt, die es erlauben die k Bänder von M in jeweils einer Spur auf den ersten Band von U geeignet zu simulieren, wobei auf dem zweiten Band eine Repräsentation von M erstellt wird. Es wird aber sichergestellt, beispielsweise mittels mehrerer Spuren auf dem zweiten Band, dass das zweite Band noch verwendet werden kann um mit un der Abschnittslänge linearem Aufwand Abschnitte auf dem ersten NBand verschieben zu können.
- Während der Simulation von M wird jede Spur auf dem ersten Band von U in Abschnitte unterteilt
 - Die Felder auf dem ersten Band von U auf dem der Kopf zu Beginn steht bildet für alle Spuren $i \in [k]$ den aus einem Feld bestehenden Abschnitt $H^{(i)}$. Die Simulation von M wird so mittels angepasster Spurentechnik durchgeführt, dass die Simulierte Kopfposition auf jeder Spur $z \in [k]$ das Feld im Abschnitt $H^{(i)}$ ist und auf keiner Spur wird dieses Feld am Ende der Simulation eines Schrittes von M mit $\#$ beschrieben sein.
 - Auf jeder Spur $i \in [k]$ ist unmittelbar links von $H^{(i)}$ der Abschnitt $L_1^{(i)}$ der Länge 2 und für jedes $j \leq 1$ unmittelbar links von $L_j^{(i)}$ der Abschnitt $L_{j+1}^{(i)}$ der Länge 2^{j+1} .
 - Analog ist auf jeder Spur $i \in [k]$ unmittelbar rechts von $H^{(i)}$ der Abschnitt $R_1^{(i)}$ der Abschnitt $R_{j+1}^{(i)}$ der Länge 2^{j+1} .
 - Jeder Abschnitt auf einer Spur $i \in [k]$, außer $H^{(i)}$, wird beim ersten Verwenden in dem Sinne als halb leer initialisiert, dass er zur Hälfte mit den die Blank-Symbole von M repräsentierenden Symbolen beschrieben.
 - Zu jedem Zeitpunkt stehen die Lückensymbole $\#$ in einem Abschnitt am weitesten links im Abschnitt.
 - Ein Abschnitt $L_j^{(i)}$ oder $R_j^{(i)}$ heißt **leer**, wenn alle Symbole in diesem Abschnitt $\#$ sind, **halb leer** wenn die Hälfte der Symbole in diesem Abschnitt $\#$ oder er nicht initial ist und **voll** wenn kein Symbol in diesem Abschnitt $\#$ ist.
 - Die Darstellung der durch die Spuren repräsentierten Bänder von M ist dabei so zu verstehen, dass die sich wie bei der Spurentechnik üblichen Darstellung der Bänder ergeben wenn die Felder mit den Lückensymbolen $\#$ ignoriert werden.
 - Vor Simulationsbeginn wird w als Eingabe an die simulierte Maschine M übergeben ohne dabei die Lückensymbole $\#$ zu überschreiben.

- Werden nun die Bandbewegungen geeignet durch die Spurenmanipulationen realisiert, so kann U die Turingmaschine M simulieren, da den Abschnitt $H^{(i)}$ die zur Simulation nötigen Informationen über die gegenwärtige Konfiguration von M entnommen werden können.
- Bandbewegungen nach links (die Kopfbewegung nach rechts sinnvoll) werden durch manipulationen der entsprechenden Spur wie folgt simuliert:
 - Für das minimale $j_0 \leq 1$, so dass $R_{j_0}^{(i)}$ nicht leer ist, ersetzt das erste Symbol der rechten Hälfte, falls $R_{j_0}^{(i)}$ halb voll ist, und der linken Hälfte, falls $R_{j_0}^{(i)}$ voll ist, das Symbol im Abschnitt $H^{(i)}$ und die $2^{j_0-1} - 1$ weitere Symbole der Hälfte von $R_{j_0}^{(i)}$ werden in unveränderter Reihenfolge auf die rechten Hälften von $R_1^{(i)}, \dots, R_{j_0-1}^{(i)}$ geschrieben.
 - Die von # verschiedenen Symbole der Abschnitte $L_1^{(i)}, \dots, L_{j_0}^{(i)}$ werden in unveränderter Reihenfolge auf den Abschnitt $L_{j_0}^{(i)}$ und die rechten Hälften der Abschnitte $L_1^{(i)}, \dots, L_{j_0-1}^{(i)}$ geschrieben und die Symbole in den linken Hälften der Abschnitte $L_1^{(i)}, \dots, L_{j_0-1}^{(i)}$ werden durch # überschrieben.
 - Schließlich wird das nun ersetzte Symbol, das zuvor im Abschnitt $H^{(i)}$ stand auf die rechte Hälfte des Abschnitts $L_1^{(i)}$ geschrieben.
- Bandbewegungen nach rechts werden analog simuliert.
- Die Bandbewegungen lassen sich eindeutig in dieser Weise simulieren da für alle $j \leq 1$ zu jeder Zeitpunk für die Abschnitte $L_j^{(i)}$ und $R_j^{(i)}$ folgendes gilt: Entweder sind beide Abschnitte halb voll oder einer ist leer und der andere voll.
- Beispiel hier!
- Die Laufzeit von U ergibt sich im Wesentlichen aus der Simulation der Bandbewegungen. Bei einer Bandbewegung ist die Anzahl der betrachteten Felder $O(\sum_{j=1}^{j_0} 2^j) = O(2^{j_0})$ für ein $j_0 \leq 1$. Dank des zweiten Bandes lassen solche Bandbewegungen in Zeit $O(2^{j_0})$ durchführen. Ist t die Laufzeit von M zur Eingabe w, so ist eine solche Bandbewegung höchstens $\frac{t}{2^{j_0}}$ und durchzuführen. Als Laufzeit von U ergibt sich so bis auf konstante Faktoren $\sum_{j_0=1}^{\log_2 t} \frac{t}{2^{j_0}} = t \log_2 t$.
 - $\frac{t}{2^{j_0}} \rightarrow$ Häufigkeit
 - $2^{j_0} \rightarrow$ Kosten

□

7.12 Satz (Universelle TM-Simulation mit Zeitbeschränkung)*

Ist für alle TM $M = (\{0, \dots, n\}, \{0, 1\}, \{\square, 0, 1\}, \Delta, 0, \{0\})$ und alle Wörter $w \in \{0, 1\}^*$ das Wort $code(M, w)$ ein geeigneter Code für (M, w) , so gibt es eine 2-TM U , so dass folgendes gilt.

- (i) \forall TM M wie oben und $\forall w \in \{0, 1\}^*$ akzeptiert U den Code $code(M, w)$ genau dann, wenn M das Wort w akzeptiert.
- (ii) Für alle Zeitschranken t , $\forall t$ - zeitbeschränkten TM M wie oben gibt es ein u_0 und c , so dass $\forall w \in \{0, 1\}^{\geq n_0}$ jede Rechnung von U zur Eingabe $code(M, w)$ Länge höchstens $ct(|w|)$ hat, falls U eine Rechnung von M zur Eingabe w simuliert, die von M akzeptiert wird.

Beweis. Die Idee ist die Bänder einer TM zu simulieren. Genauer verfährt U bei Eingabe $code(M, w)$ wie folgt:

- Zu Beginn werden einige Initialisierungsschritte ausgeführt, die es erlauben die k Bänder von M nacheinander auf dem ersten Band von U zu simulieren, wobei auf einer Spur auf dem ersten Band eine repräsentation von M erzeugt wird, die während der Simulation so verschoben wird, dass sie vor jedem zu simulierenden Schritt unmittelbar rechts von Kopf auf dem ersten Band steht.
- Die k -TM M wird mit w als Eingabe simuliert, indem nacheinander für jedes Band $i \in [k]$ alle Kopfbewegungen auf Band i bis zum Rechnungsende simuliert werden. Es werden also zunächst alle Kopfbewegungen auf dem ersten, dann auf dem zweiten Band usw. simuliert.
 - Dies erfordert im Sinne der Arbeitsweise von M Kenntnis der in jedem Schritt gelesenen Bandsymbole auf allen anderen Bändern und der gewählten Instruktion, weshalb diese zu Beginn der Simulation des ersten Bandes für die nicht simulierten Bänder nicht deterministisch geraten werden und auf dem zweiten Band gespeichert werden.
 - Bei der Simulation der weiteren Bänder wird dann geprüft, ob die tatsächlich dort auftretenden Bandsymbole zu den geratenen Bandsymbolen und Instruktion passen. Ist dies nicht der Fall \rightarrow Terminierung mit Nichtakzeptanz.
Andernfalls war die Simulation erfolgreich und die Eingabe wird genau dann akzeptiert, wenn die Simulation die Simulierte Eingabe akzeptiert hat.

Nach Konstruktion sieht man leicht, dass es ein $c, n_0 \in \mathbb{N}$ gibt, so dass U höchstens $ct(|w|)$ Schritte benötigt falls $|w| \geq n_0$. \square

7.13 Definition (Zeitbeschränkte Funktionen und Sprachen)*

Für eine Menge T von Funktionen von $\mathbb{N}_0 \rightarrow \mathbb{R}_{\geq 0}$ definieren wir

- $\mathbf{DTIME}(T) := \{L(M) \subseteq \{0, 1\}^* : \exists t \in T \text{ ist } M \text{ eine } t\text{-zeitbeschränkte DTM}\}$
- $\mathbf{FTIME}(T) := \{\varphi(M) : \mathbb{N}_0 \rightarrow \mathbb{N}_0 : \exists t \in T \text{ und } M \text{ ist eine } t\text{-zeitbeschränkte DTM}\}$
- $\mathbf{NTIME}(T) := \{L(M) \subseteq \{0, 1\}^* : \exists t \in T \text{ ist } M \text{ eine } t\text{-zeitbeschränkte TM}\}$

7.14 Definition (Zeitkonstruierbar)

Eine Zeitschranke t ist genau dann **Zeitkonstruierbar**, wenn es eine DTM M mit Eingabealphabet $\{1\}$ gibt, so dass $time_M(1^n) = t(n)$ gilt.

7.15 Bemerkung (Polynomielle Zeitkonstruierbarkeit)*

- (i) Ist p ein Polynom in einer Variable über \mathbb{Z} mit $p(n) \geq n + 1 \quad \forall n \in \mathbb{N}_0$, so ist $t : \mathbb{N}_0 \rightarrow \mathbb{N}_0, n \mapsto p(n)$ Zeitkonstruierbar.
- (ii) Ist t Zeitkonstruierbar, so ist auch $T : \mathbb{N}_0 \rightarrow \mathbb{N}_0, n \mapsto 2^{t(n)}$ Zeitkonstruierbar.

7.16 Satz (Zeithirarchiesatz für deterministische TM)

Sei t eine Zeitschranke und T eine Zeitkonsturierbare Zeitschranke mit $t(n) \log t(n) = o(T(n))$. Dann gilt $\mathbf{DTIME}(t(n)) \subsetneq \mathbf{DTIME}(T(n))$.

Beweis. Sei M_0, M_1, \dots eine geeignete effektive Aufzählung deterministische TM der Form $(\{0, \dots, n\}, \{0, 1\}, \{\square, 0, 1\}, \Delta, 0, \{0\})$ so dass \forall DTM M dieser Form es unendlich viele $e \in \mathbb{N}_0$ mit $M = M_e$ gibt.

Wir betrachten eine DTM U , die bei Eingabe 1^e die DTM M_e bei eingabe 1^e simuliert und genau dann akzeptiert, wenn die Simulation terminiert und nicht akzeptiert.

Nach Satz 7.11 können wir U so wählen, dass es $\forall c \geq 1$ und alle $ct(n) \log t(n)$ - zeitbeschränkter DTM M der obigen Form ein $e_{c1}M$ mit $M = M_{e_{c1}M}$, so dass die Rechnung von U zur Eingabe $1_{e_{c1}M}^e$ höchstens Länge $T(e_M, c)$ hat.

Da T zeitbeschränkt ist, ist es möglich U so zu modifizieren, dass U die Simulation immer nach $T(e)$ Schritten abbricht, wobei wir bei Abbruch nie akzeptieren.

Dann ist U offenbar $T(n)$ - zeitbeschränkt, es gilt also $L(U) \in \mathbf{DTIME}(T(n))$.

Wir zeigen nun, dass keine $t(n)$ - zeitbeschränkte DTM M existiert mit $L(M) = L(U)$

Ist M eine $t(n)$ - zeitbeschränkte DTM, so existiert nach Satz 7.4 ein $c \geq 1$ und eine $ct(n)$ - zeitbeschränkte DTM M' der obigen Form mit $L(M) = L(M')$.

Die TM U bricht dann die Simulation der Eingabe $1_{e_{c1}M'}^e$ nicht ab, es gilt also $L(M_{e_{c1}M'}) \neq L(U)$ und damit wegen $L(M_{e_{c1}M'}) = L(M') = L(M)$ also $L(U) \neq L(M)$. \square

7.17 Proposition (Zeitbeschränkte TM-Äquivalenz)*

Für alle $t : \mathbb{N}_0 \rightarrow \mathbb{R}_{\geq 0}$ und t - zeitbeschränkte TM M gibt es eine DTM M' mit $L(M) = L(M')$, sodass M' $2^{O(t)}$ - zeitbeschränkt ist.

Beweis. $\forall M$ wie oben gibt es $c \in \mathbb{N}_0$ sodass es bei Eingabe w höchstens $c^{t(|w|)}$ viele rechnungen zur Eingabe w gibt.

\rightsquigarrow Brute-Force \square

7.18 Satz (Zeithirarchiesatz für nichtdeterministische TM)

Sei t eine Zeitschranke und T eine zeitkonsturierbare Zeitschranke mit $t(n+1) = o(T(n))$. Dann gilt $\mathbf{NTIME}(t(n)) \subsetneq \mathbf{NTIME}(T(n))$.

Beweis. Sei M_0, M_1, \dots eine geeignete Aufzählung von TM der Form $M = (\{0, \dots, m\}, \{0, 1\}, \{\square, 0, 1\}, \Delta, 0, \{1\})$, so dass \forall TM M dieser Form unendlich viele $e \in \mathbb{N}$ mit $M = M_e$ gibt.

Sei $l_0 := 0$. Sei l_e für $e \geq 1$ deterministisch durch $l_e := 2^{T(l_{e-1})}$.

Für $e \in \mathbb{N}_0$, sei $I_e := (l_e, l_{e+1}) \cap \mathbb{N}_0$.

Wir betrachten eine TM U , die bei Eingabe 1^x wie folgt verfährt.

- Es wird $e \in \mathbb{N}_0$ mit $x \in I_e$ bestimmt.
- Gilt $x < l_{e+1}$, wird M_e bei Eingabe 1^{x+1} simuliert und dann genau akzeptiert, wenn die Simulation terminiert und akzeptiert.
- Gilt $x = l_{e+1}$, so werden alle Rechnungen von M_e bei Eingabe $1^{l_{e+1}}$ simuliert und es wird genau dann akzeptiert, wenn M_e nicht akzeptiert.

Da T Zeitkonsturierbar ist, lässt sich gegeben x , I_e (bzw. e) bestimmen. Nach Satz 7.12 können wir daher U so wählen, dass es $\forall x \leq 1$ und $\forall ct(n)$ - zeitbeschränkter TM M der obigen Form ein $e_{c,M}$ mit $M_{e_{c,M}} = M$ gibt, so dass alle rechnungen von U zur Eingabe 1^x mit $x \in I_{e_{c,M}} / \{l_{e_{c,M}+1}\}$ höchstens Länge $T(x)$ haben (falls sie eine akzeptierte Rechnung simulieren) und so dass die Rechnung von U zur Eingabe $1^{l_{e_{c,M}+1}}$ höchstens $2^{T(l_{e_{c,M}})} = l_{e_{c,M}+1} \leq T(l_{e_{c,M}+1})$ hat.

Da T Zeitkonsturierbar ist, ist es möglich U so zu modulieren, dass U die Simulation immer nach $T(x)$ Schritten abbricht, wobei U dann bei abgebrochener Simulation nie akzeptiert.

Dann ist U offenbar $T(n)$ - Zeitbeschränkt, es gilt also $L(U) \in \mathbf{NTIME}(T)$.

Wir zeigen nun, dass keine $t(n)$ - zeitbeschränkte TM M mit $L(U) = L(M)$ existiert.

Ist M eine $t(n)$ - zeitbeschränkte TM mit $L(M) = L(U)$, so existiert nach [Satz 7.4](#) ein $x \leq 1$ und eine $ct(n)$ - zeitbeschränkte TM M' der obigen Form mit

$$L(M') = L(M) = L(U)$$

Die TM U bricht dann die simulation bei allen Eingaben 1^x mit $x \in I_{e_c, M'}$ nicht ab, $\forall x \in I_{e_c, M'} / \{l_{e_c, M'+1}\}$ gilt also

$$1^x \in L(M') \Leftrightarrow 1^x / in L(U) \Leftrightarrow 1^{x+1} \in L(M')$$

Folgendes gilt

$$1^{l_{e_c, M'+1}} \in L_{M'} \Leftrightarrow 1^{l_{e_c, M'+1}} \in L(M') \Leftrightarrow \dots$$

Dies ist im Widerspruch zu $L(M') = L(M)$. □

Im Folgenden fixieren wir eine Aufzählung M_0, M_1, \dots aller normierten DTM.

7.19 Definition(abstraktes Komplexitätsmaß)

Eine partielle Funktion $\phi : \mathbb{N}_0 \times \{0, 1\}^* \rightsquigarrow \mathbb{N}_0$ heißt **abstraktes Komplexitätsmaß** falls folgendes gilt:

- (i) $\phi(e, x) \downarrow \Leftrightarrow M_e(x) \downarrow \quad \forall e \in \mathbb{N}_0, x \in \{0, 1\}^*$
- (ii) Die Relation $\{(e, x, t) \in \mathbb{N}_0 \times \{0, 1\}^* \times \mathbb{N}_0 : \phi(e, x) = t\}$ ist im Sinne entscheidbar, dass die Funktion $\psi : \mathbb{N}_0 \times \{0, 1\}^* \times \mathbb{N}_0 \rightarrow \{0, 1\}$ mit $\psi(e, x, t) = 1 \Leftrightarrow \phi(e, x) = t \quad \forall e, x, t$ berechenbar ist.

7.20 Definition(T-Beschränkte TM und DCOM)*

Sei $t : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ eine Funktion für $e \in \mathbb{N}_0$ wird die TM M_e als **t - ϕ - beschränkt** bezeichnet, wenn M_e total ist und es ein $n_0 \in \mathbb{N}_0$ gibt, sodass $\phi(e, x) \leq t(|x|) \quad \forall x \in \{0, 1\}^{\geq n_0}$ gilt. Wir setzen:

$$\mathbf{DCOM}^\phi(t) := \{L(M) \subseteq \{0, 1\}^* : M \text{ ist } t - \phi - \text{beschränkt}\}$$

Beispielsweise ist Zuordnung $e, x \mapsto time_{M_e}(x)$ ein abstraktes Komplexitätsmaß.

7.21 Satz (Lückensatz)

Sei ϕ ein abstraktes Komplexitätsmaß und sei $f, g : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ streng monoton wachsende Funktion. Dann gibt es eine streng monoton wachsende berechenbare Funktion $t : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ mit $f(n) \leq t(n) \quad \forall n \in \mathbb{N}_0$ und $\mathbf{DCOMP}^\phi(t) = \mathbf{DCOMP}^\phi(g \circ t)$.

Beweis. Wir konstruieren t so, dass es $\forall e \in \mathbb{N}_0$ ein $n_0 \in \mathbb{N}_0$ gibt, so dass $\phi(e, x) \notin [t(|x|) + 1, g(t(|x|))] \quad \forall x \in \{0, 1\}^{\geq n_0}$ gilt.

SKIZZEHIER

Dafür genügt es $t(n)$ für $n \in \mathbb{N}_0$ so zu wählen, dass $\phi(e, x) \notin [t(n) + 1, g(t(n))] \quad \forall e < n$ und $x \in \{0, 1\}^{=n}$ gilt. $\forall n \in \mathbb{N}_0$ ist $\{\phi(e, x) : e < n, x \in \{0, 1\}^{=n}\}$ eine endliche Menge.

\Rightarrow (hier noch limits eintragen!!!) effektive Bestimmung einer abstrakten Schranke möglich.

$\Rightarrow t(n)$ groß genug wählen genügt

Genauer: $t(n) := \min\{m \in \mathbb{N}_0 : t(n-1) < m \wedge f(n) < m \wedge \forall e < n : \forall x \in \{0, 1\}^{=n} : \phi(e, x) \in [m+1, g(m)]\}$.

$\Rightarrow t$ ist berechenbar, streng monoton wachsend und $f(n) \leq t(n)$

Sei $L \in \mathbf{DCOMP}^\phi(g \circ t)$. Sei $e \in \mathbb{N}_0$ so dass M_e eine $(g \circ t)$ - ϕ - zeitbeschränkte TM ist mit $L(M_e) = L$.

Wir zeigen nun, dass M_e t - ϕ - beschränkt ist. Sei $x \in \{0, 1\}^{\geq e+1}$. Es genügt zu zeigen, dass $\phi(e, x) \leq t(|x|)$.

Nach Konstruktion von t gilt $\phi(e, x) \notin [t(|x|) + 1, g(t(|x|))]$ und nach Wahl von e gilt $\phi(e, x) \leq g(t(|x|))$.

Es folgt also $\phi(e, x) \leq t(|x|)$. □

Komplexitätsklassen P und NP

8.1 Definition(Komplexitätsklassen P, FP und NP)*

Wir setzen

- $\mathbf{P} := \mathbf{DTIME}(\text{poly})$,
- $\mathbf{FP} := \mathbf{FTIME}(\text{poly})$,
- $\mathbf{NP} := \mathbf{NTIME}(\text{poly})$

wobei $\text{poly} := \{n^c + c, c \in \mathbb{N}_0\}$ Offenbar gilt $\mathbf{P} \subseteq \mathbf{NP}$, allerdings ist unklar ob $\mathbf{P} \not\subseteq \mathbf{NP}$.

8.2 Definition (p-m-Reduktion)

Eine Sprache A über $\{0, 1\}$ ist in **polynomieller Zeit many-to-one-reduzierbar**, auch **p-m-reduzierbar**, kurz $A \leq_m^P B$, auf eine Sprache B über $\{0, 1\}$, wenn es eine Funktion $f \in \mathbf{FP}$ gibt, so dass

$$w \in A \Leftrightarrow f(w) \in B, \quad \forall w \in \{0, 1\}^*$$

gilt. Gelte $A \leq_m^P B$ und $B \leq_m^P C$, so sind A und C **p-m-äquivalent**, kurz $A =_m^P C$

8.3 Bemerkung(Eigenschaften der p-m-Reduktion)*

- (i) \leq_m^P transitiv
- (ii) Seien A und B Sprachen mit $A \leq_m^P B$. Aus $B \in \mathbf{P}$, folgt $A \in \mathbf{P}$ und aus $B \in \mathbf{NP}$ folgt $A \in \mathbf{NP}$.
- (iii) Alle Sprachen $L \in \mathbf{P}$ mit $\emptyset \neq L \neq \{0, 1\}^*$ sind p-m-äquivalent.

8.4 Definition (NP-schwer, NP-vollständig)

Eine Sprache S wird als **NP-Schwer** bezeichnet, wenn $L \leq_m^P S \quad \forall L \in \mathbf{NP}$ gilt. Gilt zusätzlich $S \in \mathbf{NP}$, so wird S als **NP-vollständig** bezeichnet.

Nächstes Ziel: Finden eines NP-vollständigen Problems.

8.5 Definition (aussagenlogische Formel)

Sei Var eine abzählbare Menge mit $\neg, \Delta, \notin \text{Var}$. Die Menge der **aussagenlogischen Formeln** A ist die kleinste Inklusion kleinster Menge von Wörtern über $\text{Var} \cup \{\neg, \wedge, (\cdot, \cdot)\}$ mit $a \in A \quad \forall a \in \text{Var}$ und $\neg\varphi, (\varphi \wedge \psi) \in A \quad \varphi, \psi \in A$. Für $\varphi, \psi \in A$ schreiben wir statt $\neg(\neg\varphi \wedge \neg\psi)$ auch $(\varphi \vee \psi)$ und statt $(\neg\varphi \vee \psi)$ auch $(\varphi \rightarrow \psi)$. Wie üblich vereinbaren wir Klammerregeln zur besseren Lesbarkeit: \neg bindet stärker als \wedge und \vee .

Beispielsweise $\neg a \wedge b \wedge c = ((\neg a \wedge b) \wedge c)$

8.6 Definition(Grundbegriffe der aussagenlogischen Formeln)*

- (i) Die Elemente von Var heißen **Variablen**.
- (ii) Für $\varphi, \varphi_1, \dots, \varphi_n \in A$ mit $n \in \mathbb{N}$ heißt die aussagenlogische Formel $\neg\varphi$ **Negation** von φ , die aussagenlogische Formel $\varphi_1 \wedge \dots \wedge \varphi_n$ heißt **Konjunktion** von $\varphi_1, \dots, \varphi_n$ und $\varphi_1 \vee \dots \vee \varphi_n$ heißt **Disjunktion** von $\varphi_1, \dots, \varphi_n$.

8.7 Definition (konjunktive Normalform)

- (i) Ein **Literal** ist eine variable oder eine negation einer Variablen.
- (ii) Eine **disjunktive Klausel** ist eine Disjunktion von Literalen.
- (iii) Eine aussagenlogische Formel in **konjunktiver Normalform**, kurz KNF ist eine Konjunktion disjunktiver Klauseln

8.8 Definition (Wahrheitswert)

Eine **Belegung** ist eine Funktion $b : Var \rightarrow \{0, 1\}$. Der **Wahrheitswert** $val_b(\varphi)$ einer Formel φ für eine Belegung b ist induktiv wie folgt definiert.

$\forall a \in Var$ sei $val_b(a) = b(a)$ und für $\varphi, \psi \in A$ sei

$$val_b(\neg\varphi) := \begin{cases} 1 & val_b(\varphi) = 0 \\ 0 & val_b(\varphi) = 1 \end{cases} \quad (1)$$

und

$$val_b(\varphi \wedge \psi) := \begin{cases} 1 & val_b(\varphi) = val_b(\psi) = 1 \\ 0 & \text{sonst} \end{cases} \quad (2)$$

Eine aussagenlogische Formel φ heißt **erfüllbar**, wenn es eine Belegung b gibt, sodass $val_b = 1$ gilt.

8.9 Definition (logisch äquivalent)

Zwei aussagenlogische Formeln ψ und φ sind **logisch äquivalent** falls $val_b(\varphi) = val_b(\psi) \quad \forall$ Belegungen b gilt.

8.10 Definition (SAT)

Wir setzen

$$SAT := \{\varphi \in A : \varphi \text{ ist in KNF und erfüllbar}\}$$

8.11 Bemerkung (Das Erfüllbarkeitsproblem SAT in der Komplexitätsklasse NP)*

Es gilt $SAT \in NP$.

8.12 Satz (Satz von Cock)

Die Sprache SAT ist **NP**-vollständig.

Beweis. Nach [Bemerkung 8.11](#) gilt $\text{SAT} \in \mathbf{NP}$. Es genügt also zu zeigen, dass es $\forall p \in \text{poly}$ und p-zeitbeschränkten TM M eine Funktion $g \in \mathbf{FP}$ gibt, sodass $\forall w \in \{0,1\}^*$ die aussagenlogische Form $g(w)$ in KNF genau dann erfüllbar ist, wenn $w \in L(M)$. Dafür wählen wir $g(w)$ so, dass $g(w)$ in wesentlich der Aussage u entspricht. Sei $c \in \mathbb{N}_0$ und $p: \mathbb{N} \rightarrow \mathbb{N}$, $n \rightarrow n^c + c$ und sei $M = (Q, \{0,1\}, \Gamma, \Delta, s, F)$ eine p-zeitbeschränkte TM. [Satz 7.10](#) und [Satz 7.4](#) erlauben uns anzunehmen, dass M eine 1-TM mit Bandalphabet $\Gamma = \{\square, 0, 1\}$ ist. Sei $w \in \{0,1\}^*$ und $n := |w|$.

Wir setzen $I := [p(n)]$ und $J := \{-p(n) + 1, \dots, p(n) + 1\}$. Wir definieren nun eine ganze Ansammlung an Variablen, die die Arbeitsweise von M bei Eingabe w codieren.

Variable	Aussage
$T_{i,j,a}$ mit $i \in I, j \in J, a \in \Gamma$	Zum Zeitpunkt i ist a das Symbol auf Feld j
$p_{i,j}$ mit $i \in I, j \in J$	Zum Zeitpunkt i steht der Kopf auf dem Feld j
$S_{i,q}$ mit $i \in I, q \in Q$	Zum Zeitpunkt ist der Zustand q
$D_{i,0}$	Der i -te 'Listeneintrag' ist eine Wiederholung einer vorangehenden Stoppkonfiguration.
$D_{i,d}$ mit $i \in I$ und $d \in \Delta$	Die Instruktion d bezeugt im Sinne Definition 2.3 (Nachfolgekonfiguration) , dass der $(i+1)$ -te 'Listeneintrag' Nachfolgekonfiguration des i -ten 'Listeneintrags' ist

\rightsquigarrow Wir haben eine Liste/ Logbuch mit den Variablen $D_{i,0}$, $D_{i,d}$, die im i -ten Feld beschreiben was M im i -ten Schritt der Rechnung macht.

Es bleibt $g(w)$ zu konstruieren, so dass $\text{val}_b(g(w)) = 1$ für eine Belegung b gilt, wenn b eine Liste darstellt, die einer Rechnung von M zur Eingabe w (gegebenenfalls mit Wiederholungen der Stoppkonfiguration) entspricht.

Wir geben dazu $g(w)$. Teilformeln die wir nicht als KNF angeben, lassen sich leicht in solche umwandeln.

Die Startkonfiguration wird korrekt realisiert:

$$S_{1,s}, \quad P_{1,1}, \quad \bigwedge_{j \in [n]} T_{1,j,w(j)}, \quad \bigwedge_{j \in [n]} T_{1,j,\square}$$

Zu jedem Zeitpunkt sind die relevanten Felder mit höchstens einem Symbol beschriftet:

$$\bigwedge_{(i,j,a,a') \in I \times J \times \Gamma \times \Gamma: a \neq a'} (T_{i,j,a} \rightarrow \neg T_{i,j,a'})$$

Zu jedem Zeitpunkt gibt es höchstens eine aktuelle Kopfposition.

$$\bigwedge_{(i,j,j') \in I \times J \times J: j \neq j'} (P_{i,j} \rightarrow \neg P_{i,j'})$$

Zu jedem Zeitpunkt gibt es höchstens einen aktuellen Zustand

$$\bigwedge_{(i,q,q') \in I \times Q \times Q: q \neq q'} (S_{i,q} \rightarrow \neg S_{i,q'})$$

Zu jedem Zeitpunkt gibt es höchstens eine auszuführende Instruktion oder es wird die vorherige wiederholt:

$$\bigwedge_{(i,d,d') \in I \times \Delta^+ \times \Delta^+: d \neq d'} (D_{i,d} \rightarrow \neg D_{i,d'})$$

Zu jedem Zeitpunkt kann sich nur das Feld ändern auf dem der Kopf steht:

$$\bigwedge_{(i,j,a) \in I^- \times J \times \Gamma} ((T_{i,j,a} \wedge \neg P_{i,j}) \rightarrow T_{i+1,j,a})$$

Wird eine Stoppkonfiguration wiederholt, ändert sich das Band, Kopfposition und Zustand nicht:

$$\bigwedge_{(i,j,a) \in I^- \times J \times \Gamma} ((D_{i,0} \wedge T_{i,j,a}) \rightarrow T_{i+1,j,a})$$

$$\bigwedge_{(i,j) \in I^- \times J} ((D_{i,0} \wedge P_{i,j}) \rightarrow P_{i+1,j})$$

$$\bigwedge_{(i,q) \in I^- \times Q} ((D_{i,0} \wedge S_{i,q}) \rightarrow S_{i+1,q})$$

Zu dem Zeitpunkt gibt es eine auszuführende Instruktion oder es wird die vorherige Konfiguration wiederholt:

$$\bigwedge_{i \in I} \bigvee_{d \in \Delta^+} D_{i,d}$$

Konfigurationen, die den Bedingungsteil einer Instruktion erfüllen werden nicht wiederholt:

$$\bigwedge_{(i,j(q,a,a',a',B)) \in I^- \times J \times \Delta} ((S_{i,q} \wedge P_{i,j} \wedge T_{i,j,a}) \rightarrow \neg D_{i,0})$$

Gibt es zu einer Konfiguration eine auszuführende Instruktion, so muss deren Bedingungsteil durch die Konfiguration erfüllt werden

$$\bigwedge_{(i,(q,a,q',a',B)) \in I^- \times \Delta} D_{i,(q,a,q',a',B)} \rightarrow S_{i,q}$$

$$\bigwedge_{(i,j,(q,a,q',a',B)) \in I^- \times J \times \Delta} (D_{i,(q,a,q',a',B)} \wedge P_{i,j} \rightarrow T_{i,j,a})$$

Wird eine Instruktion angewendet, so muss die Nachfolgekonfiguration aus dieser Instruktion aus der Vorgängerkonfiguration hervorgehen:

$$\bigwedge_{(i,(q,a,q',a',B)) \in I^- \times \Delta} (D_{i,(q,a,q',a',B)} \rightarrow S_{i+1,q'})$$

$$\bigwedge_{(i,j,(q,a,q',a',B)) \in I^- \times J \times \Delta} (D_{i,(q,a,q',a',B)} \wedge P_{i,j} \rightarrow T_{i+1,j,a})$$

$$\bigwedge_{(i,j,(q,a,q',a',B)) \in I^- \times J \times \Delta} (D_{i,(q,a,q',a',B)} \wedge P_{i,j} \rightarrow P_{i+1,j,+ \delta_B})$$

wobei $\delta_L := -1$, $\delta_S := 0$, $\delta_R := 1$.

Es wird eine Stoppkonfiguration erreicht und zum Zeitpunkt $p(n)$ ist der Zustand akzeptierend:

$$D_{p(n),0}, \bigvee_{q \in F} S_{p(n),q}.$$

Nach Konstruktion ist damit $g(w)$ genau dann erfüllbar, wenn es eine endliche Rechnung von M zur Eingabe w gibt, die höchstens Länge $p(n)$ hat, also genau, wenn $w \in L(M)$ gilt. \square

8.13 Definition (k-konjunktive Normalform)

Für $k \in \mathbb{N}$ ist eine aussagenlogische Formel in k-konjunktive Normalform, kurz k-KNF, falls sie eine Konjunktion disjunktiver Klauseln, die jeweils die Länge höchstens k haben, ist.

8.14 Definition (k-SAT)

Für $k \in \mathbb{N}$

$$k\text{-SAT} := \{\varphi \in A : \varphi \text{ ist im k-KNF und erfüllbar}\}$$

8.15 Bemerkung(k-SAT als Entscheidungsproblem in der Komplexitätsklasse NP)*

Für $k \in \mathbb{N}$ gilt $k\text{-SAT} \in \text{NP}$.

8.16 Satz (NP-Vollständigkeit von k-SAT für $k \leq 3$)*

Für $k \leq 3$ ist k-SAT NP- vollständig.

SKIZZEHIER

$$(x \vee y)$$

↓

$$\left(\frac{x}{\quad} \vee a\right) \wedge \left(\neg a \vee \frac{\quad}{y}\right)$$

Nach [Bemerkung 8.15](#) genügt es SAT \leq_m^p 3-SAT. Wir müssen also eine beliebige aussagenlogische Formel φ in KNF in eine Formel in 3-KNF überführen, sodass die $\varphi \in \text{SAT} \Leftrightarrow \varphi \in 3\text{-SAT}^{***}$. Sei $\varphi \in \text{SAT}$ und wir konstruieren φ' in 3-SAT wie folgt:

Wir geben für jede Klausel von φ eine Konjunktion von Klauseln für φ an, die logisch äquivalent ist. Sei $(l, v \cdots v l_r)$ eine Klausel von φ . Sei $r \leq 4$, sonst ist nichts zu tun.

$$(l, v a_2) \wedge \left(\bigwedge_{j=2}^{r-1} (\neg a_j \wedge l_j \wedge a_{j+1}) \right) \wedge (\neg a_r \wedge l_r) \quad \circledast$$

Hier sind a_2, \dots, a_r neue Variablen.

Wir fügen \circledast für $(l, v \cdots v l_r)$ zu φ' hinzu und verfahren für jede andere Klausel analog. Nun kann man überprüfen, dass φ und φ'

SKIZZEHIER

8.17 Definition (Graphen (V, E))*

Ein **Graph** ist ein Paar (V, E) , wobei V eine endliche Menge ist, die **Eckenmenge**, und $E \subseteq 2^V$ eine Menge zweielementiger Mengen, die **Kantenmenge**, ist.

8.18 Definition (k-Färbung)

Für $k \in \mathbb{N}$ ist eine k-Färbung eines graphen (V, E) eine Funktion $\phi: V \rightarrow [k]$, sodass $\phi(u) \neq \phi(v) \quad \forall \{u, v\} \in E$ gilt.

8.19 Definition (k-COLORING)

Es sei

$$\text{COLORING} := \{(G, k) : \text{es gibt eine } k\text{-färbung des Graphen } G\}$$

und für $k \in \mathbb{N}$ sei

$$k\text{-COLORING} := \{G : \text{es gibt eine } k\text{-Färbung für den Graphen } G\}$$

8.20 Bemerkung(NP-Eigenschaft von COLORING und k-COLORING)*

Es gilt $\text{COLORING} \in \text{NP}$ und für $k \in \mathbb{N}$ gilt $k\text{-COLORING} \in \text{NP}$.

8.21 Satz(NP-Vollständigkeit von 3-Coloring)*

Das Problem 3-Coloring ist NP-vollständig.

Beweis. Nach [Bemerkung 8.20](#) und [Satz 8.16](#) genügt es $3\text{-SAT} \leq_m^P 3\text{-COLORING}$ zu zeigen. Wir müssen also eine aussagenlogische Formel φ in einen Graphen G transformieren ...

φ in 3-KNF

□

8.22 Korollar (k-COLORING NP-vollständig für $k \leq 3$)*

Für $k \leq 3$ ist $k\text{-COLORING}$ NP-vollständig.

Beweis. Übung

□

8.23 Korollar(COLORING NP-vollständig)*

COLORING ist NP-vollständig.

8.24 Definition (EXACTCOVER)

Für eine Menge S von Mengen ist eine exakte Überdeckung von S eine Teilmenge $T \subseteq S$ paarweise disjunktive Menge mit $\bigcup_{A \in T} A = \bigcup_{A \in S} A$.

$$\text{EXACTCOVER} := \{S : S \text{ ist eine endliche Menge endlicher Mengen mit einer exakten Überdeckung}\}$$

8.25 Bemerkung (EXACTCOVER NP Eigenschaft)*

EXACTCOVER $\in \text{NP}$

8.26 Satz (EXACTCOVER NP-vollständig)*

EXACTCOVER ist NP-vollständig.

ohne beweis.

8.27 Definition (SUBSETSUM)

Sei

$$\text{SUBSETSUM} := \{(a_1, \dots, a_n, b) \in \mathbb{N}_0^{n+1} : \exists x_1, \dots, x_n \in \{0, 1\} : \sum_i x_i a_i = b\}$$

8.28 Bemerkung(SUBSETSUM NP Eigenschaft)*

SUBSETSUM $\in \text{NP}$

8.29 Satz(SUBSETSUM NP-vollständig)*

SUBSETSUM ist **NP**-vollständig.

ohne beweis.

Platzkomplexität

9.1 Definition (TM vom Offline-Typ)

Eine **TM vom Offline-Typ** ist eine k -TM $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ mit $k \geq 3$ und wie folgt angepasster/eingeschränkter Arbeitsweise.

1. Auf dem ersten band, dem **Eingabeband**, wird der Kopf nur auf die mit der Eingabe beschriebenen und die beiden unmittelbar angrenzenden Felder bewegt und es werden keine Symbole durch andere ersetzt.
2. Auf dem k -ten Band, den **Azsgabeband**, wird der Kopf nie nach links bewegt.
3. die Ausgabe von M bei Konfig $(q, w_1, \dots, w_k, p_1, \dots, p_k)$ ist das Präfix w von $w_k(1) \dots w_k(|w_k|)$ maximaler Länge mit $w \in (\Gamma/\{\square\})^*$.

Die Bänder $2, \dots, k-1$ sind die Arbeitsbänder.

9.2 Definition(Platzbedarf)

Sei $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ eine DTM vom Offline Typ. Es bezeichne $space_M : \Sigma^* \mapsto \mathbb{N}_0$ die partielle Funktion mit $dom(space_M) = dom(\varphi_M)$, so dass $space_M(w)$ für alle $w \in dom(\varphi_M)$ die maximale Anzahl der auf einem einzelnen Arbeitsband von Kopf besuchten Felder ist. Für $w \in \Sigma^*$ heißt $space_M(w)$ der **Platzbedarf** von M bei Eingabe w .

9.3 Definition (platzbeschränkt)

Eine **platzschränke** ist eine berechenbare Funktion $s : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ mit $s(n) \geq \log n$ für alle $n \in \mathbb{N}$ und $s(0) \geq 1$. Sei $s : \mathbb{N}_0 \rightarrow \mathbb{R}_{\geq 0}$ eine Funktion. Eine TM $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ ist **s-platzbeschränkt**, wenn M total ist und es ein $n_0 \in \mathbb{N}$ gibt, so dass $\forall w \in \Sigma^{\geq n_0}$ in alle Rechnungen von M auf allen Arbeitsbändern höchstens $s(|w|)$ Felder besucht werden.

9.4 Satz(lineare Kompression)

Sei $c > 1$ und sei $s : \mathbb{N}_0 \rightarrow \mathbb{R}_{\geq 0}$ mit $s(n) \geq 1$ für alle $n \in \mathbb{N}_0$. Ist M eine $c \cdot s(n)$ -platzbeschränkte TM mit k Arbeitsbändern, so gibt es eine $s(n)$ -platzbeschränkte TM M' mit k -Arbeitsbändern und $L(M') = L(M)$.

Beweis. Speichere mehrere Felder in einem Feld durch ein größeres Alphabet. □

9.5 Satz(Alphabetwechsel)

Ist M eine $s(n)$ -platzbeschränkte TM mit k Arbeitsbändern und Eingabealphabet $\{0,1\}$, so gibt es eine Konstante $c \geq 1$ und eine $c \cdot s(n)$ -platzbeschränkte TM M' mit k Arbeitsbändern, Eingabealphabet $\{0,1\}$, Bandalphabet $\{\square, 0, 1\}$ und $L(M') = L(M)$.

Wie bei $time_M$ handelt es sich auch bei $space_M$ um ein abstraktes Komplexitätsmaß. Insbesondere gilt die Aussage des Lückensatzes auch für platzbeschränkt.

9.6 Satz(Abstraktes Komplexitätsmaß)*

Sei $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ eine TM. Die partielle Funktion $space_M$ ist ein abstraktes Komplexitätsmaß

Beweis. Übung. □

9.7 Satz (Spurentechnik und DTM)*

Ist für alle (deterministische) TM der Form $M = (\{0, \dots, n\}, \{0, 1\}, \{\square, 0, 1\}, \Delta, 0, \{1\})$ und alle wörter $w \in \{0, 1\}^*$ das Wort $\text{code}(M, w)$ ein geeigneter Code (M, w) , so gibt es eine (deterministische) TM mit einem Arbeitsband, so dass folgendes gilt:

- (i) \forall TM M wie oben und $\forall w \in \{0, 1\}^*$ akzeptiert U die eingabe $\text{code}(M, w)$ genau dann wenn M das Binärwort w akzeptiert.
- (ii) Für alle Platzschränken s und alle s -platzbeschränkten TM M wie oben gibt es ein $c_1, n_0 \in \mathbb{N}$, sodass alle wörter $w \in \{0, 1\}^{\geq n_0}$ in der Rechnung von U zur Eingabe $\text{code}(M, w)$ auf dem Arbeitsband höchstens $cs(|w|)$ Felder besucht werden.

Beweis. Die TM U arbeitet wie im Sinne bei der Normierung verwendete Spurentechnik, wobei bei Eingabe $\text{code}(M, w)$ auf einer zusätzlichen Spur eine geeignete Darstellung von M erzeugt wird. Dies hat konstante Länge, für hinreichendlanges w werden auf dieser Spur also weniger als $s(|w|)$ Felder benötigt.

Ist M deterministisch, so kann U auch deterministisch definiert werden. □

9.8 Definition (Klassen der Platzbeschränkten TM)*

Für eine Menge S von Funktionen von \mathbb{N}_0 nach $\mathbb{R}_{\geq 0}$ definieren wir:

- $\mathbf{DSPACE}(s) := \{L(M) \subseteq \{0, 1\}^* : \text{Für } \text{eins} \in \text{SistMeines} - \text{platzbeschränkteDTM}\}$
- $\mathbf{FSPACE}(s) := \{\varphi_M : \{0, 1\}^* \rightarrow \mathbb{N}_0 : \text{Für } \text{eins} \in \text{SistMeines} - \text{platzbeschränkteDTM}\}$
- $\mathbf{NSPACE}(s) := \{L(M) \subseteq \{0, 1\}^* : \text{Für } \text{eins} \in \text{SistMeines} - \text{platzbeschränkteTM}\}$
- $\mathbf{CONSPACE}(s) := \{\{0, 1\}^*/L(M) : \text{Für } \text{eins} \in \text{SiseMeines} - \text{platzbeschränkteTM}\}$

Für eine Funktion $s : \mathbb{N}_0 \rightarrow \mathbb{R}_{\geq 0}$ setzen wir $\mathbf{DSPACE}(s) := \mathbf{DSPACE}(\{s\})$ und analog für die anderen Klassen.

9.9 Definition (Platzkonstruierbar)

eine Platzschränke s ist genau dann **platzkonstruierbar**, wenn es eine DTM M mit Eingabealphabet $\{1\}$ gibt, sodass, $\text{space}_M(1^n) = s(n)$ für alle $n \in \mathbb{N}_0$ gilt.

9.10 Bemerkung (Platzkonstruierbare Funktionen: Eigenschaften)*

- (i) Die Platzschränke s mit $s(0) = 1$ und $s(n) = \lfloor \log n \rfloor$ für $n \in \mathbb{N}$ ist platzkonstruierbar
- (ii) Ist p ein Polynom in einer Variable über \mathbb{Z} mit $p(n) \geq n + 1$ für alle $n \geq \mathbb{N}_0$, so ist $s : \mathbb{N}_0 \rightarrow \mathbb{N}_0, \quad n \mapsto p(n)$ platzkonstruierbar.
- (iii) ist s platzkonstruierbar, so ist auch $s : \mathbb{N}_0 \rightarrow \mathbb{N}_0, \quad n \mapsto 2^{s(n)}$ platzkonstruierbar.

9.11 Satz (Platzhierarchiesatz für DTM)

Sei s eine Platzschränke und S eine platzkonstruierbare Platzschränke mit $s(n) = o(S(n))$. Dann gilt $\mathbf{DSPACE}(s(n)) \subsetneq \mathbf{DSPACE}(S(n))$.