

EINFÜHRUNG IN DIE THEORETISCHE INFORMATIK - SCRIPT

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



Inhaltsverzeichnis

1 Grundlagen	4
1.1 Notationen und begriffe	4
1.2 Alphabet, Wörter und Sprachen	4
1.2.1 Definition (Alphabet)	4
1.2.2 Definition (Wörter)	5
1.2.3 Definition (Binäraphabet, Binärwörter)	5
1.2.4 Sprache	5
1.2.5 Definition	5
1.2.6 Definition (Verkettung)	5
1.2.7 Definition (Präfix, Infix, Suffix)	5
1.2.8 Definition (präfixfrei)	5
1.2.9 Definition (Homomorphismus)	6
1.2.10 Definition (Längenlexikographische Ordnung)	6
1.2.11 Bemerkung	6
1.2.12 Definition	6
1.2.13 Bemerkung	6
2 Turingmaschine	7
2.1 Definition (Turingmaschine, Alan Turing, 1936)	8
2.2 Definition (Konfiguration)	9
2.3 Definition (Nachfolgekonfiguration)	9
2.4 Definition (Rechnung)	9
2.5 Bemerkung	9
2.6 Definition (total)	9
2.7 Definition (akzeptierte Sprache)	10
2.8 Definition(entscheidbar)	10
2.9 Definition(rekursiv aufzählbar)	10
2.10 Bemerkung	10
2.11 Bemerkung	10
2.12 Bemerkung	10
2.13 Definition (Ausgabe)	11
2.14 Definition (berechnete Funktion)	11
2.15 Definition (partiell berechenbar)	11
2.16 Definition (charakteristische Funktion, partielle charakteristische Funktion)	11
2.17 Bemerkung	11
2.18 Bemerkung (normiert)	11
2.19 Bemerkung	13
2.20 Church- Turing- These	13
3 Berechenbarkeit	14
3.1 Definition (Code)	15
3.2 Definition (standardaufzählung)	15
3.3 Bemerkung	15
3.4 Definition (U)	15
3.5 Definition (Universell)	16
3.6 Bemerkung	16
3.7 Satz (s_n^m - Theorem)	16
3.8 Definition (diagonales Halteproblem)	17
3.9 Proposition	17
3.10 Satz	17
3.11 m-Reduktion	17

3.12	Bemerkung	17
3.13	Satz	18
3.14	Definition (Postsches Korrespondenzproblem, Emil Post, 1946)	18
3.15	Lemma	19
3.16	Lemma	19
3.17	Lemma	20
3.18	Beispiel	21
3.19	Satz	21
3.20	Fixpunktsatz, Rekursionstheorem und Satz von Rice	21
3.20.1	Definition (Fixpunktsatz)	21
3.20.2	Satz (Fixpunktsatz, Hartley Rogers jr., 1967)	21
3.20.3	Satz (Rekursionstheorem, Stephen Cole Kleen, 1938)	22
3.20.4	Korollar	22
3.20.5	Definition (Indexmenge)	22
3.20.6	Satz (Satz von Rice, Henry Horden Rice, 1951)	22
4	Automaten	23
4.1	Definition (Endliche Automaten)	24
4.2	Definition (Übergangsfunktion eines EA)	24
4.3	Bemerkung	24
4.4	Definition (Übergangsfunktion eines DEA)	25
4.5	Bemerkung	25
4.6	Bemerkung	25
4.7	Definition (akzeptierte Sprache)	25
4.8	Definition (regulär)	25
4.9	Beispiel	25
4.10	Definition (Potenzautomaten)	26
4.11	Satz	26
5	Reguläre Sprachen	27
5.1	Definition (Äquivalenzrelation)	28
5.2	Definition (A-Äquivalenz)	28
5.3	Bemerkung	28
5.4	Definition (Rechtskongruenz)	28
5.5	Proposition	28
5.6	Definition	28
5.7	Lemma	29
5.8	Satz	29
5.9	Korollar	29
5.10	Definition(erreichbar)	29
5.11	Definition(isomorph)	29
5.12	Satz	30
5.13	Definition (L-Äquivalenz)	30
5.14	Bemerkung	30
5.15	Definition (Parition)	30
5.16	Definition (Vereinfachung)	30
5.17	Bemerkung	31
5.18	Proposition	31
5.19	Definition (Minimalautomat)	31
5.20	Satz	31
5.21	Satz (Satz von Myhill und Nerode)	32
5.22	Satz (Pumping-Lemma)	32
5.23	Beispiel	32

6 Formale Grammatiken	33
6.1 Definition (Grammatiken)	34
6.2 Definition (Ableitung)	34
6.3 Definition (Erzeugte Sprache)	34
6.4 Lemma	35
6.5 Satz	35
6.6 Rechtslinear	35
6.7 Satz	35
7 Bilder	36

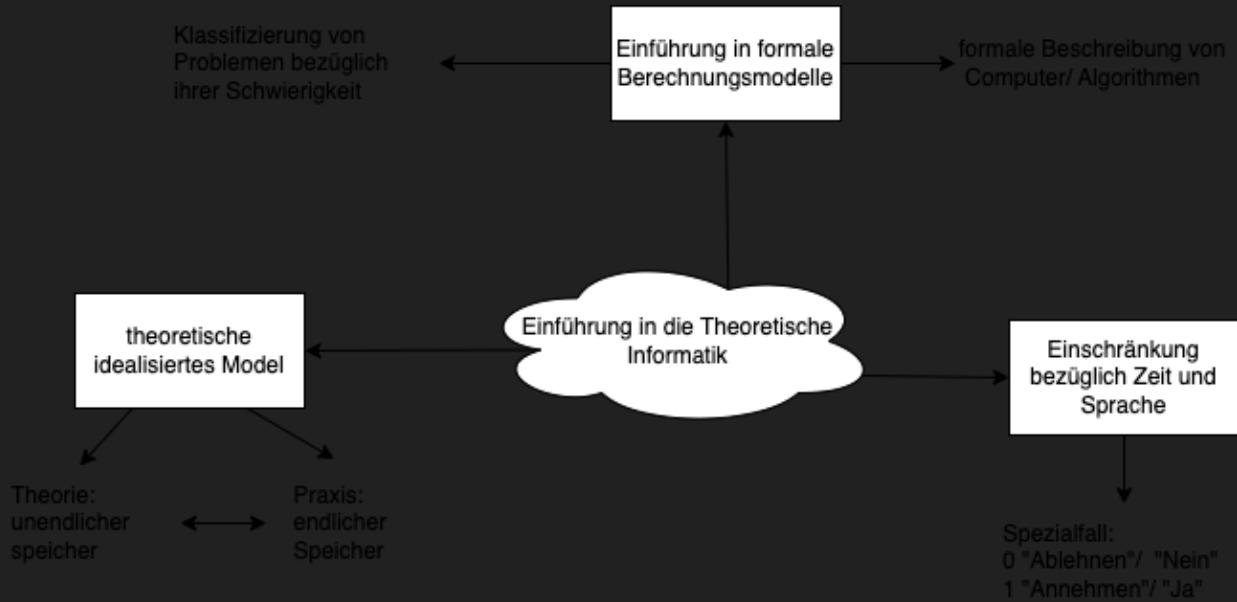


Abbildung 1: Überblick theoretische Informatik

1 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 $\text{dom}(\varphi) \supseteq A^n$ 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 $\leq \subseteq A^n$ 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 $(a, b) \in \leq$):
 - $a \leq a \forall a \in A$ (Reflexivität)
 - $a \leq b \wedge b \leq a \Rightarrow a = b \forall a, b \in A$ (Antisymmetrie)
 - $a \leq b, b \leq c \Rightarrow a \leq c$ for all $a, b, c \in A$ (Transitivität)
 - $a \leq b \vee b \leq a \forall a, b \in A$ (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 $w(i)$ 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 (Binäraphabet, Binärwörter)

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

1.2.4 Sprache

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

1.2.5 Definition

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 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^k induktiv definiert durch $w^0 := \lambda$ falls $n = 0$ und $w^n := w^{n-1} \circ w^n$ falls $n \geq 1$. Für eine Sprachen L_1, L_2 sei durch $L_1 \circ L_2$, auch $L_1 L_2$ definiert durch

$$L_1 \circ L_2 := \{w_1 w_2 : w_1 \in L_1, w_2 \in L_2\}$$

Für eine Sprache L und $n \in \mathbb{N}_0$ ist L^n moduliert definiert durch $L^0 = \{\lambda\}$ 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 $0;??$ 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 $v = w_1 uw_2$
- (iii) u ist Suffix von v , falls es ein Wort w gibt, sodass $v = uw$.

1.2.8 Definition (präfixfrei)

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

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_{llex} auf Σ^* die lineare Ordnung für die $u \leq_{llex} v$ 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 $u(i) \leq v(i)$.

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

1.2.11 Bemerkung

Sei Σ ein Alphabet $\forall w \in \Sigma^*$ ist $v \in \Sigma^* : v \leq_{llex} w$ 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.

$$\begin{array}{ccccc} \mathbb{N}_0 & 0 & 1 & 2 & 3 \\ \{0, 1\}^* & \lambda & 0 & 1 & 00 \end{array}$$

1.2.12 Definition

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

1.2.13 Bemerkung

$\forall i \in \mathbb{N}_0$ ist $bin(i)$ 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.

2 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 betrachten das folgende, sehr bekannt, berechnungsmodell. Anschaulich lässt es sich wie folgt beschreiben.

- Es gibt einen Speicher "↔ k unendlich lange Arrays (**Bänder**)
- Es gibt einen Arbeitsspeicher "↔ 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 Käpfe 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.

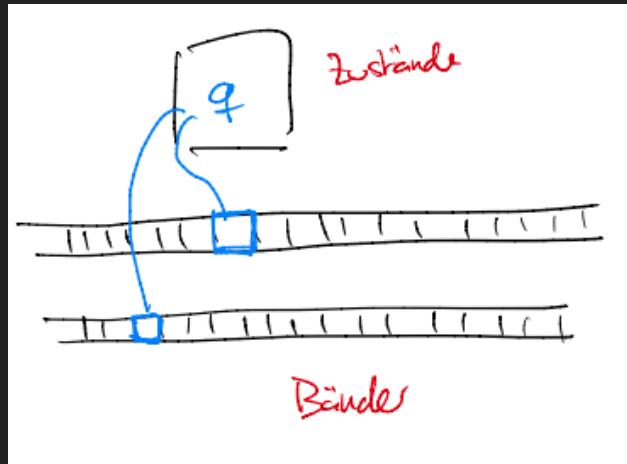


Abbildung 2: Turingmaschine.

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 $\square \in \Gamma / \Sigma$
- $\Delta \subseteq Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times L, S, R^k$ 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 $(q_1, a_1, \dots, a_k, q', a'_1, \dots, a'_k, B_1, \dots, B_k)$ **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

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

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 $C_i \rightarrow_M C_{i+1} \forall i \in \mathbb{N}$. 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

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 ein 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 mindestend **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.

2.8 Definition(entscheidbar)

Eine Sprache L ist genau dann **entscheidbar**, wenn es ein totale k-TM M mit $L(M) = L$ gibt. Wir schreiben **REC** für die Klasse der entscheidbaren Sprachen. Der Begriff entscheidbar für Sprachen ergibt sich hier daraus, dass effektiv entschieden werden kann ob eine gegebene Eingabe in der Sprache liegt oder nicht. Insbesondere steht dies voraus, dass Eingabe, die nicht in der Sprache liegen effektiv als nicht in der Sprache liegend erkannt werden.

Begriff: effektiv \rightsquigarrow eine TM erlebt dies in endlicher Zeit. Da sich der durch TM formatierte Berechenbarkeitsbegriff, also die Formalisierung dessen was effektiv durchführbar ist, auch äquivalent durch rekursive Funktion definieren lässt, werden entscheidbare Sprachen auch als rekursiv bezeichnet.

2.9 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 Alphabett 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.

"Rekursiv aufzählbar ist ein Begriff der verwendet wird um eine Menge zu 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"

Wenn wir sagen, dass eine Sprache "rekursiv aufzählbar ist, bedeutet das, dass es einen Algorithmus oder ein Computerprogramm gibt, das alle Wörter in dieser Sprache auflisten" kann. Es könnte einige Wörter mehrmals auflisten und es könnte eine sehr lange Zeit dauern, aber es würde schließlich jedes Wort in der Sprache "treffen". Eine "k-TM" ist eine Art von Maschine, die wir in der theoretischen Informatik verwenden, um diese Art von Aufzählung zu machen. Wenn es eine k-TM gibt, die eine Sprache akzeptiert, bedeutet das, dass die Sprache rekursiv aufzählbar ist.

2.10 Bemerkung

Jede entscheidbare Sprache ist rekursiv aufzählbar.

2.11 Bemerkung

Alle endlichen Sprachen sind entscheidbar.

2.12 Bemerkung

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

2.13 Definition (Ausgabe)

Sei $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ eine 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 $w \in (\Gamma/\square)^*$.

2.14 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 $M(w_1, \dots, w_n)$.

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

Mittels der Induktivität von \mathbb{N}_0 und $0,1^*$ können so auch partielle Funktionen, die von oder nach \mathbb{N}_0 abilden 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 $bin \circ \Phi \circ bin^{-1}$ partiell berechenbar ist. Gewissermaßen verfügen die hier definierten TM über zwei Ausgabemechanismen. Die Ausgabe im engeren Sinne in Definition 2.13 und das Ablesen von Akzeptanz anhand des schließlich erreichten Zustands in Definition 2.7. Im Sinne der folgenden Bemerkung wäre der zweiten Fall nicht notwendig, allerdings ist dies ein wichtiger Spezialfall.

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

2.17 Bemerkung

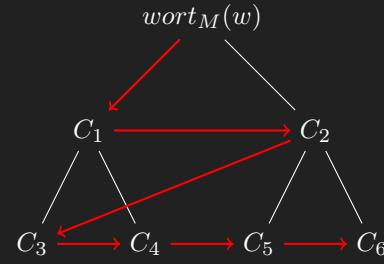
Sei L eine Sprache über einem Alphabet Σ .

- (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.

2.18 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.

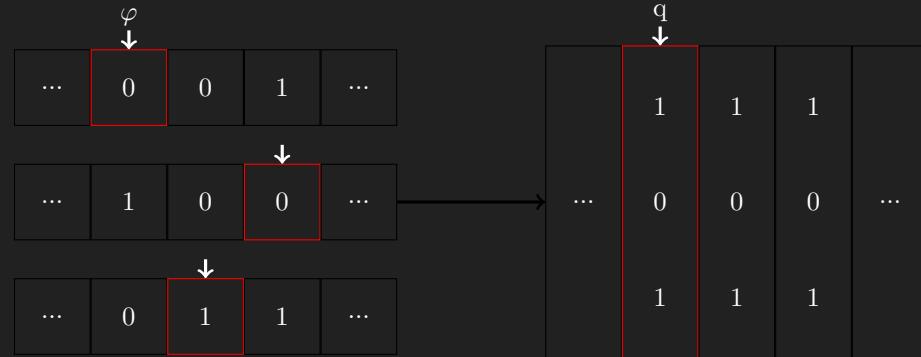
- Von Nichtdeterminismus zu Determinismus: Eine DTM kann die Rechnungen einer nichtdeterministischen TM parallel im Sinne von abwechselnd schrittweise durchführen um schließlich das Verhalten der simulierten TM zu ???. Dies entspricht einer **Breitensuche im Rechnungsbaum**.



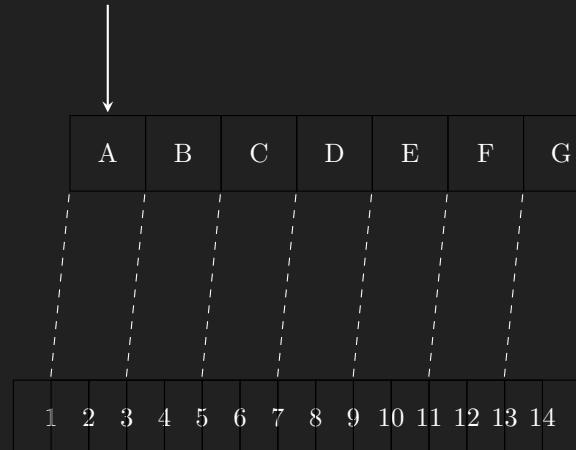
- Von mehreren Bändern zu einem Band: Intuitiv können k Bänder auf ein Band simuliert werden, indem die Felder des einen Bandes in k -teilfelder unterteilt werden, die jeweils die gleiche Bandalphabetbuchstaben wie zufällig als Beschreibung zulassen und es zudem erlaubt zu markieren, dass der simulierte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



- 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.



2.19 Bemerkung

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.

2.20 Church- Turing- These

Berechenbarkeit auf einer Turingmaschine entspricht intuitiver Berechenbarkeit.

3 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

Für $n \in \mathbb{N}$ und eine partiell berechenbare 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

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

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

$$\begin{aligned} (x, y) &\mapsto x^y \\ y &\mapsto 2^y \\ (x_1, \dots, x_m, y_1, \dots, y_n) &\mapsto \varphi(x_1, \dots, x_m, y_1, \dots, y_n) \text{ partiellberechenbar} \\ &\rightsquigarrow (y_1, \dots, y_m) \mapsto \varphi(x_1, \dots, x_m, y_1, \dots, y_n) \text{ partiellberechenbar} \end{aligned}$$

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 vorfährt.

- Zunächst bestimmt S den Code von \mathcal{M}_e
- der Code von \mathcal{M}_e 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, dan den Kopf auf das erste Feld des beschriebenen Bandteils bewegt und dann wie \mathcal{M}_e arbeitet.
- Es wird bestimmt an welcher Stelle der Standardaufzählung der Code von 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

$\mathbb{1}_{L_0}(0)$	$\mathbb{1}_{L_0}(1)$	$\mathbb{1}_{L_0}(2)$	$\mathbb{1}_{L_0}(3)$
$\mathbb{1}_{L_1}(0)$	$\mathbb{1}_{L_1}(1)$	$\mathbb{1}_{L_1}(2)$	$\mathbb{1}_{L_1}(3)$
$\mathbb{1}_{L_2}(0)$	$\mathbb{1}_{L_2}(1)$	$\mathbb{1}_{L_2}(2)$	$\mathbb{1}_{L_2}(3)$

Standardaufzählung

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

□

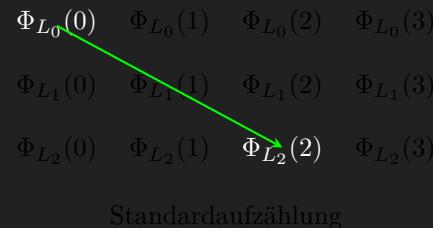
3.8 Definition (diagonales Halteproblem)

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

3.9 Proposition

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



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

3.10 Satz

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

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

Die ist ein Widerspruch. \square

3.11 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

- (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

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?

0111	1	0	0	1
0	01	1	000	011

0111	0	0	0	0	1
0	1	1	1	000	01

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 lösbarer 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 ist 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 lösbarer Instanzen des modifizierten Postschen Korrespondenzproblems über Σ .

Plan:

Für Alphabet mit $|\Sigma| \geq 2$:

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

3.15 Lemma

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 $\varphi : \Gamma^* \rightarrow \Sigma^*$ 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 $\varphi(u'_i) = u_i$ 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

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

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 geeigneten Alphabet Γ transformiert, so dass (p, I) genau dann lösbar ist, wenn I' lösbar ist. \square

Idee:

0	1	0	0	1	0	1	1	1	0	1
0	1	0	0	1	0	1	1	1	0	1

... Betrachte die Homomorphismus von Sprachen $\delta_{\rightarrow}, \delta_{\leftarrow} : \Sigma^* \rightarrow (\Sigma \cup *)^*$ 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

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

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

$$\delta_{\leftarrow}(u_1) \cdots \delta_{\leftarrow}(u_n)* = \delta_{\leftarrow}(u_1 \cdots u_n)*$$

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

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 $p' = (\delta_{\leftarrow}(u_1), \#(wirklichnichttau?) \delta_{\rightarrow}(v_1))$. 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

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 $v_1 \cdots v_n \sqsubset u_1 \cdots u_n$ 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 wie 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 Stopkonfiguration 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

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

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) fürgen wir das Paar $(\hat{q}a, \dagger a)$ hinzu und auch

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

$$\begin{aligned} & (\square\dagger,\dagger), (0\dagger,\dagger), (1\dagger,\dagger) \\ & (\# \dagger \# 0, 0) \end{aligned}$$

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

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

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 apriell berechenbare partielle Funktion mit $dom(\Phi_{\Phi_e(x)}) = \emptyset$. 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

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

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 sogenannte "Fixpunkte" und kein syntaktischen "Fixpunkte". Aus dem Fixpunktsatz kann man leicht das Rekursionstheorem folgen, da 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 Programmereigenschaft unentscheidbar ist.

3.20.4 Korollar

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

4 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 Turingmähen un stark einschränken. Wir betrachten ein Modell, das im wesentlichen ohne Speicher zurechtkommt (=TM ohne Band → 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 lehnen aber fest, dass Symbole nicht überschrieben werden. Die Symbole die 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 Machine enden. Wir soll die Rechnung nicht vor dem Einlesen des ersten \square Symbols enden.

??Die?? bedeutet, dass wir DM $M = (Q, \Sigma \cup \{\square\}, \delta, s, F)$ 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.

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.

er endlicher Automat A ist ein deterministischer endlicher Automat, 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 $M_a = (Q, \sigma, \dots) \rightsquigarrow$ Band spielt keine wesentliche Rolle, Zustände mir 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 $\delta_A : Q \times \Sigma \rightarrow 2^Q$ (=Potenzmenge von Q) mit $\delta_A(q, a) = \{q' \in Q : (q, a, q') \in \Delta\} \forall q \in Q, a \in \Sigma$ **erweiterte Übergangsfunktion** von A ist die Funktion $\delta_A^* : Q \times \Sigma^* \rightarrow 2^Q$ $\delta_A^*(q, \lambda) = \{q\}$ und $\delta_A^*(q, aw) = \bigcup_{q' \in \delta_A(q, a)} \delta_A^*(q', w) \forall q \in Q, a \in \Sigma$ 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 ubtem?? Konfig einer Rechnung von M_A zur Eingabe zu ergeben.

4.3 Bemerkung

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

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

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 $\delta_A(q, a) = \{\delta_{det,A}(q, a)\} \forall q \in Q$ 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

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

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 gegewnigen 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

Sei $A := (\{q_0, q_1\}, \{0, 1\}, \Delta, q_0, \{q_1\})$ mit $\Delta = \{(q_0)\}$ Übergangsdiagramm und übergangstabelle von sehen wie folgt aus:

Zustand/Symbol	0	1
q_0	q_0	q_1
$q_1, *$	q_1	q_0

[Hier muss noch ein Übergangsdiagramm hin!]

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 $P_A = (2^Q, \Sigma, \Delta', \{s\}, \{P \subseteq Q : P \cup F \neq \emptyset\})$ 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 Tutoriumsaufschrieb. (das wird hier in Zukunft angefügt)

4.11 Satz

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:

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

Denn damit folgt

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

Wir zeigen $(*)$ 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 $(*)$. Sei $va := w$ mit $a \in \Sigma$ und $|v| = |w| - 1$.

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

□

5 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 $\leq A^2$, 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) $a \sim b \Rightarrow b \sim a \quad \forall a, b, c \in A$ (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 $A_{/\sim} := [a]_{\sim} : a \in A$ falls diese endlich ist und ∞ andernfalls.

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 $A = (Q, \Sigma, \Delta, s, F)$ eine 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) □

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 Äquivalenzklassen von \sim_A ist. Tatsächlich gilt auch die Umkehrung: Ist L die Vereinigung von Äquivalenzklassen einer Rechtskongruenz \sim mit endlichem Index, so gibt es einen DEA A mit $L(A) = L$

5.6 Definition

Sei Σ eine Alphabet und L Vereinigung von Äquivalenzklassen 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 $w \in \Sigma^+ \dots$ \square

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 5.7 folgt

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

\square

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 \square

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 endlichen 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 unerreichbare Zustände, so gilt $A \cong A_{\sim A, L(A)}$
- (ii) Ist L die Vereinigung von Äquivalenzklassen 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 unerreichbare 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 $w \in \delta'(f(q), a) \Leftrightarrow w \in \delta'([v]_\sim, a) \Leftrightarrow w \sim va \Leftrightarrow \delta^*(s, w) = \delta^*(s, va) \Leftrightarrow \delta^*(s, w) = \delta^*(q, a)$

- (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 A' $\{w \in \Sigma^* : w \in L\}$ und $\sim' := \sim_{A'}$. 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 $u \sim v \Leftrightarrow [u]_\sim = [v]_\sim \Leftrightarrow \delta'(u, v) = \delta'(v, u)$. \square

Satz 5.12 bedeutet insbesondere folgendes: Ist $A_i, i \in \{1, 2\}$ ein DEA ohne unerreichbare 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 unerreichbare 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 (Partition)

Sei A eine Menge. Eine Partition 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 (Verefeinerung)

Seien \mathcal{A}_1 und \mathcal{A}_2 Partitionen einer Menge A . Die Partition \mathcal{A}_2 **Verefeinert** \mathcal{A}_1 (heißt Verefeinerung 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 Partitionen einer Menge A_1 , so dass A_2 die Partition \mathcal{A}_1 vereinfacht.

- (i) $\forall A' \in \mathcal{A}_1$, gibt es eine Teilmengen $\mathcal{A}'_2 \subseteq \mathcal{A}_2$, die eine Partition 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 Partition Σ^*/\sim ist eine Vereinfachung der partition $\Sigma^*/\sim L$.

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 Partition, 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

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 Vereinfachung 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 unerreichbare 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 Vereinfachung von Äquivalenzklassen davon. Damit sind Σ^*/\sim_A und Σ^*/\sim_M nach Proposition 5.18 Vereinfachungen 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 Vereinfachungen von Σ^*/\sim_L sind, folgt mit Bemerkung 5.17 (ii) somit $\Sigma^*/\sim_A = \Sigma^*/\sim_M = \Sigma^*/\sim_L$ und $\sim_A = \sim_L = \sim_M$. □

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 Äquivalenzklassen 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 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 gilt es Wörter $uvw \in \Sigma^*$ 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. \square

5.23 Beispiel

Die Sprache $L = \{0^n 1^n : n \in \mathbb{N}_0\}$ ist nicht regulär. Dies lässt sich mit den 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 $uw = 0^{k-l} 1^k$ \square

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 ien 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 $u \rightarrow_G^n v$ wenn es eine Ableitung von v aus u der Länge n gibt und wir schreiben $u \rightarrow_G^* v$ 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" aus dem 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 $\alpha_1, \dots, \alpha_n$ ein 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 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.

7 Bilder

