

THEORETISCHE INFORMATIK

WINTERSEMESTER 2024/2025

Prof. Klaus Meer

Institut für Informatik, Fakultät 1

Fachgebiet Theoretische Informatik

Brandenburgische Technische Universität Cottbus-Senftenberg

WISSENSKATALOG

ERSTELLER: Ole Matzky

 Ole.Mkzy 

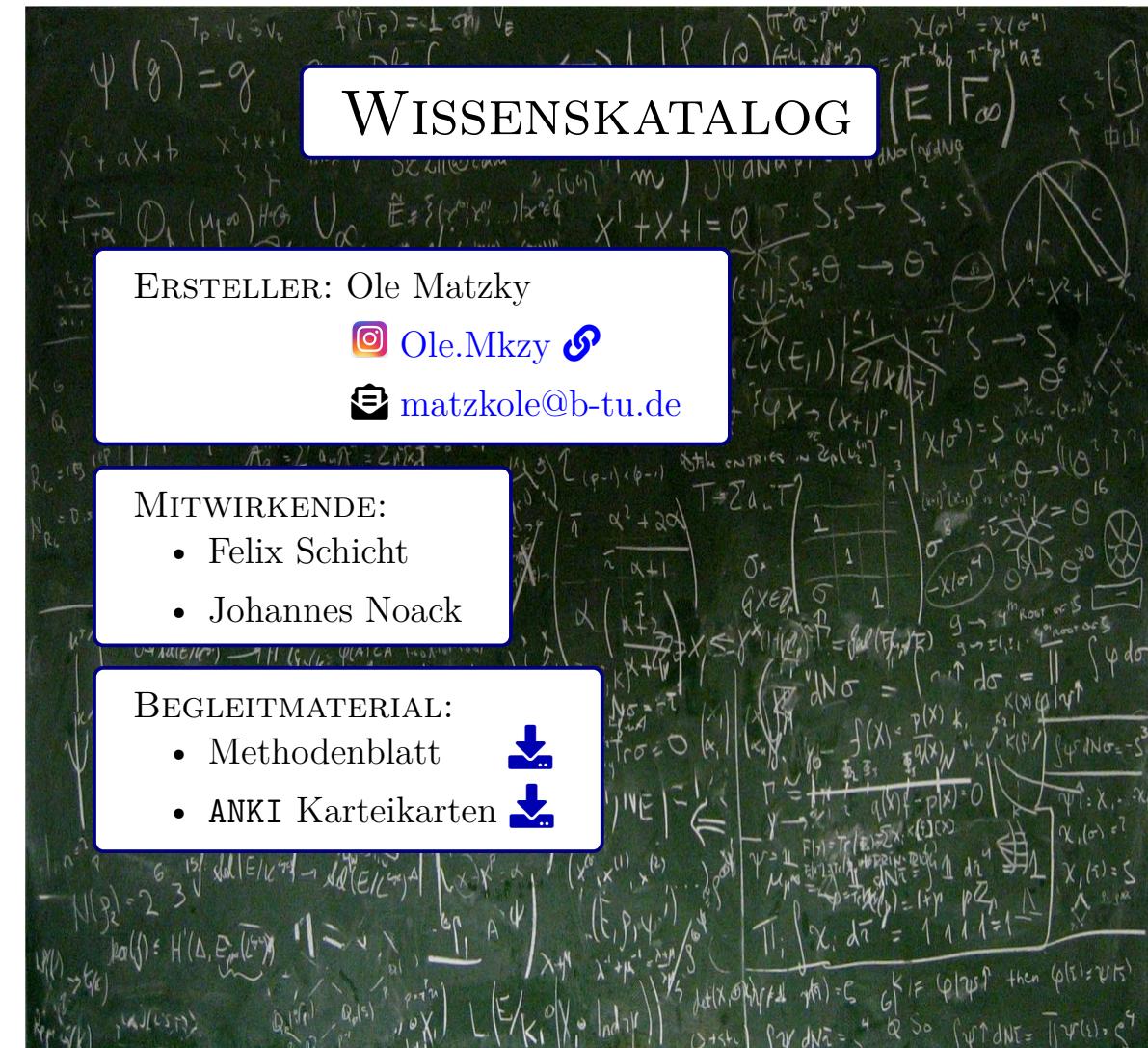
 matzkole@b-tu.de

MITWIRKENDE:

- Felix Schicht
- Johannes Noack

BEGLEITMATERIAL:

- Methodenblatt 
- ANKI Karteikarten 



(Stand: 11. März 2025 | Es wird kein Anspruch auf Vollständigkeit/Korrektheit erhoben

Bei Anmerkungen, Verbesserungen etc. bitte an die Ersteller wenden)

Inhaltsverzeichnis

1 Wiederholung IT-1	1
1.1 Mengen	1
1.1.1 Teilmenge	1
1.1.2 Gleichheit von Mengen	1
1.1.3 leere Menge	1
1.1.4 Potenzmenge	1
1.1.5 Operationen auf Mengen	1
1.1.5.1 Disjunkte Vereinigung	2
1.2 Tupel und Kreuzprodukte	2
1.2.1 Kreuzprodukt (kartesische Produkt)	2
1.2.2 Tupel	2
1.3 Abbildungen	3
1.3.1 Gleichheit von Abbildungen	3
1.3.2 Menge aller Abbildungen	3
1.3.3 Bild und Urbild	3
1.3.4 Eigenschaften von Abbildungen	3
1.3.5 Umkehrabbildung	4
1.3.6 Komposition	4
1.3.7 identische Abbildung	4
1.4 Mächtigkeit	4
1.4.1 Endlich	4
1.4.2 Gleichheitmächtigkeit	4
1.4.2.1 Cantor's erstes Diagonalargument	4
1.4.2.2 Cantorsche Paarungsfunktion	5
1.4.3 Abzählbar unendlich	5
1.4.4 Überabzählbar unendlich	6
1.4.4.1 Cantor's zweites Diagonalargument (\mathbb{R} überabzählbar)	6
1.4.4.2 Satz von Cantor (Potenzmenge)	7
1.4.4.3 Menge aller Abbildungen $\mathbb{N} \rightarrow \mathbb{N}$	8
1.5 Relationen	8
1.5.1 Relation	8
1.5.2 Eigenschaften von Relation	8
1.5.3 Partition	9
1.5.4 Äquivalenzrelation	9
1.5.5 Partielle Ordnung	9
1.5.6 Totale Ordnung	9
2 Wiederholung IT-2	10
2.1 Gruppe	10
2.1.1 Halbgruppe	10
2.1.2 abelsche Gruppe	10
2.1.3 Untergruppe	10
2.2 Lineare Abbildungen für allgemeine Vektorräume	11
2.2.1 Lineare Abbildung	11

3 Sprachen	12
3.1 Allgemeine Begriffe	12
3.1.1 endliches Alphabet Σ	12
3.1.2 Wort	12
3.1.3 leere Wort	12
3.1.4 Funktionen auf Σ	12
3.1.5 Konkatenation	13
3.1.6 Formale Sprache	13
3.1.7 Funktionen auf Sprachen	13
3.2 Längen-lexikografische Ordnung	13
3.3 Reguläre Sprachen	15
3.3.1 Reguläre Ausdrücke	15
3.3.2 Reguläre Sprache	15
3.4 Mächtigkeiten	15
4 Automaten	17
4.1 Deterministische Endliche Automaten	17
4.1.1 DEA	17
4.1.2 Rechnung	17
4.1.3 akzeptierende Sprache	17
4.1.4 Konfiguration	17
4.2 Nicht-Deterministische Endliche Automaten	18
4.2.1 Nicht-Deterministische Übergänge	18
4.2.2 NDEA	18
4.3 NDEA \rightarrow DEA: Potenzautomat	18
4.3.1 Algorithmus Potenzmengenkonstruktion	19
4.4 2 DEA simultan simulieren: Produktautomat	20
4.5 REG \rightarrow NDEA	20
4.5.1 Top-Down	20
4.5.2 Bottom-Up	21
4.6 NDEA \rightarrow REG	21
4.6.1 R(i,j,k)-Konstruktion	21
4.6.2 Umkehrung von Top-Down	21
4.7 Äquivalenz von Automaten	22
4.8 Satz von Rabin & Scott	22
4.9 Abgeschlossenheit der von (N)DEAs akzeptierten Sprachen	24
4.10 Satz von Kleene	25
4.11 Reguläre Pumping-Eigenschaft	26
4.12 Reguläres Pumping-Lemma	26
4.13 Äquivalenzrelationen bezüglich L bzw. M	27
4.14 Satz von Myhill-Nerode	28
4.15 Minimaler Zustandsautomat	29
4.15.1 Algorithmus Zustandsminimierung	30
4.16 Algorithmische Fragen zu DEAs und reg. Sprachen	30
4.16.1 Wortproblem:	30
4.16.2 Wortproblem II:	30
4.16.3 Äquivalenzproblem	30
4.16.4 Äquivalenzproblem II	31
4.16.5 Wortproblem für reguläre Sprachen	31

4.16.6	Endlichkeitsproblem	31
4.16.7	Leerheitsproblem	32
4.17	Homomorphiesatz für reguläre Sprachen	32
4.18	$\tilde{A}K \rightarrow REG$	33
5	Kontextfreie Sprachen	34
5.1	Grammatiken	34
5.2	Kontextfreie Grammatiken	34
5.3	Kellerautomat	35
5.3.1	Operationen	36
5.3.2	Konfiguration und Rechnung	36
5.4	Äquivalenz von PDA und kfr. Grammatik	37
5.5	Chomsky Normalform	40
5.6	Kontextfreies Pumping-Lemma	41
5.7	Abschlusseigenschaften	42
5.8	Das Wortproblem für kfr. Sprachen	43
5.8.1	Komplexität CYK-Algorithmus	44
6	Turingmaschinen	45
6.1	Arbeitsweise der Turingmaschine	45
6.2	Formale Definition	46
6.3	Funktionskomposition	47
6.4	Äquivalenz charakteristische Funktion und Turing- akzeptierbar	48
6.5	Entscheidbarkeit	48
6.6	Rekursiv aufzählbar	49
6.7	Äquivalenz beliebige Grammatik und semi-entscheidbar	51
6.8	Halteproblem	52
6.9	Reduktionen	53
6.10	Satz von Rice	55
6.11	Varianten von TMs	56
6.11.1	k-Band-TMs	56
6.11.2	Nichtdeterministische TMs	57
6.11.3	Linear-beschränkte TMs	57
6.12	Wortproblem für linear-beschränkte TMs	57
6.13	Kontextsensitive Grammatiken	58
6.14	Wortproblem für kontextsensitive Grammatiken	58
6.15	Sprachen $L \notin RE \wedge L \notin co - RE$	58
6.16	Abschlusseigenschaften von RE	58
6.17	Abschlusseigenschaften von REC	59
6.18	Chomsky-Hierarchie	59
7	Komplexitätstheorie	60
7.1	Komplexitätsklasse P	60
7.2	Erfüllbarkeitsproblem	61
7.2.1	SAT	61
7.2.2	k-SAT	62
7.3	Komplexitätsklasse NP	62
7.4	NP-vollständige Probleme, Polynomzeit-Reduktion	64
7.4.1	Beispiele	65
7.4.2	Existenz von NP-vollständigen Problemen	65

7.5	Satz von Cook, Levin	65
7.6	Asymptotisches Wachstumsverhalten	68
7.6.1	Landau-Notation	68
7.6.2	Regel von L'Hospital	69
7.6.3	Stirling-Formel	69
7.6.4	Logarithmen-Gesetze	69
7.6.5	Beispielrechnungen	70
8	Wahr oder Falsch	71
9	Methodenblatt 	72
9.1	Definitionen/Sätze/Lemma	72
9.2	Reguläre Sprachen (<i>REG</i>)	72
9.2.1	Beweise/Widerlege $L \in REG$	72
9.2.2	NDEA \leftrightarrow REG	72
9.2.3	NDEA \rightarrow DEA	73
9.2.4	2 DEA \rightarrow DEA	73
9.2.5	DEA minimieren	73
9.2.6	ÄK bzgl. $\sim_M \rightarrow$ REG	74
9.3	Kontextfreie Sprachen (<i>CFL</i>)	74
9.3.1	Beweise/Widerlege $L \in CFL$	74
9.3.2	CFL \leftrightarrow NDEA	74
9.3.3	CFL \leftrightarrow PDA	75
9.3.4	CFL \rightarrow CNF	75
9.3.5	Prüfe $w \in L(G)$ (CYK-Alg.)	75
9.4	Rekursiv aufzählbare Sprachen (<i>RE</i>)	76
9.5	Entscheidbare Sprachen (<i>REC</i>)	76
10	Anhang	77
10.1	Beispiele	77
10.2	Beweise	104
10.3	„Wahr oder Falsch“-Fragen	106
10.4	„Wahr oder Falsch“-Lösungen	114

Kapitel 1 Wiederholung IT-1

1.1 Mengen

1.1.1 Teilmenge

Definition

Seien A und B Mengen. Dann heißt A eine **Teilmenge** von B , wenn jedes Element von A auch Element von B ist; wir verwenden in diesem Fall die Bezeichnung $A \subseteq B$ und nennen die Relation \subseteq **Inklusion**.

Wenn A Teilmenge von B aber B keine Teilmenge von A ist, so sagt man, dass A eine **echte Teilmenge** von B ist; wir verwenden in diesem Fall die Bezeichnung $A \subsetneq B$ und nennen die Relation \subsetneq eine **echte bzw. strikte Inklusion**.

1.1.2 Gleichheit von Mengen

Definition

Zwei Mengen A und B heißen **gleich**, wenn $A \subseteq B$ und $B \subseteq A$ gilt. Wir schreiben dann $A = B$.

1.1.3 leere Menge

Definition

Die **leere Menge** ist die Menge, die keine Elemente enthält; sie wird als \emptyset bezeichnet.

1.1.4 Potenzmenge

Definition

Sei X eine Menge. Dann ist die **Potenzmenge** von X die Menge aller Teilmengen von X ; für diese Menge benutzen wir die Bezeichnung 2^X . Nach unserer Definition ist 2^X als

$$2^X := \{A : A \subseteq X\}.$$

gegeben.

1.1.5 Operationen auf Mengen

Definition

Seien A, B Mengen. Dann heißt:

- $A \cap B := \{x : (x \in A) \wedge (x \in B)\}$ **Durchschnitt** von A und B ,
- $A \cup B := \{x : (x \in A) \vee (x \in B)\}$ **Vereinigung** von A und B ,
- $A \setminus B := \{x : (x \in A) \wedge (x \notin B)\}$ **Differenz** von A und B ,
- $A \Delta B := (A \setminus B) \cup (B \setminus A)$ **Symmetrische Differenz** von A und B .

- $\bar{A} := M \setminus A$ **Komplement** von A (bei einer festgelegten Grundmenge M).

Beispiel zu 1.1.5 Operationen auf Mengen

1.1.5.1 Disjunkte Vereinigung

Definition

Seien A, B Mengen. A und B heißen genau dann **disjunkt**, wenn $A \cap B = \emptyset$ gilt. In diesem Fall wird die Vereinigung von A und B eine **disjunkte Vereinigung** genannt und als $A \cup B$ bezeichnet.

1.2 Tupel und Kreuzprodukte

1.2.1 Kreuzprodukt (kartesische Produkt)

Definition

Für beliebige Objekte a, b kann man das **(geordnete) Paar** (a, b) definieren. Das Paar (a, b) besteht aus der ersten Komponente a und der zweiten Komponente b . Die Gleichheit $(a, b) = (c, d)$ von Paaren wird durch die Gleichheit $a = c$ und $b = d$ der jeweiligen Komponenten definiert. Für Mengen A, B definiert man das **Kreuzprodukt** (wird auch das **kartesische Produkt** genannt) $A \times B$ als die Menge

$$A \times B := \{(a, b) : a \in A, b \in B\}$$

aller Paare bei denen die erste Komponente in A und die zweite Komponente in B liegt.

1.2.2 Tupel

Definition

Komplett analog zu (geordneten) Paaren definiert man auch (geordnete) Tripel (a, b, c) , (geordnete) Quadrupel (a, b, c, d) und allgemeiner (geordnete) n -Tupel (x_1, \dots, x_n) mit $n \in \mathbb{N}_0$. Dementsprechend betrachtet man auch das Kreuzprodukt $A \times B \times C$ von drei Mengen, das Kreuzprodukt $A \times B \times C \times D$ von vier Mengen und allgemein das Kreuzprodukt

$$X_1 \times \dots \times X_n := \{(x_1, \dots, x_n) : x_1 \in X_1, \dots, x_n \in X_n\}$$

von Mengen X_1, \dots, X_n .

Das Element x_i mit $i \in \{1, \dots, n\}$ im n -Tupel (x_1, \dots, x_n) heißt die i -te **Komponente** des Tupels.

Für eine Menge X führt man die Bezeichnung

$$X^n := \underbrace{X \times \dots \times X}_{n \text{ mal}} = \{(x_1, \dots, x_n) : x_1, \dots, x_n \in X\}$$

ein.

1.3 Abbildungen

Definition

Seien X und Y Mengen. Eine **Abbildung** f von X nach Y ist eine Vorschrift, die jedem $x \in X$ genau ein Element aus Y zuordnet. Das Element aus Y , das einem Element $x \in X$ zugeordnet wird, wird durch $f(x)$ bezeichnet.

Wenn f eine Abbildung von X nach Y ist, dann notieren wir dies durch $f : X \rightarrow Y$. Die Menge X heißt dann der **Definitionsbereich** von f , und die Menge Y der **Wertebereich** von f .

Wenn der Wertebereich Y von f eine Teilmenge von \mathbb{R} ist, so nennen wir $f : X \rightarrow Y$ eine **Funktion**. Bei der Beschreibung von Abbildungen benutzt man auch die Bezeichnung $x \mapsto f(x)$, um zu verdeutlichen, dass x auf $f(x)$ abgebildet wird.

1.3.1 Gleichheit von Abbildungen

Definition

Zwei Abbildungen $f, g : X \rightarrow Y$ heißen gleich, falls $f(x) = g(x)$ für alle $x \in X$ gilt.

Wichtig: Bei Gleichheit von zwei Abbildungen müssen insbesondere Definitionsbereich und Wertebereich gleich sein, nicht nur die Zuweisungsvorschrift.

1.3.2 Menge aller Abbildungen

Definition

Für zwei Mengen X und Y , bezeichnet man als Y^X die Menge aller Abbildungen von X nach Y , also $\{f \mid f : X \rightarrow Y\}$.

Die Mächtigkeit (Kardinalität) von Y^X ist $|Y|^{|X|}$.

1.3.3 Bild und Urbild

Definition

Seien X, Y, A, B Mengen mit $A \subseteq X$ und $B \subseteq Y$ und sei $f : X \rightarrow Y$ eine Abbildung. Dann heißt $f(A) := \{f(x) : x \in A\}$ das **Bild** von A bzgl. f und $f^{-1}(B) := \{x \in X : f(x) \in B\}$ das **Urbild** von B bzgl. f .

1.3.4 Eigenschaften von Abbildungen

Definition

Seien X, Y Mengen und sei $f : X \rightarrow Y$. Dann heißt f

- **injektiv**, falls $\forall x_1, x_2 \in X : x_1 \neq x_2 \implies f(x_1) \neq f(x_2)$ gilt.
- **surjektiv**, falls $\forall y \in Y \exists x \in X : f(x) = y$ gilt.
- **bijektiv**, falls f injektiv und surjektiv ist.

Beispiel zu 1.3.4 Eigenschaften von Abbildungen

1.3.5 Umkehrabbildung

Definition

Seien X, Y Mengen und sei $f : X \rightarrow Y$ bijektiv. Die Abbildung, die jedem $y \in Y$ das eindeutige $x \in X$ mit $f(x) = y$ zuordnet, heißt die **Umkehrabbildung** von f und wird durch $f^{-1} : Y \rightarrow X$ bezeichnet.

1.3.6 Komposition

Definition

Seien X, Y, Z Mengen, $f : X \rightarrow Y$ und $g : Y \rightarrow Z$ Abbildungen. Dann heißt $g \circ f : X \rightarrow Z$ mit $(g \circ f)(x) := g(f(x))$ für alle $x \in X$ die **Komposition** (oder Verkettung) von g und f .

1.3.7 identische Abbildung

Definition

Sei X eine Menge. Dann heißt die Abbildung $\text{id}_X : X \rightarrow X$ mit $\text{id}_X(x) := x$ für alle $x \in X$ die **identische Abbildung** auf X . Man schreibt auch häufig id , wenn die Wahl von X aus dem Kontext klar ist.

1.4 Mächtigkeit

1.4.1 Endlich

Definition

Eine Menge X heißt endlich, falls $X = \emptyset$ oder falls eine bijektive Abbildung von $\{1, \dots, n\}$ nach X existiert mit $n \in \mathbb{N}$. Bei einer endlichen Menge X bezeichnet die Mächtigkeit die Anzahl der Elemente von X . Man notiert die Mächtigkeit von X durch $|X|$ mit $|X| = n$. Man setzt die Kardinalität von \emptyset gleich 0.

1.4.2 Gleichheitmächtigkeit

Definition

Eine Menge A heißt gleichmächtig zu einer Menge B , wenn es eine Bijektion $f : A \rightarrow B$ gibt. Man schreibt dann: $|A| = |B|$

Ist A gleichmächtig zu B , dann gibt es eine Bijektion f zwischen A und B . Da f bijektiv ist, gibt es eine eindeutige Umkehrfunktion von f , die auch eine Bijektion ist, also ist auch B gleichmächtig zu A .

→ Die Gleichheitmächtigkeit ist eine Äquivalenzrelation auf der Menge aller Mengen

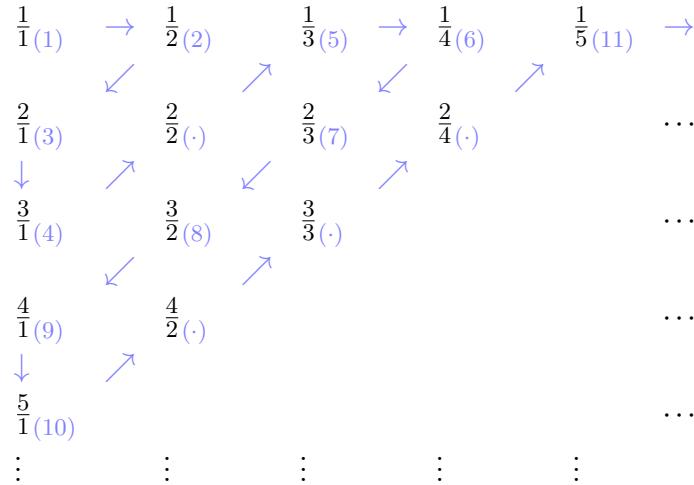
Beispiel zu 1.4.2 Gleichheitmächtigkeit

1.4.2.1 Cantor's erstes Diagonalargument

Cantors erstes Diagonalargument ist ein mathematisches Beweisverfahren, mit dem man gegebenenfalls zeigen kann, dass zwei unendliche Mengen gleichmächtig sind.

Bsp. \mathbb{N} und Menge der rationalen Zahlen \mathbb{Q} sind gleichmächtig.

Dies lässt sich mit *Cantor's Diagonalargument* zeigen, indem man die Menge der positiven rationalen Zahlen \mathbb{Q}^+ als Brüche in einem zweidimensionalen Schema anordnet und in einer „schlangenförmigen“ Reihenfolge *diagonal* abzählt (nicht vollständig gekürzte Brüche werden übersprungen (.)):



Man erhält auf diese Weise eine Bijektion von \mathbb{N} (Abzählung) auf \mathbb{Q}^+ :

$$\begin{array}{ccccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & \dots \\ \downarrow & \downarrow \\ 1 & \frac{1}{2} & 2 & 3 & \frac{1}{3} & \frac{1}{4} & \frac{2}{3} & \frac{3}{2} & 4 & 5 & \frac{1}{5} & \dots \end{array}$$

Um die Gleichmächtigkeit aller rationalen Zahlen und der natürlichen Zahlen zu zeigen, erweitert man diese Abzählung, indem man vor der Eins noch eine Null und hinter jeder Zahl deren Negatives einfügt:

$$\begin{array}{ccccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & \dots \\ \downarrow & \downarrow \\ 0 & 1 & -1 & \frac{1}{2} & -\frac{1}{2} & 2 & -2 & 3 & -3 & \frac{1}{3} & -\frac{1}{3} & \frac{1}{4} & -\frac{1}{4} & \frac{2}{3} & -\frac{2}{3} & \dots \end{array}$$

1.4.2.2 Cantorsche Paarungsfunktion

Die Cantorsche Paarungsfunktion ist eine bijektive totale Funktion $\pi: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, mit der man ein beliebiges Paar (x, y) natürlicher Zahlen durch eine einzige natürliche Zahl n darstellen kann:

$$(x, y) \mapsto \pi(x, y) := y + \frac{1}{2} \cdot (x + y) \cdot (x + y + 1)$$

Mit ihr kann also gezeigt werden, dass $\mathbb{N} \times \mathbb{N}$ und \mathbb{N} gleichmächtig sind.

1.4.3 Abzählbar unendlich

Definition

Eine Menge A wird als abzählbar unendlich bezeichnet, wenn sie die **gleiche Mächtigkeit** hat wie die Menge der natürlichen Zahlen \mathbb{N} . Dies bedeutet, dass es eine Bijektion zwischen A und \mathbb{N} gibt, die Elemente der Menge A also „durchnummerniert“ werden

können. Die Mächtigkeit wird dann wie folgt bezeichnet:

$$|A| = \aleph_0 \text{ (Kardinalzahl Aleph Null)}$$

(die Menge der natürlichen Zahlen \mathbb{N} ist per Definition abzählbar unendlich, da sie dieselbe Mächtigkeit wie sie selbst besitzt.)

Beispiel zu 1.4.3 Abzählbar unendlich

1.4.4 Überabzählbar unendlich

Definition

Eine Menge A heißt überabzählbar, wenn sie nicht abzählbar ist. Eine Menge A ist also genau dann überabzählbar, wenn ihre Mächtigkeit größer ist als die der Menge der natürlichen Zahlen \mathbb{N} . Die Mächtigkeit wird dann wie folgt bezeichnet:

$$|A| = 2^{\aleph_0} = \beth_1 \text{ (Kardinalzahl Beth Eins)}$$

Folgerung

Die **Menge aller Abbildungen** von einer abzählbar unendlichen Menge A nach einer abzählbar unendlichen Menge B ist überabzählbar.

Beweis

Die Kardinalität von B^A ist $|B|^{|A|} = \aleph_0^{\aleph_0} > \aleph_0$

□

Beispiel zu 1.4.4 Überabzählbar unendlich

1.4.4.1 Cantor's zweites Diagonalargument (\mathbb{R} überabzählbar)

Cantors zweites Diagonalargument ist ein mathematischer Beweis dafür, dass die Menge der reellen Zahlen \mathbb{R} überabzählbar ist, und allgemeiner, dass die Abbildungen einer Menge nach $\{0,1\}$ sowie die Potenzmenge einer Menge mächtiger als diese Menge sind (**Satz von Cantor**).

Sei $(z_i)_{i \in \mathbb{N}}$ eine beliebige Folge reeller Zahlen im offenen Intervall $(0, 1)$. Die Zahlen z_i können dabei wie folgt als Dezimalzahl mit den Dezimalstellen a_{ij} dargestellt werden:

$z_1 = 0,$	a_{11}	a_{12}	a_{13}	a_{14}	\dots
$z_2 = 0,$	a_{21}	a_{22}	a_{23}	a_{24}	\dots
$z_3 = 0,$	a_{31}	a_{32}	a_{33}	a_{34}	\dots
\vdots					\ddots

Aus den Diagonalelementen (gelb hervorgehoben) konstruiert man eine neue Zahl

$$x = 0, x_1 x_2 x_3 \dots \text{ mit } x_i = \begin{cases} 4, & a_{ii} = 5 \\ 5, & \text{sonst} \end{cases}.$$

Damit erhält man die sogenannte *Diagonalzahl* x , die der Folge (z_i) zugeordnet ist. Diese unterscheidet sich von allen Zahlen in der Folge in mindestens einer Dezimalstelle und ist

größer als 0 und kleiner als 1.

Jede Folge enthält also nicht die ihr zugeordnete Diagonalzahl, somit enthält keine Folge jede reelle Zahl zwischen 0 und 1. Mit Folgen als Abbildungen $\mathbb{N} \rightarrow (0, 1)$ aufgefasst, gibt es also keine surjektive Abbildung $\mathbb{N} \rightarrow (0, 1)$. Die Menge der reellen Zahlen im Intervall $(0, 1)$ ist deshalb nicht gleichmächtig zu \mathbb{N} . Da $(0, 1)$ noch zusätzlich die Diagonalzahl enthält, gilt $|(\mathbb{0}, 1)| > |\mathbb{N}|$ und $(0, 1)$ ist überabzählbar

1.4.4.2 Satz von Cantor (Potenzmenge)

Definition

Der Satz von Cantor, eine Verallgemeinerung von **Cantors zweitem Diagonalargument**, besagt, dass eine Menge A weniger mächtig als ihre Potenzmenge $\mathcal{P}(A)$ (der Menge aller Teilmengen) ist, dass also $|A| < |\mathcal{P}(A)|$ gilt.

Folgerung

Die Potenzmenge $\mathcal{P}(A)$ einer abzählbar unendlichen Menge A ist überabzählbar. Genauso auch die Menge aller Abbildungen von A nach $\{0, 1\}$.

Beweis

Die Potenzmenge $\mathcal{P}(A)$ einer abzählbar unendlichen Menge A ist die Menge aller Teilmengen von A . Eine Teilmenge von A kann über die charakteristische Funktion $\chi_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases}$ für alle $x \in A$ beschreiben, was wiederum einer Abbildung $A \rightarrow \{0, 1\}$ entspricht. Die Potenzmenge ist also äquivalent zu der Menge aller Abbildungen von A nach $\{0, 1\}$, was man auch bei den Mächtigkeiten erkennt:

$$|\mathcal{P}(A)| = 2^{|A|} = |\{0, 1\}^{|A|}| = |\{0, 1\}^A| = |\{f \mid f : A \rightarrow \{0, 1\}\}|$$

Mittels **Cantors zweitem Diagonalargument** kann man einen Widerspruchsbeweis führen, der beweist, dass die Potenzmenge überabzählbar ist:

Annahme: Menge aller Teilmengen von A wäre abzählbar. Dann lassen sich die Teilmengen als T_1, T_2, T_3, \dots abzählen und in folgendes Schema eintragen:

	x_1	x_2	x_3	x_4	\dots
T_1	1	0	0	1	\dots
T_2	0	0	1	1	\dots
T_3	1	0	1	0	\dots
⋮				⋮	

mit $T_i \subseteq A$ und $x_i \in A$. Aus den Diagonalelementen (gelb hervorgehoben) konstruiert man eine neue Teilmenge T_d , die *Diagonalteilmenge*. Das Komplement $\overline{T_d}$ der Diagonalteilmenge ist nun eine Teilmenge, die in diesem Schema nicht vorkommen kann, denn $\overline{T_d}$ kann keines der T_i sein, weil $\overline{T_d}$ definitionsgemäß in der i-ten Spalte des Schemas genau dann eine 0 hat, wenn T_d dort eine 1 hat, und umgekehrt. Mit $\overline{T_d}$ hat man also eine Teilmenge gefunden, die man nicht durch das Abzählen mit T_i darstellen kann, also ist die Potenzmenge überabzählbar. \square

1.4.4.3 Menge aller Abbildungen $\mathbb{N} \rightarrow \mathbb{N}$

Die Menge aller Abbildung von \mathbb{N} nach \mathbb{N} , $\mathbb{N}^{\mathbb{N}} := \{f \mid f : \mathbb{N} \rightarrow \mathbb{N}\}$ ist überabzählbar unendlich, da $|\mathbb{N}^{\mathbb{N}}| = |\mathbb{N}|^{|\mathbb{N}|} = \aleph_0^{\aleph_0} = 2^{\aleph_0} = \beth_1$.

Beweis

Um zu zeigen, dass $\mathbb{N}^{\mathbb{N}}$ überabzählbar ist, nutzen wir ein Argument ähnlich **Cantors zweites Diagonalargument**. Angenommen, $\mathbb{N}^{\mathbb{N}}$ wäre abzählbar. Dann könnten wir alle Funktionen $f \in \mathbb{N}^{\mathbb{N}}$ in eine Liste schreiben:

$$f_1, f_2, f_3, \dots$$

wobei jede Funktion f_i durch ihre Wert an den natürlichen Zahlen $0, 1, 2, \dots$ beschrieben wird:

$$f_i = (f_i(0), f_i(1), f_i(2), \dots)$$

Wir konstruieren nun eine „diagonale“ Funktion g mit $g(i) = f_i(i) + 1 \quad \forall i \in \mathbb{N}$. Damit unterscheidet sich g von jeder Funktion f_i an der i -ten Stelle, da $g(i) \neq f_i(i)$. Somit kann g nicht in der Liste der f_i enthalten sein. Also ist die Menge aller Abbildungen von \mathbb{N} nach \mathbb{N} nicht abzählbar unendlich! \square

1.5 Relationen

1.5.1 Relation

Definition

Seien X, Y Mengen. Dann heißt eine Teilmenge R von $X \times Y$ eine (binäre) **Relation** zwischen X und Y . Bei $X = Y$, heißt R eine (binäre) Relation auf X .

Ein Paar (x, y) ist Teil der Relation R :

$$(x, y) \in R \text{ oder } xRy$$

1.5.2 Eigenschaften von Relation

Für eine Relation R auf der Menge X sind folgende Eigenschaften definiert:

- reflexiv: $\forall a \in X : (a, a) \in R$
- irreflexiv: $\forall a \in X : (a, a) \notin R$
- symmetrisch: $\forall a, b \in X : (a, b) \in R \implies (b, a) \in R$
- antisymmetrisch: $\forall a, b \in X : (a, b) \in R \wedge (b, a) \in R \implies a = b$
- asymmetrisch: $\neg \exists a, b \in X : (a, b) \in R \implies (b, a) \in R$
- transitiv: $\forall a, b, c \in X : (a, b) \in R \wedge (b, c) \in R \implies (a, c) \in R$
- total: $\forall a, b \in X : (a, b) \in R \vee (b, a) \in R$

Bemerkung

reflexiv und irreflexiv, sowie symmetrisch und antisymmetrisch sind nicht das Gegenteil von einander, sondern zwei gegenüberliegende Pole (alle oder keine Paare erfüllen die Eigenschaft, häufig erfüllt nur ein Teil die Eigenschaft, und der andere Teil sie nicht)

1.5.3 Partition**Definition**

Eine Partition einer Menge ist eine Zerlegung dieser Menge in nichtleere paarweise disjunkte Teilmengen.

1.5.4 Äquivalenzrelation**Definition**

Sei \sim eine Relation auf der Menge X . Dann heißt \sim **Äquivalenzrelation**, falls \sim reflexiv, symmetrisch und transitiv ist (siehe **Eigenschaften von Relation**)

Für eine Äquivalenzrelation \sim auf der Menge X und ein $x \in X$ heißt

$$[x]_{\sim} := \{y \in X : x \sim y\}$$

die **Äquivalenzklasse** von x bzgl. \sim .

Die Menge aller Äquivalenzklassen von \sim ist

$$X/\sim := \{[x]_{\sim} : x \in X\}.$$

und die Menge aller Äquivalenzklassen entspricht einer **Partitionierung** der Menge X

1.5.5 Partielle Ordnung**Definition**

Eine binäre Relation \preceq auf einer Menge X heißt **partielle Ordnung** auf X , wenn \preceq reflexiv, antisymmetrisch und transitiv ist.

Eine Menge X mit partiellen Ordnung \preceq darauf heißt **Poset** (partiell geordnete Menge; engl. partially ordered set), Schreibweise:

$$(X, \preceq)$$

1.5.6 Totale Ordnung**Definition**

Wenn für ein Poset (X, \preceq) für alle Elemente von X die Totalität erfüllt ist, so nennt man (X, \preceq) eine **total geordnete Menge** und \preceq eine **totale Ordnung** auf X .

Kapitel 2 Wiederholung IT-2

2.1 Gruppe

Definition

Sei G Menge und $* : G \times G \rightarrow G$. Dann heißt $(G, *)$ **Gruppe**, falls für $*$ gilt:

- **Abgeschlossenheit:** $a * b \in G \quad \forall a, b \in G$
- **Assoziativität:** $a * (b * c) = (a * b) * c \quad \forall a, b, c \in G$
- **neutrales Element** (eindeutig!) $e \in G$: $e * a = a * e = a \quad \forall a \in G$
- **inverses Element** (eindeutig!) $a^{-1} \in G$: $a^{-1} * a = a * a^{-1} = e \quad \forall a \in G$

Eine Gruppe erfüllt die **Sudoku**-Eigenschaften, d.h. $a, b, c \in G \Rightarrow a * b \neq a * c$.

Mit anderen Worten: In der Verknüpfungstabelle taucht in einer Spalte oder Zeile kein Element mehrmals auf!

2.1.1 Halbgruppe

Definition

Sei G Menge und $* : G \times G \rightarrow G$. Dann heißt $(G, *)$ **Halbgruppe**, falls für $*$ gilt:

- **Abgeschlossenheit:** $a * b \in G \quad \forall a, b \in G$
- **Assoziativität:** $a * (b * c) = (a * b) * c \quad \forall a, b, c \in G$

Die Halbgruppe ist eine Verallgemeinerung der **Gruppe**. Für sie muss es nicht zwingend ein neutrales Element geben.

2.1.2 abelsche Gruppe

Definition

Sei G Menge und $* : G \times G \rightarrow G$. Dann heißt $(G, *)$ **abelsche Gruppe**, falls für $*$ gilt:

- $(G, *)$ ist eine **Gruppe**
- **Kommutativität:** $a * b = b * a \quad \forall a, b \in G$

Abelsche Gruppen heißen auch **kommulative Gruppen**.

Bsp.: $(\mathbb{Z}, +), (\mathbb{Q}, +), (\mathbb{R}, +), (\mathbb{Q} \setminus \{0\}, \cdot), (\mathbb{R} \setminus \{0\}, \cdot)$

2.1.3 Untergruppe

Definition

Sei (G, \cdot) **Gruppe** und $\emptyset \neq H \subseteq G$. Dann heißt H Untergruppe von (G, \cdot) , falls für alle $a, b \in H$ gilt:

- **Abgeschlossenheit:** $a \cdot b \in H$

- **inverses Element** (eindeutig!) $\mathbf{a}^{-1} \in G$
 H ist bezüglich der Verknüpfung \cdot selber wieder eine **Gruppe**.

2.2 Lineare Abbildungen für allgemeine Vektorräume

2.2.1 Lineare Abbildung

Definition

Seien V und W Vektorräume über \mathbb{K} und sei $F : V \rightarrow W$. Die Abbildung F heißt linear, falls:

$$(L1) \quad F(v + w) = F(v) + F(w) \quad \forall v, w \in V$$

$$(L2) \quad F(\lambda v) = \lambda F(v) \quad \forall v \in V, \forall \lambda \in \mathbb{K}$$

$$(L) \quad F(\lambda v + \mu w) = \lambda F(v) + \mu F(w) \quad \forall v, w \in V, \forall \lambda, \mu \in \mathbb{K} \quad (\text{L1 und L2 zusammengefasst})$$

Kapitel 3 Sprachen

3.1 Allgemeine Begriffe

3.1.1 endliches Alphabet Σ

Definition

Ein endliches Alphabet ist eine endliche Menge Σ

$$\Sigma := \{a_1, \dots, a_s\} \quad (a_i \text{ Buchstabe, nicht zerteilbar})$$

3.1.2 Wort

Definition

Ein Wort $w := \{w_1, \dots, w_n\}$ ist ein geordnetes Tupel aus Buchstaben $w_i \in \Sigma$, $|w| := n$ ist die Länge von w

3.1.3 leere Wort

Definition

Das **leere Wort** λ ist ein spezielles Wort mit $|\lambda| := 0$

3.1.4 Funktionen auf Σ

Definition

$$\Sigma^n := \underbrace{\Sigma \times \Sigma \times \dots \times \Sigma}_{n\text{-fache kartesische Produkt}}$$

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

$$\begin{aligned}\Sigma^0 &:= \{\lambda\} \\ \Sigma^* &:= \bigcup_{i=0}^{\infty} \Sigma^i \implies \lambda \in \Sigma^* \\ \Sigma^+ &:= \Sigma^* \setminus \{\lambda\}\end{aligned}$$

Σ^* ist die Menge aller Wörter über Σ

3.1.5 Konkatenation

Definition

Für $x = (x_1, \dots, x_n)$, $y = (y_1, \dots, y_m)$ aus Σ^* ist eine Konkatenation definiert als

$$x.y := (x_1, \dots, x_n, y_1, \dots, y_m) = (x_1.x_2 \dots y_1.y_2 \dots y_m)$$

$$|x.y| = |x| + |y|$$

$$\begin{array}{l} x.y \rightarrow x \text{ ist Präfix} \\ \quad \quad \quad \rightarrow y \text{ ist Suffix} \end{array}$$

λ ist das neutrale Element der Konkatenation: $x.\lambda = \lambda.x = x \quad (\forall x \in \Sigma^*)$

3.1.6 Formale Sprache

Definition

Ein $L \subseteq \Sigma^*$ heißt **formale Sprache**

3.1.7 Funktionen auf Sprachen

Definition

Sei Σ ein endliches Alphabet und $L, L_1, L_2 \subseteq \Sigma^*$:

- $\bar{L} := \Sigma^* \setminus L$ (Komplement von L bezüglich Σ^*)
- $L_1 \cup L_2 := \{w \in \Sigma^* \mid w \in L_1 \vee w \in L_2\}$
- $L_1 \cap L_2 := \{w \in \Sigma^* \mid w \in L_1 \wedge w \in L_2\}$
- $L_1.L_2 := \{w_1.w_2 \in \Sigma^* \mid w_1 \in L_1 \wedge w_2 \in L_2\}$
- $L^* := \{(w_1, \dots, w_t) \mid t \in \mathbb{N}, w_i \in L \vee w_i = \lambda \ \forall 1 \leq i \leq t\}$
 $\lambda \in L^*$ mit $t = 1, w_1 = \lambda$
 L^* heißt Kleene-Stern von L
- $L^+ := L.L^* \implies \lambda \in L^+ \Leftrightarrow \lambda \in L$
- $L^n := \underbrace{L.L \dots L}_{n\text{-fach Konkatenation}}$

$$(L^* = \bigcup_{i=0}^{\infty} L^i)$$

Beispiel zu 3.1.7 Funktionen auf Sprachen

3.2 Längen-lexikografische Ordnung

Definition

Die **längen-lexikografische Ordnung** ist eine Methode, Wörter einer Sprache $L \subseteq \Sigma^*$ systematisch zu ordnen. Sie ordnet die Wörter in zwei Schritten:

1. Zuerst werden die Wörter nach ihrer Länge sortiert (kürzere Wörter stehen vor längeren).

2. Wörter gleicher Länge werden lexikografisch entsprechend der Reihenfolge der Zeichen in Σ sortiert (analog zur Wörterbuchreihenfolge).

Beispiel zu 3.2 Längen-lexikografische Ordnung

Bijektion zwischen \mathbb{N} und Σ^*

Die längen-lexikografische Ordnung definiert eine Bijektion zwischen der Menge der natürlichen Zahlen \mathbb{N} und der Menge Σ^* :

- Jedes Wort $w \in \Sigma^*$ hat eine eindeutige Position $n \in \mathbb{N}$ in der Ordnung.
- Umgekehrt kann jede natürliche Zahl $n \in \mathbb{N}$ eindeutig einem Wort $w \in \Sigma^*$ zugeordnet werden.

Damit gilt sowohl $f : \mathbb{N} \rightarrow \Sigma^*$ als auch $f^{-1} : \Sigma^* \rightarrow \mathbb{N}$.

Berechenbarkeit der Abbildung

Die Abbildung $f : \mathbb{N} \rightarrow \Sigma^*$ ist berechenbar, da:

1. Die Länge des n -ten Wortes kann durch einfache arithmetische Berechnungen ermittelt werden, da die Anzahl der Wörter mit einer festen Länge k durch $|\Sigma|^k$ gegeben ist.
2. Die Position innerhalb einer festen Länge wird durch eine Darstellung der Position als Zahl im Basis- $|\Sigma|$ -System berechnet, wobei jede Ziffer einem Zeichen in Σ entspricht.

Da auch die Umkehrfunktion $f^{-1} : \Sigma^* \rightarrow \mathbb{N}$ durch die gleiche Methode (Ermittlung der Länge und Position) berechenbar ist, handelt es sich um eine bijektive, berechenbare Abbildung.

Abzählbarkeit komplexer Objekte

Mittels der längen-lexikografischen Ordnung können auch komplexere Objekte wie endliche Automaten, reguläre Ausdrücke, Grammatiken und Turingmaschinen systematisch „nummeriert“ werden.

Dies funktioniert, weil solche Objekte durch endliche Beschreibungen darstellbar sind:

- DEA: endliche Beschreibung seiner Zustandsmenge, Übergangsfunktion und akzeptierenden Zustände
- regulärer Ausdruck: endliche Zeichenkette
- Grammatiken: endliche Beschreibung der NTS und Regeln
- Turingmaschinen: endliche Beschreibung seiner Zustandsmenge, Übergangsfunktion und akzeptierenden Zustände

Durch die Kodierung solcher Objekte als Wörter über einem Alphabet Σ (z. B. in einer Notation wie JSON oder binärer Darstellung) und die Anwendung der längen-lexikografischen Ordnung auf diese Kodierungen kann eine Bijektion auf \mathbb{N} gefunden werden, womit Menge aller Objekte einer solchen Klasse jeweils **abzählbar unendlich** ist.

3.3 Reguläre Sprachen

3.3.1 Reguläre Ausdrücke

Definition

Sei Σ ein endliches Alphabet. Die Menge der regulären Ausdrücke über Σ und der durch sie dargestellten Sprachen ist wie folgt definiert:

- \emptyset , λ , und jedes $a \in \Sigma$ sind reguläre Grundausdrücke. Die zugehörigen Sprachen sind:

$$\begin{aligned}\mathcal{L}(\emptyset) &:= \emptyset \\ \mathcal{L}(\lambda) &:= \{\lambda\} \\ \mathcal{L}(a) &:= \{a\}\end{aligned}$$

- Seien x, y reguläre Ausdrücke, dann sind auch die folgende Ausdrücke regulär:
 - $x.y$ mit zugehörigen Sprache $\mathcal{L}(x.y) = \mathcal{L}(x).\mathcal{L}(y)$
 - $x \cup y$ (oder $\{x, y\}$) mit Sprache $\mathcal{L}(x \cup y) = \mathcal{L}(x) \cup \mathcal{L}(y)$
 - x^* mit Sprache $\mathcal{L}(x^*) := (\mathcal{L}(x))^*$

Alles, was sich auf dieser Art in **endlich vielen** Schritten aus den regulären Grundausdrücken erzeugen lässt, ist ein regulärer Ausdruck, und nichts sonst.

Beispiel zu 3.3.1 Reguläre Ausdrücke

3.3.2 Reguläre Sprache

Definition

Ein $L \subseteq \Sigma^*$ heißt reguläre Sprache, wenn es einen regulären Ausdruck x gibt mit $L = \mathcal{L}(x)$.

Die Darstellung einer Sprache durch einen regulären Ausdruck ist allgemein nicht eindeutig.

3.4 Mächtigkeiten

- die Menge aller Wörter Σ^* ist abzählbar unendlich

Beweis zu Σ^* ist abzählbar unendlich

- die Menge aller Sprachen ist überabzählbar unendlich

Beweis zu Menge aller Sprachen ist überabzählbar unendlich

- die Menge der regulären Sprachen ist abzählbar unendlich

Beweis zu Menge der regulären Sprachen ist abzählbar unendlich

Folgerung

→ Es gibt nicht-reguläre Sprachen, denn die Menge aller Sprachen ist überabzählbar unendlich, aber die Menge der regulären Sprachen ist lediglich abzählbar unendlich.

Kapitel 4 Automaten

4.1 Deterministische Endliche Automaten

4.1.1 DEA

Definition

Ein deterministischer endlicher Automat **DEA** ist ein 5-Tupel $M = (K, \Sigma, \delta, s, F)$ mit

- K endliche Menge von Zuständen
- Σ endliches Alphabet
- $s \in K$ Startzustand
- $F \subseteq K$ Menge der akzeptierenden Endzustände
- $\delta : K \times \Sigma \rightarrow K$ Übergangsfunktion

4.1.2 Rechnung

Definition

Der DEA M führt wie folgt eine Rechnung auf dem Wort $w := w_1, \dots, w_n \in \Sigma^*$ aus:
 M beginnt in s , liest w_1 und wechselt in den Zustand $s_1 = \delta(s, w_1)$. Dann liest M w_2 und wechselt zu $s_2 = \delta(s_1, w_2)$ usw. bis $s_n = \delta(s_{n-1}, w_n)$. Hier hält M und **akzeptiert** w genau dann, wenn $s_n \in F$.

In jeder Situation ist der nächste Rechenschritt von M eindeutig festgelegt → Determinismus

Beispiel zu 4.1.2 Rechnung

4.1.3 akzeptierende Sprache

Definition

Die von M akzeptierende/erkannte Sprache ist:

$$L(M) = \{x \in \Sigma^* \mid M \text{ beendet Rechnung auf } x \text{ in } F\}$$

4.1.4 Konfiguration

Definition

Eine **Konfiguration** von M ist ein Element in $K \times \Sigma^*$, interpretiert als der aktuelle Zustand und der noch zu lesende Teil der Eingabe. Für $(q_1, w), (q_2, v)$ zwei Konfigurationen sagen wir (q_2, v) ist (unmittelbare) Nachfolgekonfiguration von (q_1, w) , wenn sie in (einen)/mehreren Schritten aus (q_1, w) entsteht. Notation:

$$(q_1, w) \xrightarrow{M} (q_2, v) \text{ ein Schritt}$$

$$(q_1, w) \xrightarrow{M}^* (q_2, v) \text{ endliche viele Schritt}$$

4.2 Nicht-Deterministische Endliche Automaten

4.2.1 Nicht-Deterministische Übergänge

Der Nicht-Determinismus wird durch nicht-deterministische Übergänge hervorgerufen:

- λ -Übergänge (der Automat liest das leere Wort (nichts) und rät dann den nächsten Zustand)
- mehrere mögliche Übergänge (der Automat liest ein Wort und hat mehrere Folgezustände zur Auswahl)

→ für eine Eingabe kann es also mehrere, verschiedene Rechnungen geben (\rightsquigarrow Nicht-Determinismus). Diese müssen nicht zwingend alle in F landen, M akzeptiert, wenn mindestens eine Rechnung in F endet.

Beispiel zu 4.2.1 Nicht-Deterministische Übergänge

4.2.2 NDEA

Definition

Ein nicht-deterministischer endlicher Automat **NDEA** ist ein 5-Tupel $M = (K, \Sigma, \Delta, s, F)$ mit K, Σ, s, F wie beim DEA und $\Delta \subseteq K \times (\Sigma \cup \{\lambda\}) \times K$, die Übergangsrelation.

(d.h. bei Zustand q und gelesenen $a \in \Sigma \cup \{\lambda\}$ kann M potenziell mehrere Folgezustände anlaufen)

Notation für Übergänge: $p \in \Delta(q, a)$, $(q, a, p) \in \Delta$, $((q, a) \xrightarrow{\Delta} p)$

Konfiguration: wie bei DEAs

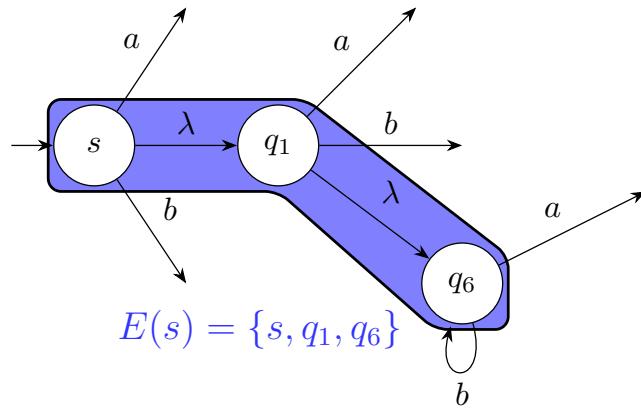
4.3 NDEA → DEA: Potenzautomat

Definition

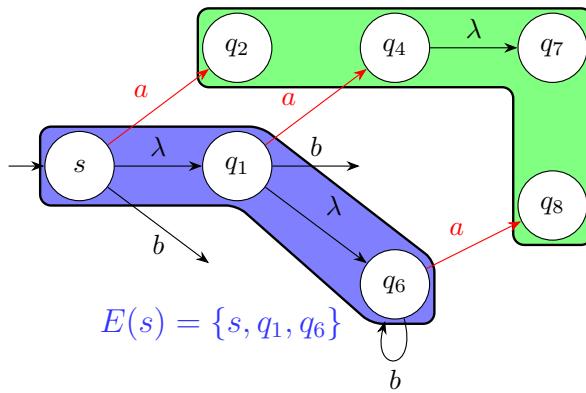
Sei $M = (K, \Sigma, \Delta, s, F)$ ein NDEA. Zu $q \in K$ definiert man den Abschluss $E(q) := \{p \in K \mid (q, \lambda) \xrightarrow{M}^* (p, \lambda)\} \cup \{q\}$ (Alle von q mit λ -Übergänge erreichbaren Zustände)

Definition des DEAs (Potenzautomat) $M' = (K', \Sigma, \delta', s', F')$ mit

- $K' = \mathcal{P}(K)$ (Potenzmenge)
- $s' = E(s) \in K'$
- $F' = \{Q \subset K \mid Q \cap F \neq \emptyset\}$
- $\delta'(Q, a) := \bigcup_{q \in Q} \{E(p) \mid p \in K \wedge (q, a, p) \in \Delta_M\}$



$$\delta'(Q, a) = \{q_2, q_4, q_7, q_8\} \quad (Q = E(s))$$



4.3.1 Algorithmus Potenzmengenkonstruktion

Mittels der **Potenzmengenkonstruktion** kann man aus dem nicht-deterministischen Automat $\mathcal{NA} = \{K, \Sigma, \Delta, s, F\}$ einen äquivalenten, (endlichen, vollständigen,) deterministischen Automaten $\mathcal{A} = \{K', \Sigma, \delta, s', F'\}$ folgendermaßen konstruieren:

Algorithmus

1. Starte mit leeren Zustandsmengen K' und F'
2. Wähle den Startzustand s' von \mathcal{A} als einelementige Menge $s' = \{s\}$ des Startzustandes $s \in K$ von \mathcal{NA} . Füge s' zur Menge der Zustände K'
3. Für alle Zustände q' , die bereits in K' enthalten sind:
 - Für jedes Eingabezeichen $a \in \Sigma \cup \{\lambda\}$:
 - Konstruiere einen Folgezustand q'' als Menge aller Zustände, die \mathcal{NA} ausgehend von einem Zustand aus q' unter Eingabe von a erreichen kann.
 - Also $q'' = \bigcup_{r \in q'} \{t \mid (r, a, t) \in \Delta\}$
 - Füge den Zustand q'' zu K' hinzu, falls er noch nicht in der Menge der Zustände von \mathcal{A} enthalten ist.
 - Ergänze die Übergangsfunktion δ um den Übergang $\delta(q', a) = q''$.

Für jedes Eingabezeichen $a \in \Sigma \cup \{\lambda\}$:

- Konstruiere einen Folgezustand q'' als Menge aller Zustände, die \mathcal{NA} ausgehend von einem Zustand aus q' unter Eingabe von a erreichen kann.

$$\text{Also } q'' = \bigcup_{r \in q'} \{t \mid (r, a, t) \in \Delta\}$$

- Füge den Zustand q'' zu K' hinzu, falls er noch nicht in der Menge der Zustände von \mathcal{A} enthalten ist.
- Ergänze die Übergangsfunktion δ um den Übergang $\delta(q', a) = q''$.

4. Wiederhole Schritt 3. so lange, bis sich K' und δ nicht mehr ändern.
5. Wähle die Menge der akzeptierenden Zustände F' von \mathcal{A} als diejenige Teilmenge von K' , deren Zustände einen akzeptierenden Zustand aus F enthalten.

Beispiel zu NDEA zu DEA

4.4 2 DEA simultan simulieren: Produktautomat

Definition

Seien $M_1 = (K_1, \Sigma, \delta_1, s_1, F_1)$ ein DEA, der die Sprache L_1 akzeptiert und $M_2 = (K_2, \Sigma, \delta_2, s_2, F_2)$ ein DEA, der die Sprache L_2 akzeptiert.

Definition des DEAs (Produktautomat) $M = (K, \Sigma, \delta, s, F)$ mit

- $K = K_1 \times K_2$
- $s = (s_1, s_2)$
- $F = \begin{cases} F_1 \times F_2 & \text{für } L_1 \cap L_2 \\ K_1 \times F_2 \cup K_2 \times F_1 & \text{für } L_1 \cup L_2 \end{cases}$
- $\delta : ((q_1, q_2), a) \rightarrow (\delta_1(q_1, a), \delta_2(q_2, a))$

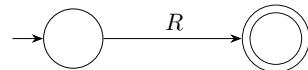
Beispiel zu 4.4 2 DEA simultan simulieren: Produktautomat

4.5 REG → NDEA

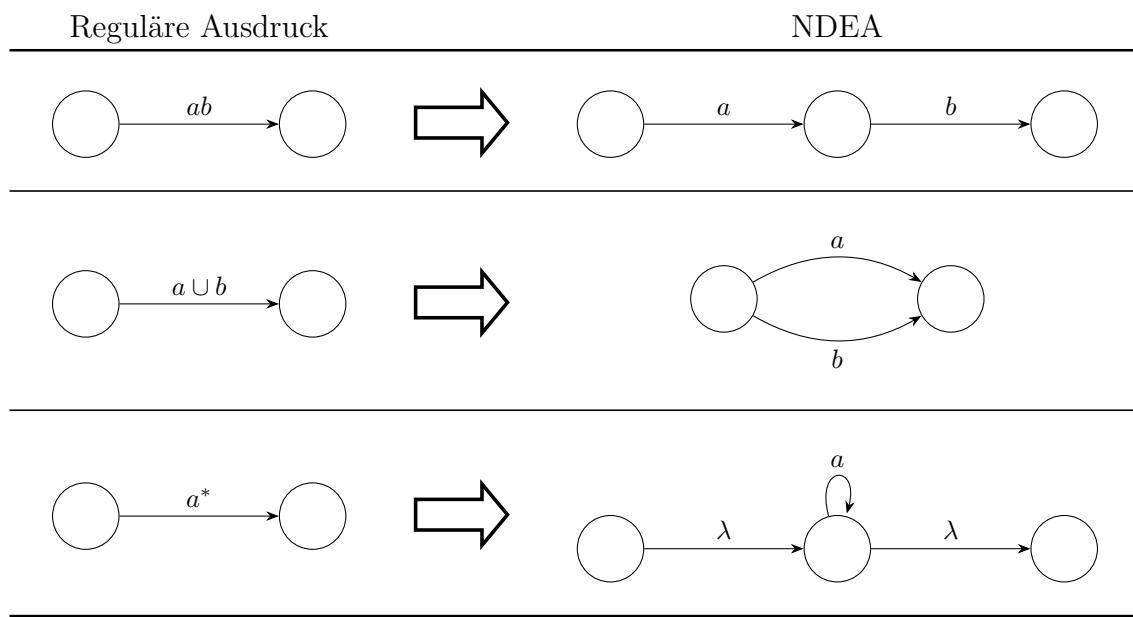
4.5.1 Top-Down

Beim *Top-Down*-Verfahren startet man mit dem kompletten regulären Ausdruck R und zerteilt ihn schrittweise in Übergänge in einem NDEA.

Dafür startet man mit einem Automaten, der lediglich einen Start- sowie Endzustand besitzt, und einen Zustandsübergang vom Start zum Ende, beschriftet mit dem regulären Ausdruck R :



Danach wird der reguläre Ausdruck nach folgenden Regeln iterativ zerlegt:



Beispiel zu 4.5.1 Top-Down

4.5.2 Bottom-Up

siehe <https://hwlang.de/theor/regulaerer-ausdruck-zu-automat-bottomup.htm>

4.6 NDEA → REG

4.6.1 R(i,j,k)-Konstruktion

Zu $i, j \in \{1, \dots, n\}$, $k \in \{0, \dots, n\}$ definiere folgende Sprachen/Mengen

$$R(i, j, k) := \left\{ v \in \Sigma^* \mid \begin{array}{l} \text{startend in } q_i \text{ soll } M \text{ beim Lesen} \\ \text{von } v \text{ nach } q_j \text{ laufen; dazwischen werden nur} \\ \text{Zustände mit Index } \leq k \text{ durchlaufen} \end{array} \right\}$$

Speziell für $k = 0$:

$$R(i, j, 0) := \{a \in \Sigma \mid \delta(q_i, a) = q_j\}$$

Beachte:

$$R(i, j, n) := \{v \in \Sigma^* \mid (q_i, v) \xrightarrow[M]{*} (q_j, \lambda)\}$$

$$L(M) = \bigcup_{q_j \in F} R(1, j, n)$$

$$R(i, j, k) = \underbrace{R(i, j, k-1)}_{\substack{\text{Rechnung läuft} \\ \text{nicht durch } q_k}} \cup \underbrace{R(i, k, k-1)}_{\substack{\text{Rechnung bis} \\ \text{zum ersten} \\ \text{Besuch von } q_k}} \cdot \underbrace{R(k, k, k-1)^*}_{\substack{\text{beliebig oft} \\ \text{über } q_k}} \cdot \underbrace{R(k, j, k-1)}_{\substack{\text{Rechnung von} \\ \text{ } q_k \text{ nach } q_j}}$$

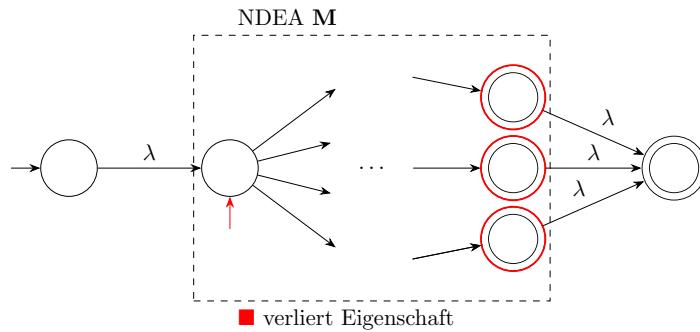
(siehe **Satz von Kleene** zweiter Beweis)

4.6.2 Umkehrung von Top-Down

Bei dem Verfahren werden aus dem Automaten nach und nach die inneren Zustände entfernt, und dafür die Kanten mit zunehmend komplexeren regulären Ausdrücken beschriftet.

Zunächst wird ein neuer Startzustand eingeführt und durch einen λ -Übergang mit dem ur-

sprünglichen Startzustand verbunden. Außerdem wird ein neuer Endzustand eingeführt, und alle bisherigen erhalten einen λ -Übergang zum neuen. Alle bisherigen Endzustände sowie der bisherige Startzustand verlieren ihre Eigenschaft!



Außer dem Startzustand und dem Endzustand werden nun schrittweise alle Zustände nach folgenden Regeln entfernt:

NDEA		Reguläre Ausdruck
	\Rightarrow	
	\Rightarrow	
	\Rightarrow	

Beispiel zu 4.6.2 Umkehrung von Top-Down

4.7 Äquivalenz von Automaten

Definition

Zwei EA M_1 und M_2 heißen äquivalent, wenn $L(M_1) = L(M_2)$.

W | F 1 W | F 2

4.8 Satz von Rabin & Scott

Satz

Zu jedem NDEA M gibt es einen äquivalenten DEA M' (d.h. $L(M) = L(M')$)

Beweis

Sei $M = (K, \Sigma, \Delta, s, F)$ ein NDEA.

Idee: Potenzmengenkonstruktion

Wir untersuchen, welche Zustände M von s aus beim Lesen von w prinzipiell erreichen kann. Zustände von M' werden daher Teilmengen von K . Um mit λ -Übergängen umzugehen, definieren wir zu jedem $q \in K$ den Abschluss

$$E(q) := \{p \in K \mid (q, \lambda) \vdash_M^* (p, \lambda)\} \cup \{q\}$$

D.h. in $E(q)$ liegen alle $p \in K$, die man von q mit endlich vielen λ -Übergängen erreichen kann.

Beachte: $E(q)$ kann z.B. durch Breitensuche berechnet werden.

Definiere DEA $M' := (K', \Sigma, \delta', s', F')$ mit

- $K' := \mathcal{P}(K) \rightarrow |\mathcal{P}(K)| = 2^{|K|}$
- $s' := E(s)$
- $F' := \{Q \subseteq K \mid Q \cap F \neq \emptyset\}$
- $\delta'(Q, a) := \bigcup_{q \in Q} \{E(p) \mid p \in K \wedge (q, a, p) \in \Delta_M\}$

(alle Zustände, die man in M von jedem Zustand aus Q nach Lesen von beliebig vielen λ , dann a und wieder beliebig vielen λ erreichen kann)

Klar: M' ist DEA.

Zu zeigen: $L(M) = L(M')$

Behauptung: Für $w \in \Sigma^*$ und $p, q \in K$ gilt:

$$(q, w) \vdash_M^* (p, \lambda) \Leftrightarrow \exists P \in K' \text{ mit } (E(q), w) \vdash_{M'}^* (P, \lambda) \text{ und } p \in P \quad (\star)$$

Beachte: Beweis der Behauptung reicht aus, um $L(M) = L(M')$ zu zeigen, indem man $q = s$ und $p \in F$ wählt!

Beweis von (*) mittels vollständiger Induktion:**• Induktionsanfang**

(*) gilt für $|w| = 0$ d.h. $w = \lambda$

Gelte $(q, \lambda) \vdash_M^* (p, \lambda)$, d.h. $p \in E(q)$ und die rechte Seite gilt mit der Definition von M' . Umkehrung analog.

• Induktionsvoraussetzung

(*) gilt für Wörter der Länge $\leq n$ für ein $n \geq 0$

• Induktionsschritt

Sei $w := va$ mit $|v| = n$, $a \in \Sigma$

– „ \Rightarrow “ Sei $(q, \underbrace{va}_w) \vdash_M^* (r_1, a) \vdash_M^* (p, \lambda)$ eine Rechnung von M auf w . Insbesondere

gilt $(q, v) \vdash_M^* (r_1, \lambda)$ mit $r_1 \in K$

Nach IV gilt in M' :

$(E(q), v) \vdash_{M'}^* (R_1, \lambda)$ mit $r_1 \in R_1 \in K'$.

Die Schritte $(r_1, a) \vdash_M^* (p, \lambda)$ implizieren jetzt für M' , dass $p \in \delta'(R_1, a) =: P$. Somit gilt die rechte Seite in (*)

– „ \Leftarrow “ folgt analog

□

4.9 Abgeschlossenheit der von (N)DEAs akzeptierten Sprachen

Definition

Die Klasse der von (N)DEAs akzeptierten Sprachen ist abgeschlossen unter Vereinigung, Schnitt, Konkatenation, Kleene-Stern sowie Komplementbildung (d.h. z.B. werden L_1, L_2 von DEAs akzeptiert, dann auch $L_1^*, L_1 \cup L_2, L_1 \cap L_2, L_1 \cdot L_2, \dots$)

Realisierung:

- L_1^* : neuen Startzustand q_{vor} (akzeptierend, $q_{\text{vor}} \in F_1$) vorschalten, und alle akzeptierende Endzustände um einen λ -Übergang zu q_{vor} erweitern
- $L_1 \cup L_2$: Produktautomat mit $F = K_1 \times F_2 \cup K_2 \times F_1$
- $L_1 \cap L_2$: Produktautomat mit $F = F_1 \times F_2$
- $L_1 \cdot L_2$: Endzustände von M_1 um einen λ -Übergang zum Startzustand von M_2 erweitern, und dann Endzustände von M_1 und Startzustand von M_2 entfernen
- $\Sigma^* \setminus L_1$: Endzustände von M_1 tauschen, also $F' = K \setminus F$ (M_1 muss dafür deterministisch, endlich und vollständig sein)

(Wenn 2 Automaten involviert sind, sollte sichergestellt werden, dass die Zustandsmengen disjunkt sind, also $K_1 \cap K_2 = \emptyset$!)

4.10 Satz von Kleene

Satz

Die Menge der regulären Sprachen ist gleich der Menge der von DEAs akzeptierten Sprachen (für alle endlichen Alphabeten)

REG \subseteq DEA

Induktion über $m := \#\text{Operationen, um } x \text{ zu erzeugen.}$

Ind.-Anfang $m = 0 :$

x ist reg. Grundausdruck, dann ist $\mathcal{L}(x)$ entweder $\emptyset, \{\lambda\}$ oder $\{a\}, a \in \Sigma$, also wird jede dieser Sprachen von einem DEA akzeptiert.

Ind.-Annahme

Sei $m \geq 0$ und x ein reg. Ausdruck, der in $\leq m$ Operationen erzeugt werden kann. Dann ist $\mathcal{L}(x)$ von einem DEA akzeptierbar.

Ind.-Behauptung

Sei y reg. Ausdruck, der in $m + 1$ Schritten erzeugt wird. Dann wird $\mathcal{L}(y)$ von DEA akzeptiert.

Ind.-Schritt

Betrachte den letzten Konstruktionsschritt für y . Hierbei gibt es folgende Möglichkeiten:

- $y = y_1 \cdot y_2$
- $y = y_1 \cup y_2$
- $y = (y_1)^*$

y_1, y_2 sind in $\leq m$ Schritten konstruierbar, also werden nach IV $\mathcal{L}(y_1)$ und $\mathcal{L}(y_2)$ durch DEA akzeptiert. Die Behauptung folgt mit den Abschlusseigenschaften. \square

DEA \subseteq REG

Sei jetzt umgekehrt $L = L(M)$ von einem DEA $M = (K, \Sigma, \delta, s, F)$ mit $K := \{q_1, \dots, q_n\}$, $s := q_1$.

Zu $i, j \in \{1, \dots, n\}$, $k \in \{0, \dots, n\}$ definiere folgende Sprachen/Mengen

$$R(i, j, k) := \left\{ \begin{array}{l} v \in \Sigma^* \mid \text{startend in } q_i \text{ soll } M \text{ beim Lesen} \\ \text{von } v \text{ nach } q_j \text{ laufen; dazwischen werden nur} \\ \text{Zustände mit Index } \leq k \text{ durchlaufen} \end{array} \right\}$$

Speziell für

- $k = 0:$

$$R(i, i, 0) := \{\lambda\} \cup \{a \in \Sigma \mid \delta(q_i, a) = q_i\}$$

- $i \neq j:$

$$R(i, j, 0) := \{a \in \Sigma \mid \delta(q_i, a) = q_j\}$$

Beachte:

$$R(i, j, n) := \{v \in \Sigma^* \mid (q_i, v) \xrightarrow{M}^* (q_j, \lambda)\}$$

$$L(M) = \bigcup_{q_j \in F} R(1, j, n)$$

Es reicht daher zu zeigen, dass alle $R(i, j, k)$ durch reg. Ausdrücke darstellbar sind.

Beweis mit Induktion über k .

Ind.-Anfang

$k = 0 \rightarrow$ per Definition erfüllt.

Ind.-Schritt

$k - 1 \rightarrow k$: Seien alle $R(i, j, k - 1)$ regulär.

Wir wollen zeigen, dass auch alle $R(i, j, k)$ regulär sind.

$$\underbrace{R(i, j, k - 1)}_{\substack{\text{Rechnung läuft} \\ \text{nicht durch } q_k}} \cup \underbrace{R(i, k, k - 1)}_{\substack{\text{Rechnung bis} \\ \text{zum ersten} \\ \text{Besuch von } q_k}} \cdot \underbrace{R(k, k, k - 1)^*}_{\substack{\text{beliebig oft} \\ \text{über } q_k}} \cdot \underbrace{R(k, j, k - 1)}_{\substack{\text{Rechnung von} \\ q_k \text{ nach } q_j}}$$

Nach Ind-Voraussetzung sind alle R-Mengen rechts regulär, und mit Abschlusseigenschaften / Konstruktionsoperationen gilt das auch für $R(i, j, k)$. \square

4.11 Reguläre Pumping-Eigenschaft

Definition

Eine Sprache $L \in \Sigma^*$ hat die Reguläre Pumping-Eigenschaft, wenn gilt:

$\exists n \in \mathbb{N} \forall w \in L \text{ mit } |w| \geq n : \exists x, y, z \in \Sigma^* \text{ mit:}$

1. $w = xyz$
2. $|xy| \leq n$
3. $|y| \geq 1$ (bzw. $y \neq \lambda$)
4. $\forall i \in \mathbb{N}_0 : xy^i z \in L$

W | F 5 W | F 6

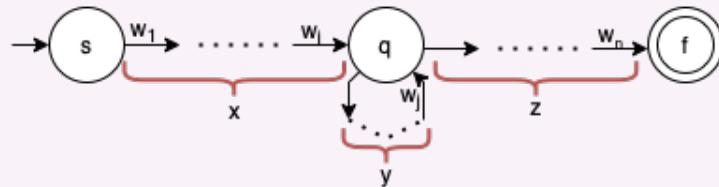
4.12 Reguläres Pumping-Lemma

Lemma

Sei $L \subseteq \Sigma^*$ regulär, dann hat L die Reguläre Pumping-Eigenschaft. (\Rightarrow)

Beweis

Sei $L = L(M)$ mit DEA $M = (K, \Sigma, \delta, s, F)$ und $n := |K|$. Dann gilt für eine akzeptierende Rechnung von M auf einem genügend langen Wort $w \in L$ mit $|w| \geq n$:



Beim Lesen der ersten n Zeichen von w ($|w| \geq n$) werden $n + 1$ Zustände besucht (Startzustand inklusive). Da M nur n Zustände hat, muss ein Zustand mehrfach besucht werden (Schubfachprinzip).

Sei q der erste Zustand, der mehr als einmal besucht wird. Dann gibt eine Zerlegung von w in $x, y, z \in \Sigma^*$ mit:

- $w = xyz$
 - x ist das Präfix von w bis zum ersten Besuch von q
 - y ist der wiederholte Teil (Zyklus) von w
 - z der verbleibende Suffix von w
- $|xy| \leq n \rightarrow xy$ enthält die ersten n Zeichen von w , bevor sich ein Zustand wiederholen muss.
- $|y| \geq 1 \rightarrow$ Da sich ein Zustand wiederholt, ist der Zyklus nicht leer
- $\forall i \in \mathbb{N}_0 : xy^i z \in L \rightarrow$ Da y ein Zyklus ist, kann dieser beliebig oft wiederholt werden, und die Rechnung wird immer noch akzeptiert.

□

Bemerkung

Da es sich bei dem Regulären Pumping-Lemma nur um eine Implikation, und nicht um eine Äquivalenz handelt, kann es ausschließlich genutzt werden, um zu zeigen, dass eine Sprache nicht regulär ist:

$$\neg\text{Pumping-Eigenschaft} \implies \neg L \text{ regulär}$$

(Es gibt nicht reguläre Sprachen, die die reguläre Pumping-Eigenschaft erfüllen.)

Beispiele von Beweisen

hat reg. PE	hat reg. PE nicht
reg. PE: $a^{4n}b^m$	keine reg. PE: $a^n b^n$ keine reg. PE: a^i mit i Primzahl

4.13 Äquivalenzrelationen bezüglich L bzw. M

Definition

- a) Sei $L \subseteq \Sigma^*$
 Zwei Wörter $x, y \in \Sigma^*$ heißen äquivalent bezüglich L (Notation $x \approx_L y$), wenn gilt:

$$\forall z \in \Sigma^* : x.z \in L \Leftrightarrow y.z \in L$$

- b) Sei M DEA

Zwei Wörter $x, y \in \Sigma^*$ heißen äquivalent bezüglich M (in Zeichen $x \sim_M y$), wenn gilt:

$$\exists q \in K \text{ mit } (s, x) \xrightarrow{M}^* (q, \lambda) \text{ und } (s, y) \xrightarrow{M}^* (q, \lambda)$$

Lemma

\approx_L und \sim_M sind Äquivalenzrelationen auf Σ^* !

Die Relation \sim_M ist eine Verfeinerung von $\approx_{L(M)}$, d.h. es gilt:

$$x \sim_M y \Rightarrow x \approx_{L(M)} y$$

Insbesondere: Die Äquivalenzklassen bzgl. \sim_M sind Teilmengen der Äquivalenzklassen

bzgl. $\approx_{L(M)}$.

Beweis

\approx_L und \sim_M sind Äquivalenzrelationen. Einfaches Prüfen der Eigenschaften.
Sei $x \sim_M y$, dann gilt:

$$\begin{aligned} \forall z \in \Sigma^* : (s, xz) \xrightarrow{M}^* (q, z) \xrightarrow{M}^* (p, \lambda) \quad (q, p \in K) \\ (s, yz) \xrightarrow{M}^* (q, z) \xrightarrow{M}^* (p, \lambda) \end{aligned}$$

In beiden Rechnungen terminiert der Automat jeweils in einem akzeptierenden Endzustand, oder jeweils nicht, daraus folgt:

$$\begin{aligned} xz \in L(M) &\Leftrightarrow yz \in L(M) \\ \implies x &\approx_{L(M)} y \end{aligned}$$

□

Folgerung

Sei L regulär und M ein DEA mit $L = L(M)$. Sei T die Anzahl ($\#$) der Äquivalenzklassen bzgl. \approx_L . Dann hat $M \geq T$ Zustände.

W | F 7 W | F 8

4.14 Satz von Myhill-Nerode

Satz

Sei $L \subseteq \Sigma^*$ regulär mit T vielen Äquivalenzklassen bzgl. \approx_L . Dann gibt es einen DEA M mit T Zuständen und $L = L(M)$. Damit hat M die minimale Anzahl an Zuständen für alle Automaten, die M erkennen, also ist M der minimale Zustandsautomat!

Insbesondere:

$$L \text{ regulär} \Leftrightarrow T \text{ endlich}$$

Beweis

Wenn L regulär ist, dann ist T endlich.

Wir konstruieren $M = (K, \Sigma, \delta, s, F)$. Zu $x \in \Sigma^*$ sei $[x] \subseteq \Sigma^*$ die Äquivalenzklasse, in der x liegt:

- $K := \{[x] \mid x \in \Sigma^*\}$ Menge der Äquivalenzklassen bzgl. \approx_L (T viele)
 $([x] := \{y \in \Sigma^* \mid x \approx_L y\} \subseteq \Sigma^*)$
- $s := [\lambda]$
- $F := \{[x] \mid x \in L\}$
- $\delta([x], a) := [xa]$

Beachte: δ ist wohldefiniert, d.h. δ ist unabhängig von der Wahl des Repräsentanten x für die ÄK $[x]$:

Sei $y \in \Sigma^*$ und $x \approx_L y$, dann muss gelten:

$$xa \approx_L ya \Leftrightarrow \forall z \in \Sigma^* : x \underbrace{az}_{\tilde{z}} \in L \Leftrightarrow y \underbrace{az}_{\tilde{z}} \in L \quad \checkmark$$

(gilt nach $x \approx_L y$). □

Es bleibt zu zeigen: $L = L(M)$

Für $x, y \in \Sigma^*$ gilt $([x], y) \vdash_M^* ([xy], \lambda)$ und damit

$$x \in L(M) \Leftrightarrow ([\lambda], x) \vdash_M^* (q, \lambda) \text{ mit } q \in F \text{ und } q = [x] \Leftrightarrow x \in L$$

□

$a^n b^n$ nicht regulär

a^n mit n Quadratzahl nicht regulär

W | F 9 W | F 10

4.15 Minimaler Zustandsautomat

Sei $M = (K, \Sigma, \delta, s, F)$ ein DEA.

Ohne Einschränkung entfernen wir alle Zustände, die von s aus nicht erreichbar sind. Durch Breitensuche erkunden wir, welche nicht erreichbar sind.

Definition

$A_M \subseteq K \times \Sigma^*$ ist definiert durch:

$$(q, w) \in A_M \Leftrightarrow (q, w) \vdash_M^* (f, \lambda) \quad \text{für ein } f \in F$$

Zwei Zustände p, q heißen äquivalent (in Zeichen $p \equiv_M q$), falls

$$\forall z \in \Sigma^* : (q, z) \in A_M \Leftrightarrow (p, z) \in A_M$$

Beachte: Seien $p \equiv_M q$ und seien

$$A_q := \{w \in \Sigma^* \mid (s, w) \vdash_M^* q\} \quad A_p := \{w \in \Sigma^* \mid (s, w) \vdash_M^* p\}$$

zwei ÄK bzgl \sim_M . Dann liegen diese beiden ÄK in den selben Klassen bzgl. \approx_L

Unser Ziel ist daher, die ÄK bzgl. \equiv_M zu berechnen. Dies geschieht über die Berechnung weiterer Äquivalenzrelationen $\equiv_0, \equiv_1, \equiv_2, \dots$

\equiv_M entsteht aus dieser Folge als eine Art Grenzwert.

Definition

Zu $n \in \mathbb{N}_0$ und $p, q \in K$ sei $p \equiv_M q$ definiert als:

$$p \equiv_n q \Leftrightarrow \forall z \in \Sigma^*, |z| \leq n : (q, z) \in A_M \Leftrightarrow (p, z) \in A_M$$

Speziell: $p \equiv_0 q$ hat zwei ÄK. Endzustände F und Nicht-Endzustände $K \setminus F$

Lemma

Seien $n \in \mathbb{N}_0, p, q \in K$. Dann gilt $p \equiv_n q$ genau dann wenn:

1. $q \equiv_{n-1} p$
2. $\forall a \in \Sigma : \delta(q, a) \equiv_{n-1} \delta(p, a)$

Beweis

Einsetzen der Definitionen von \equiv_n und \equiv_{n-1}

□

4.15.1 Algorithmus Zustandsminimierung

Algorithmus

1. Für $n = 0$ berechne F und $K \setminus F$ als ÄK zu \equiv_0 (Sofern beide Mengen $\neq \emptyset$, sonst ist $L = \Sigma^*$ oder $L = \emptyset$, also Zustandsminimierung trivial)
2. Solange $\equiv_n \neq \equiv_{n-1}$ berechne über das Lemma \equiv_n aus \equiv_{n-1}
3. Gilt für ein $n \equiv_n = \equiv_{n-1}$, dann hält das Verfahren und \equiv_n ist \equiv_M

Hieraus lässt sich der minimale Zustandsautomat unmittelbar berechnen.

Beachte: Das Verfahren hält, weil in jedem (echten) Schritt eine ÄK von Zuständen getrennt wird, aber nie zwei Zustände in bislang verschiedenen ÄK wieder in eine gemeinsame ÄK kommen.

Beispiel zu 4.15.1 Algorithmus Zustandsminimierung

W | F 11 W | F 12

4.16 Algorithmische Fragen zu DEAs und reg. Sprachen

4.16.1 Wortproblem:

Gegeben: DEA $M, w \in \Sigma^*$

Frage: Gilt $w \in L(M)$?

Algorithmus: Simuliere die Rechnung von M auf w und prüfe ob diese in $q \in F$ hält

4.16.2 Wortproblem II:

Gegeben: NDEA $M, w \in \Sigma^*$

Frage: Gilt $w \in L(M)$?

Variante 1: Wandle M in äquivalenten DEA, dann *Wortproblem*
(rechenintensiv, effizienter wenn mehrmals Wortproblem II durchgeführt wird)

Variante 2: Berechne zunächst $E(s)$. Lese danach den ersten Buchstaben w_1 von w und bilde $E_1 := \{q \mid q \text{ ist von } E(s) \text{ mit } w_1 \text{ erreichbar}\}$. Fahre analog mit w_2 fort. Prüfe am Ende, ob $E_n \cap F \neq \emptyset$.

4.16.3 Äquivalenzproblem

- Gegeben: M_1, M_2 DEAs
 Frage: Gilt $L(M_1) = L(M_2)$?
 Algorithmus: Minimiere beide Automaten. Wenn die minimalen DEAs unterschiedlich viele Zustände haben, dann folgt $L(M_1) \neq L(M_2)$. Andernfalls identifizierte die Startzustände beider Automaten. Identifizierte dann weiter die Zustände, wie sie beim Lesen von $a \in \Sigma$ angelaufen werden, bis entweder ein Widerspruch auftritt oder alle Zustände zugeordnet sind.

4.16.4 Äquivalenzproblem II

- Gegeben: M_1, M_2 NDEAs
 Frage: Gilt $L(M_1) = L(M_2)$?
 Algorithmus: Mache M_1, M_2 deterministisch und wende obigen Algorithmus an.

Geht das schneller / besser?

→ Kernproblem der **Komplexitätstheorie**

Das Problem ist PSPACE-vollständig. Es sind keine wesentlich besseren Algorithmen bekannt.

Bemerkung

Im Allgemeinen wächst die Anzahl der Zustände eines äquivalenten DEA exponentiell in der Anzahl der Zustände des NDEA.

Beispiel zu 4.16.4 Äquivalenzproblem II

4.16.5 Wortproblem für reguläre Sprachen

- Gegeben: regulärer Ausdruck $x, w \in \Sigma^*$
 Frage: Gilt $w \in \mathcal{L}(x)$?
 Algorithmus: Bilde äquivalente NDEA für $\mathcal{L}(x)$ und löse dort das Wortproblem

4.16.6 Endlichkeitsproblem

- Gegeben: DEA M
 Frage: Ist $L(M)$ endlich?
 Nach PL gilt: $L(M)$ ist unendlich g.d.w. $\exists w \in L(M)$ mit $|w| \geq n := \#\text{Zustände}(M)$
 Wie prüft man das?
 Algorithmus: Prüfe für alle Wörter der Länge n in Σ^* , ob sie von M akzeptiert werden.
 Falls $\Sigma^n \cap L(M) \neq \emptyset$, dann ist $L(M)$ unendlich.
 Sonst prüfe alle Wörter $|w| = n+1$ usw.
 Problem: Wann kann man aufhören?
 Dieser naive Algorithmus hält nicht, falls $L(M)$ endlich ist. Wir brauchen ein Argument, das uns garantiert, wann wir mit dem Algorithmus stoppen können.
 Aber: Es reicht den Algorithmus bis Länge $2n - 1$ auszuführen, dann muss ein $w \in L(M)$ gefunden worden sein, falls $L(M)$ unendlich ist ($|xz| \leq n-1, |y| \leq n$)

4.16.7 Leerheitsproblem

Gegeben: DEA M

Frage: Ist $L(M) = \emptyset$?

Algorithmus 1: Prüfe graphentheoretisch, ob ein $q \in F$ im Graphen des DEA erreichbar ist.

Algorithmus 2: Prüfe für alle w mit $|w| \leq n - 1$, ob eines in $L(M)$ liegt.

4.17 Homomorphiesatz für reguläre Sprachen

Der Satz hilft gelegentlich nachzuweisen, dass ein L nicht regulär ist, auch wenn es die reguläre PE erfüllt.

Definition

Seien Σ, Γ zwei endliche Alphabete.

Eine Abbildung $h : \Sigma^* \rightarrow \Gamma^*$ heißt **Homomorphismus** wenn gilt:

$$\forall u, v \in \Sigma^* : h(\underbrace{u.v}_{\in \Sigma^*}) = \underbrace{h(u).h(v)}_{\in \Gamma^*}$$

Beachte: Ein Homomorphismus ist eindeutig festgelegt durch die Bilder $h(a), a \in \Sigma$

$$h(\lambda) = h(\lambda.\lambda) = h(\lambda).h(\lambda) = h(\lambda) = \lambda$$

⊓

Beachte: $\tilde{h} : \Sigma \rightarrow \Gamma$ ist Homomorphismus, wenn man \tilde{h} auf Σ^* fortsetzt:

$$\tilde{h}(w_1 \dots w_n) = \tilde{h}(w_1) \cdot \dots \cdot \tilde{h}(w_n)$$

↪ kann nicht jeden Homomorphismus erzeugen:

$$h(a) = aa$$

$$h(b) = bb$$

(ein Buchstabe muss nicht auf genau ein Buchstabe abgebildet werden!) ⊥

Lemma

Sei $L \subseteq \Sigma^*$ regulär und $h : \Sigma^* \rightarrow \Gamma^*$ homomorph. Dann ist $h(L) := \{h(w) \mid w \in L\} \subseteq \Gamma^*$ regulär.

Beweis

Sei L gegeben durch einen regulären Ausdruck x . Wir bilden einen regulären Ausdruck indem wir in x jedes $a \in \Sigma^*$ durch $h(a)$ ersetzen. Der entstehende reguläre Ausdruck ist dann für $h(L)$. □

Lemma

Sei $h : \Sigma^* \rightarrow \Gamma^*$ Homomorphismus und $L \subseteq \Gamma^*$ regulär. Dann ist $h^{-1}(L) := \{w \in \Sigma^* \mid h(w) \in L\} \subseteq \Sigma^*$ regulär.

Beweis

Sei $L = L(M)$ für einen DEA M . Wir konstruieren einen DEA M' für $h^{-1}(L)$. Der Automat M' liest buchstabenweise $w_1 \dots w_n$ und simuliert in einem Schritt, was M potentiell in mehreren Schritten auf $h(w_i)$ tut \square

Beispiel für inversen Homomorphismus

W | F 13

4.18 ÄK → REG

Gegeben ein deterministischer endlicher Automat $M = (K, \Sigma, \delta, s, F)$, der die Sprache $L \subseteq \Sigma^*$ akzeptiert, also $L(M) = L$. Den regulären Ausdruck für die Äquivalenzklasse $A = [a]$ bezüglich der Nerode-Äquivalenzrelation \sim_M bestimmt man wie folgt:

1. Für M den minimalen Automaten M' bestimmen
2. Bestimme den richtigen Zustand q , indem man M' auf dem Repräsentanten $a \in A$ simuliert und schaut, in welchem Zustand die Rechnung endet (ob akzeptierend oder nicht ist irrelevant) (die Zustände des minimalen Automaten sind genau die Äquivalenzklassen bzgl. \sim_M)
3. Der reguläre Ausdruck α für A ist dann:

$$\alpha = \left(\bigcup \{\text{Pfad von } s \text{ zu } q\} \right) \cdot \left(\bigcup \{\text{Zyklus von } q \text{ zu } q\} \right)^*$$

Beispiel zu 4.18 ÄK → REG

Kapitel 5 Kontextfreie Sprachen

5.1 Grammatiken

Definition

- a) Eine **Grammatik** ist ein 4-Tupel $G = (V, \Sigma, R, S)$ mit
- | | |
|---|------------------------------|
| V | = endliches Alphabet |
| Σ | = Nicht-Terminalsymbole, NTS |
| $S \in V$ | = Startsymbol |
| $R \subseteq \underbrace{(V \cup \Sigma)^*.V.(V \cup \Sigma)^*}_{\substack{\text{linke Regelseite} \\ (\text{mind. 1 NTS})}}$ | = Menge der Ableitungsregeln |
| $\times \underbrace{(V \cup \Sigma)^*}_{\text{rechte Regelseite}}$ | |
- b) Für ein G ist die von G generierte Sprache $L(G) \subseteq \Sigma^*$ der in G ableitbaren Wörter so definiert:
- Eine Abbildung beginnt mit S
 - Angenommen, ein Wort $uvw \in (V \cup \Sigma)^+$ mit $v \in (V \cup \Sigma)^+$ ist schon erzeugt und es gibt eine Regel $v \rightarrow \tilde{v} \in R$, dann liefert die Anwendung der Regel als neues „Zwischenresultat“ $u\tilde{v}w$.
- Notationen:**
- | | |
|----------------------------------|-----------------------------|
| $uvw \overline{ }_G u\tilde{v}w$ | bei einer Regelanwendung |
| $uvw \overline{ }_G^* w$ | bei mehreren Regelanwendung |
- c) Die von G erzeugte Sprache ist die Menge aller Wörter, die aus S in endlich vielen Schritten ableitbar ist:

$$L(G) = \{w \in \Sigma^* \mid S \overline{|}_G^* w\}$$

Bemerkung

Ableiten in einem G ist i.A. ein nicht-deterministischer Prozess. Es kann mehrere anwendbare Regeln geben und eine Regel kann an mehreren Stellen anwendbar sein.

Grammatik zu $a^n b^n$

Grammatik zu $a^n b^n c^n$

W | F 14 W | F 15

5.2 Kontextfreie Grammatiken

spezielle strukturelle Einschränkung:

Die linke Regelseite enthält keine Kontexte, sondern besteht nur aus einem NTS:

Definition

- i) Eine Grammatik G heißt **kontextfrei**, wenn alle Regeln die Form $A \rightarrow w$ mit $A \in V$ und $w \in (V \cup \Sigma)^*$ haben, d.h.

$$R \subseteq V \times (V \cup \Sigma)^*$$

- ii) G heißt **rechtslinear** (bzw. **regulär**), wenn alle Regeln die Form $A \rightarrow wB$ bzw. $A \rightarrow w$ mit $A, B \in V, w \in \Sigma^*$, d.h.

$$R \subseteq V \times (\Sigma^* V \cup \Sigma^*)$$

- iii) L heißt **kontextfrei (kfr.)**, wenn es eine kfr. Grammatik G mit $L(G) = L$ gibt.

Lemma

Die rechtslinearen Sprachen sind genau die regulären Sprachen.

Beweis

- a) Sei G rechts-linear

Behauptung: $L(G)$ ist regulär

Ein äquivalenter NDEA hat V (Menge der NTS) als Zustandsmenge mit S als Startzustand, und aus $A \rightarrow wB$ wird zu $(A, w, B) \in \Delta$ (für $|w| \geq 2$ werden noch endlich viele, weitere Zustände zwischen geschalten)

- b) Sei L regulär mit DEA M und $L(M) = L$

In einer rechtslinearen Grammatik werden die Zustände von M die NTS (V) mit $S := s$, und die Übergänge $\delta(q, a) = p$ werden zu $Q \rightarrow aP$. Für akzeptierende Endzustände $q \in F$ werden noch die Regeln $Q \rightarrow \lambda$ hinzugefügt.

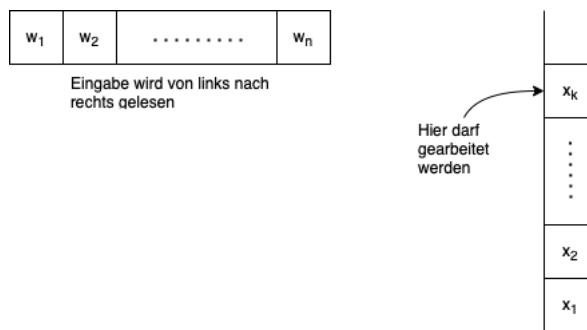
□

Folgerung

Die regulären Sprachen sind eine echte Teilmenge der kontextfreien Sprachen!

5.3 Kellerautomat

Wir erweitern die „Hardware“ eines DEA um einen Keller / Stack. Ein Stack ist eine Datenstruktur, die jeweils ein Wort eines Kelleralphabets Γ abspeichern kann. Es darf oben am Stack gearbeitet werden, d.h. Buchstaben aus Γ können gelesen, hinzugefügt oder gelöscht werden.



Definition

Ein Kellerautomat KA (push down automata PDA) ist ein 6-Tupel $M = (K, \Sigma, \Gamma, \Delta, s, F)$ mit

K	= endliche Zustandsmenge
Σ	= endliches Alphabet
Γ	= endliches Kelleralphabet
$s \in K$	= Startzustand
$F \subseteq K$	= Menge von Endzuständen
$\Delta \subseteq (K \times \Sigma \cup \{\lambda\} \times \Gamma \cup \{\lambda\}) \times (K \times \Gamma^*)$	= Überführungsrelation

M liest auf dem Eingabeband ein $a \in \Sigma$ oder λ und oben auf dem Stack ein $b \in \Gamma$ oder λ , und schreibt dann ein Wort aus Γ^* auf den Stack.

M startet mit Eingaben w in Zustand s bei leerem Stack. M akzeptiert ein $w \in \Sigma^*$, wenn w komplett gelesen ist, der Stack leer ist und M sich in einem $q \in F$ befindet.

$L(M) \subseteq \Sigma^*$ sind die von M akzeptierten Wörter.

Beispiel zu 5.3 Kellerautomat

5.3.1 Operationen

Operationen

„push“ b : b wird auf den Stack gelegt, $(q, u, \lambda) \xrightarrow{\Delta} (p, b)$

„pop“ b : b wird vom Stack gelesen, $(q, u, b) \xrightarrow{\Delta} (p, \lambda)$

$(u \in \Sigma \cup \{\lambda\}, b \in \Gamma)$

5.3.2 Konfiguration und Rechnung

Definition

Sei M ein PDA

i) Eine **Konfiguration** von M ist ein Element in $K \times \Sigma^* \times \Gamma^*$ mit

K = aktueller Zustand

Σ^* = noch zu lesender Teil der Eingabe

Γ^* = Wort im Stack

ii) Eine **Rechnung** von M :

Eine Konfiguration $(p, ay, \beta\eta)$ mit

p = Zustand

$ay =: x \in \Sigma^*, a \in \Sigma \cup \{\lambda\}, y \in \Sigma^*$

$\beta\eta =: z \in \Gamma^*, \beta \in \Gamma \cup \{\lambda\}, \eta \in \Gamma^*$

impliziert die Nachfolgekonfiguration $(q, y, \gamma\eta)$, wenn es in Δ die Transition $(p, a, \beta) \xrightarrow{\Delta} (q, \gamma)$ gibt.

Notation: $(p, x, z) \xrightarrow{M} (q, y, \gamma\eta)$ in einem Schritt

$(p, x, z) \xrightarrow{M}^* (q, y, \gamma\eta)$ in mehreren Schritten

Speziell: $x \in L(M) \Leftrightarrow (s, x, \lambda) \xrightarrow{M}^* (q, \lambda, \lambda), q \in F$

Bemerkung

Unser PDA-Modell ist nicht deterministisch.

Es gibt auch das Modell eines det. PDA. Dieses ist echt schwächer.

[W | F 16](#) [W | F 17](#)

5.4 Äquivalenz von PDA und kfr. Grammatik

Lemma

Sei L kfr. mit Grammatik G . Dann gibt es einen PDA M mit $L = L(M)$.

Beweis

Wir definieren einen PDA $M = (K, \Sigma, \Gamma, s, \Delta, F)$ mit

$$\begin{aligned} K &:= \{p, q\} \\ \Gamma &:= V \cup \Sigma \\ s &:= p \\ F &:= \{q\} \\ \Delta &:= \{(p, \lambda, \lambda) \rightarrow (q, S) \\ &\quad (q, \lambda, A) \rightarrow (q, X) \quad \forall (A \rightarrow X) \in R \\ &\quad (q, a, a) \rightarrow (q, \lambda) \quad \forall a \in \Sigma \\ &\quad \} \end{aligned}$$

□

Lemma

Sei M PDA. Dann gibt es eine kfr. Grammatik G mit $L(G) = L(M)$.

Bemerkung

Beachte: Es gibt verschiedene PDA-Modelle (mit unterschiedlichen Bedingungen), die aber gleich „mächtig“ sind. z.B.:

- Man kann ein ganzes Wort auf dem Keller lesen oder man darf pro Transition max. ein Buchstaben auf den Keller schreiben
- Der PDA akzeptiert immer mit leerem Keller, wenn w komplett gelesen ist ($F = K$)

→ Umsetzung:

Zu Beginn jeder Rechnung wird ein neues Kellersymbol z_0 auf den Keller gelegt. Dieses wird am Ende nur gelöscht, wenn sich der ursprüngliche Automat in einem akzeptierenden Endzustand befindet.

①

- Jede Transition muss auf dem Keller etwas lesen (d.h. kein λ)

(2)

→ Umsetzung:

Zu Beginn jeder Rechnung wird ein neues Kellersymbol z_0 auf den Keller gelegt.

Anschließend wird jede „alte“ Transition $(s, a, \lambda) \vdash (q, \beta)$ zu $(s, a, z_0) \vdash (q, \beta z_0)$.

Füge außerdem für alle $q \in K$ die Transition $(q, \lambda, z_0) \vdash (q, \lambda)$ hinzu (Löschen von z_0 am Ende).

Alle anderen Transitionen, wo λ auf dem Keller gelesen wird, werden wie folgt geändert:

$(q, a, \lambda) \vdash (q, \beta)$ wird zu $(q, a, b) \vdash (p, \beta b) \quad \forall b \in \Gamma \cup \{z_0\}$

Beweis

Sei $M = (K, \Sigma, \Gamma, \Delta, s, F)$ ein PDA. Ohne Einschränkung gilt, dass M die obigen Bedingungen ① und ② erfüllt!

Konstruktionsidee für G

- M akzeptiert mit leerem Stack, also muss jedes Symbol, das irgendwann auf dem Stack liegt, wieder gelöscht werden. Dazu sind entweder Teile von w oder λ zu lesen.
- Idee ist jetzt, solche Rechenschritte von M , in deren Verlauf auf dem Stack ein Symbol gelöscht wird, in den NTS von G zu kodieren.
- Wir wählen als NTS 3-Tupel der Bauart $[q, A, p]$ mit $p, q \in K, A \in \Gamma$
- Interpretation: M bewegt sich von q nach p und löscht dabei „netto“ A vom Stack.

Sei eine Transition von M $(q, a, A) \xrightarrow{M} (q_1, B_1 \dots B_m)$

M löscht zwar A , aber schreibt neue Symbole B_1, \dots, B_m auf dem Stack. Um A also „netto“ zu löschen, müssen danach B_1, \dots, B_m auch gelöscht werden. Zusätzlich wird ein $a \in \Sigma \cup \{\lambda\}$ gelesen. Beim Löschen von B_1, \dots, B_m läuft M durch irgendwelche Zustände q_2, \dots, q_m . Wir nehmen daher für die Transition folgende Regeln in G auf:

$$[q, A, p] \rightarrow a[q_1, B_1, q_2][q_2, B_2, q_3] \dots [q_m, B_m, p] \quad \forall q_2, q_3, \dots, q_m, p \in K \quad (p \stackrel{\Delta}{=} q_{m+1})$$

Aus jedem $[q_i, B_i, q_{i+1}]$ wird dann entweder B_i in einer Transition von M „netto“ gelöscht, oder es gibt wieder mehrere Zwischenzustände analog zu oben.

Insgesamt hat G folgende Regeln:

- (1) $S \rightarrow [s, z_0, q] \quad \forall q \in K$
- (2) $[q, A, q_{m+1}] \rightarrow a[q_1, B_1, q_2] \dots [q_m, B_m, q_{m+1}]$ für jede Transition $(q, a, A) \xrightarrow{M} (q_1, B_1 \dots B_m)$ in M und jede Wahl von $q_2, \dots, q_{m+1} \in K$
- (3) $[q, A, q_1] \rightarrow a$ für $a \in \Sigma \cup \{\lambda\}$, falls $(q, a, A) \xrightarrow{M} (q_1, \lambda)$ in M

Es bleibt zu zeigen: $L(G) = L(M)$

Es reicht zu zeigen:

$$(\star) \quad [q, A, p] \xrightarrow[G]{*} x \Leftrightarrow (q, x, A) \mid_M^* (p, \lambda, \lambda) \quad \forall p, q \in K, x \in \Sigma^*, A \in \Gamma$$

Begründung, dass der Nachweis von (\star) reicht:
 Sei $x \in L(G)$, also $\exists q \in K$ mit

$$S \rightarrow [s, z_0, p] \xrightarrow[G]{*} x \in \Sigma^* \Leftrightarrow (s, x, z_0) \vdash_M^* (p, \lambda, \lambda) \Leftrightarrow x \in L(M)$$

Beweis von (\star)

„ \Rightarrow “ mit Induktion über Anzahl der Ableitungsschritte in G

Induktionsanfang:

$n = 1$

$$[q, A, p] \xrightarrow[G]{1} x \in \Sigma$$

Dies ist eine Regel vom Typ (3), also gibt es in M die Transition $(q, x, A) \xrightarrow[M]{ } (p, \lambda)$ ($x \in \Sigma \cup \{\lambda\}$) und damit $(q, x, A) \vdash_M^* (p, \lambda, \lambda)$ in Konfigurationsschreibweise.
 Und somit ist der Ind.-Anfang korrekt.

Induktionsvoraussetzung:

„ \Rightarrow “ gilt für Ableitungen in G , die $\leq n$ Schritte haben.

Induktionsschritt:

Betrachte eine Ableitung in G $[q, A, p] \xrightarrow[G]{*} x \in \Sigma^*$ in $n+1$ Schritten.

Somit gilt in G :

$$[q, A, p] \rightarrow a[q_1, B_1, q_2][q_2, B_2, q_3] \dots [q_m, B_m, p] =: w_1 \quad \text{Typ(2) Regel}$$

$$\xrightarrow[G]{*} w_2 \xrightarrow[G]{*} w_3 \xrightarrow[G]{*} \dots \xrightarrow[G]{*} w_{n+1} = x \in \Sigma^*, \quad w_1 \in (V \cup \Sigma^*)$$

Sei jetzt $x = ax_1 \dots x_m$, wobei jedes $x_i \in \Sigma^*$ mit $[q_i, B_i, q_{i+1}] \xrightarrow[G]{*} x_i$, $1 \leq i \leq m$. Dies geht in $\leq n$ Schritten.

Nach **IV** gilt somit im PDA M :

$$(q_i, x_i, B_i) \vdash_M^* (q_{i+1}, \lambda, \lambda)$$

Die erste Regel von G ergibt sich aus $(q, a, A) \vdash_M^* (q_1, B_1 \dots B_m)$.

Daraus folgt

$$\begin{aligned} (q, x, A) &= (q, x_1 \dots x_m, A) \\ &\vdash_M^* (q_1, x_1 \dots x_m, B_1 \dots B_m) \\ &\vdash_M^* (q_2, x_2 \dots x_m, B_2 \dots B_m) \\ &\vdash_M^* \dots \\ &\vdash_M^* (q_m, x_m, B_m) \\ &\vdash_M^* (p, \lambda, \lambda) \end{aligned}$$

d.h. die rechte Seite von (\star) gilt.

„ \Leftarrow “ in (\star) folgt analog (Induktion über die Anzahl der Transitionen in M) □

Beispiel zu 5.4 Äquivalenz von PDA und kfr. Grammatik

W | F 18 W | F 19

5.5 Chomsky Normalform

Definition

Eine kfr. Grammatik G steht in Chomsky-Normalform (CNF), wenn alle Regeln folgendes Aussehen haben.

$$\begin{array}{ll} A \rightarrow BC & A, B, C \in V \\ A \rightarrow a & A \in V, a \in \Sigma \\ S \rightarrow \lambda & \text{sofern } \lambda \in L(G) \\ & \hookrightarrow S \text{ taucht in keiner Regel rechts auf.} \end{array}$$

Lemma

Zu jeder kfr. Grammatik G gibt es eine äquivalente in CNF.

Beweis

Wir eliminieren schrittweise verbotene Regeln:

Schritt 1

Falls $\lambda \in L(G)$: füge $S_{neu} \rightarrow \lambda$ und $S_{neu} \rightarrow S$ hinzu

Für Regeln $B \rightarrow \lambda$ verfahren wie folgt:

Sei beispielsweise eine Regel $A \rightarrow \alpha B \beta B \gamma$ in G vorhanden. Füge dann folgende Regeln hinzu:

$$\begin{aligned} A &\rightarrow \alpha \beta B \gamma \\ A &\rightarrow \alpha B \beta \gamma \\ A &\rightarrow \alpha \beta \gamma \end{aligned}$$

Verfahren analog für alle Regeln mit B auf der rechten Seite und lösche schlussendlich die Regel $B \rightarrow \lambda$

Schritt 2

Sei $A \rightarrow w_1 w_2 \dots w_k$ Regeln mit $w_i \in V \cup \Sigma, k \geq 2$

Ersetze diese durch $A \rightarrow Z_1 \dots Z_k$, $Z_i \rightarrow w_1 \forall 1 \leq i \leq k$ mit neuen NTS Z_i

Zwischenergebnis

Nach Schritt 2 hat die Grammatik nur noch Regeln der Form:

$$\begin{array}{l} S_{neu} \rightarrow \lambda (\checkmark) \\ A \rightarrow a \quad A \in V, a \in \Sigma (\checkmark) \\ A \rightarrow B \quad A, B \in V (\cancel{\checkmark}) \\ A \rightarrow E_1 \dots E_k, \quad k \geq 2 \quad A, E_i \in V (\cancel{\checkmark} \text{ für } k \geq 3) \end{array}$$

Schritt 3

Elimination von $B \rightarrow C$ durch Untersuchung von Ableitungssequenzen von C aus:

- $C \xrightarrow[G]{*} a \in \Sigma^*$: Füge $B \rightarrow a$ hinzu
- $C \xrightarrow[G]{*} E_1 E_2 \dots E_k \quad (k \geq 2)$: Füge $B \rightarrow E_1 E_2 \dots E_k$ hinzu

Lösche $B \rightarrow C$

Einer dieser Fälle muss nach endlich vielen Regelanwendungen auftreten, da sonst in einer Sequenz $C \rightarrow D_1 \rightarrow D_2 \rightarrow \dots \rightarrow D_l$ ein NTS doppelt auftritt (Schubfachprinzip), und die Sequenz somit nutzlos ist.

Schritt 4

Elimination von Regeln $A \rightarrow A_1A_2\dots A_k$ $k \geq 3$ mit neuen NTS T_1, \dots, T_{k-2}

$$\begin{array}{lcl} A & \rightarrow & A_1T_1 \\ T_1 & \rightarrow & A_2T_2 \\ & \vdots & \\ T_{k-2} & \rightarrow & A_{k-1}A_k \end{array}$$

□

Beispiel zu 5.5 Chomsky Normalform

5.6 Kontextfreies Pumping-Lemma

Definition

Sei $L \subseteq \Sigma^*$. L hat die kfr. PE., wenn gilt:

$$\exists m \in \mathbb{N} \ \forall w \in L, |w| \geq 2^m \ \exists r, s, t, u, v \in \Sigma^* :$$

- i) $w = rstuv$
- ii) $|su| \geq 1$
- iii) $|stu| \leq 2^m$
- iv) $\forall i \in \mathbb{N}_0 : rs^i tu^i v \in L$

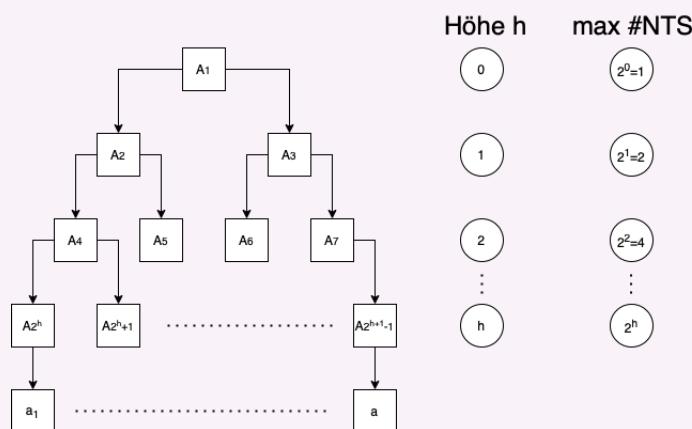
Lemma

Jede kfr. Sprache L hat die kfr. PE.

Beweis

Sei G kfr. in CNF mit $L = L(G)$.

Sei $m = |V|$. Betrachte den Ableitungsbaum in G für Wörter $w \in L$

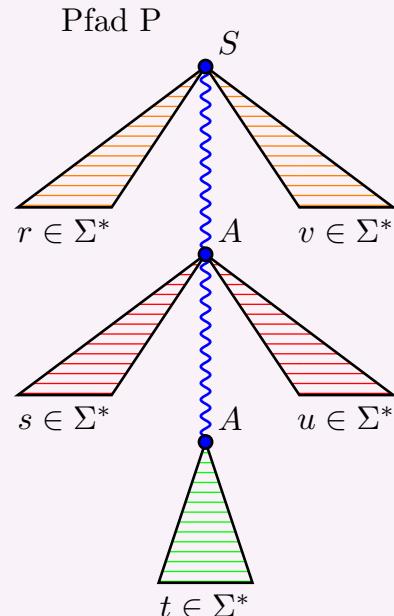


Auf Höhe h stehen max. 2^h NTS, d.h. mit einem Baum der Höhe $h+1$ kann man ein

Wort der maximalen Länge 2^h ableiten (wegen CNF kann man aus einem NTS immer nur maximal 2 machen).

Umgekehrt: Für ein $w \in L$ mit $|w| \geq 2^m$ benötigt G einen Ableitungsbaum der Höhe $h \geq m + 1$. Also gibt es einen Pfad P mit $m + 1$ Kanten, und wiederum $m + 2$ Ecken, wovon $m + 1$ viele ein NTS sind und der letzte ein $a \in \Sigma$ ist. Entlang P gibt es also (mindestens) ein NTS, das (mindestens) doppelt auftritt (Schubfachprinzip)!

Sei A das NTS, das vom Blatt a aus zur Wurzel S entlang P als erstes zweimal vorkommt.



Es gilt also bei der Ableitung von w :

$$\begin{aligned} \exists r, s, t, u, v \in \Sigma^* \text{ mit} \\ S &\xrightarrow[G]{*} rAv \\ A &\xrightarrow[G]{*} sAu \\ \Rightarrow A &\xrightarrow[G]{*} s^i Au^i, i \in \mathbb{N}_0 \\ A &\xrightarrow[G]{*} t \end{aligned}$$

Ableitungsbaum von w

Es folgt somit die Existenz einer Zerlegung von w mit i) und iv). Ferner folgt $|stu| \leq 2^m$ aus der Bedingung $|w| \geq 2^m$ (iii) und $|su| \geq 1$ aus der CNF (ii). \square

kfr. PE für $a^n b^n c^n$

kfr. PE für $L := \{xxx \mid x \in \{a, b\}^*\}$

Quadratzahl hat keine kfr. PE

5.7 Abschlusseigenschaften

Lemma

Die kfr. Sprache sind abgeschlossen unter Konkatenation, Vereinigung, Kleene Stern. Ebenso ist der Schnitt einer kfr. Sprache mit einer regulären Sprache wieder kfr.

Beweis

- | | |
|----------------|---|
| L_1, L_2 | Regel $S_{\text{neu}} \rightarrow S_1 S_2$ hinzufügen ($V_1 \cap V_2 = \emptyset$, Mengen der NTS muss disjunkt sein) |
| $L_1 \cup L_2$ | Regel $S_{\text{neu}} \rightarrow S_1 \mid S_2$ hinzufügen |
| L^* | Regeln $S \rightarrow SS \mid \lambda$ hinzufügen |

**Lemma**

Die kfr. Sprachen sind weder unter Schnitt noch unter Komplementbildung abgeschlossen.

Beweis

Wir schreiben das nicht-kfr. $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ als Schnitt zweier kfr. Sprachen $L_1 \cap L_2$:

$$\begin{aligned}L_1 &= \{a^n b^n c^m \mid n, m \in \mathbb{N}\} \\L_2 &= \{a^n b^m c^m \mid n, m \in \mathbb{N}\}\end{aligned}$$

Wären die kfr. Sprachen abgeschlossen unter Komplementbildung, dann auch unter Schnitt.

$$A \cap B = \Sigma^* \setminus ((\Sigma^* \setminus A) \cup (\Sigma^* \setminus B))$$



5.8 Das Wortproblem für kfr. Sprachen

Gegeben: Eine kfr. Grammatik in CNF, $w \in \Sigma^*$

Frage: Gilt $w \in L(G)$?

Anwendung: **CYK**-Algorithmus (von Cocke, Younger u. Kasami) verwendet dynamische Programmierung

Compilerbau → Syntaxanalyse

Sei $w = w_1 w_2 \dots w_n$.

Wir konstruieren eine $(n+1) \times (n+1)$ -Matrix \mathcal{N} . Die Einträge $\mathcal{N}[i, j]$ sind Teilmengen von V , also Mengen von NTS. Dabei soll gelten:

$$A \in \mathcal{N}[i, j] \Leftrightarrow A \xrightarrow[G]{*} w_i w_{i+1} \dots w_{j-1}, i < j$$

Speziell: $\mathcal{N}[i, j] = \emptyset$, für $i \geq j$

Klar: $w \in L(G) \Leftrightarrow S \in \mathcal{N}[1, n+1]$

Start: die Mengen $\mathcal{N}[i, i+1]$ (erste Nebendiagonale) sind leicht aus G zu ermitteln, denn $\mathcal{N}[i, i+1] \Leftrightarrow A \xrightarrow[G]{!} w_i$ muss Regel in G sein (G ist in CNF)

Der CYK-Algorithmus berechnet nun schrittweise alle $\mathcal{N}[i, i+2]$, dann $\mathcal{N}[i, i+3]$ usw. mittels „Matrixmultiplikation“:

Für zwei Teilmengen $M_1, M_2 \subseteq V$ definiere:

$$\begin{aligned}M_1 \oplus M_2 &:= M_1 \cup M_2 \\M_1 \odot M_2 &:= \{A \in V \mid A \xrightarrow[G]{*} BC \wedge B \in M_1, C \in M_2\}\end{aligned}$$

$$\mathcal{N}^{(1)} = \begin{pmatrix} \emptyset & \mathcal{N}[1, 2] & \emptyset & \dots & \emptyset \\ \vdots & \ddots & \mathcal{N}[2, 3] & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \emptyset \\ \emptyset & \dots & \emptyset & \mathcal{N}[n, n+1] & \emptyset \end{pmatrix}$$

$$\mathcal{N}^{(2)} = \mathcal{N}^{(1)} \odot \mathcal{N}^{(1)} = \begin{pmatrix} \emptyset & \emptyset & \mathcal{N}[1, 3] & \emptyset & \dots & \emptyset \\ \vdots & & \mathcal{N}[2, 4] & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \emptyset \\ \emptyset & \dots & & & \mathcal{N}[n-1, n+1] & \emptyset \end{pmatrix}$$

↑
Matrixprodukt
(zweite Nebendiagonale)

Berechnung von $\mathcal{N}[1,3]$

Analog erhält man

$$\mathcal{N}^{(3)} = \mathcal{N}^{(2)} \odot \mathcal{N}^{(1)} \oplus \mathcal{N}^{(1)} \odot \mathcal{N}^{(2)}$$

$$\mathcal{N}^{(k)} = \sum_{r=1}^{k-1} \mathcal{N}^{(r)} \odot \mathcal{N}^{(k-r)}$$

$$\mathcal{N} = \mathcal{N}^{(1)} \oplus \mathcal{N}^{(2)} \oplus \dots \oplus \mathcal{N}^{(n)}$$

Beispiel zu 5.8 Das Wortproblem für kfr. Sprachen

5.8.1 Komplexität CYK-Algorithmus

Bestimmung von \mathcal{N} erfordert

$$\sum_{r=2}^{n+1} (r-1) = \frac{1}{2} \cdot n \cdot (n+1)$$

Produkte von Matrizen. Jedes solcher erfordert $\leq n$ Produkte von Mengen von NTS
 $\rightarrow O(n^3)$ Produkte von Mengen. Jedes Produkt kann mit $O(|V|^2)$ Operationen berechnet werden.

W | F 20 W | F 21

Kapitel 6 Turingmaschinen

Gibt es sinnvolle weitere Sprachklassen oberhalb der kontextfreien Sprachen oder sogar unendliche Hierarchien solcher Klassen mit immer mächtigeren Automatenmodellen / Grammatiken?

Frage: Berechnen Endliche Automaten und Kellerautomaten irgendwas / Funktionen?
Ja, bei DEAs wird die charakteristische Funktion.

$$\chi_L : \Sigma^* \rightarrow \{0, 1\}$$
$$\chi_L(w) = \begin{cases} 1, & w \in L \\ 0, & w \notin L \end{cases}$$

berechnet.

Jetzt folgt die Einführung eines weiteren Algorithmenmodells, das in zu begründender Weise das mächtigste Modell ist, um Wortfunktionen

$$f : \Sigma^* - \rightarrow \Sigma^*$$

zu berechnen.

Das Modell der Turingmaschine hat sich als fundamental erwiesen bei Präzisierung der Begriffe *Algorithmus, Berechenbarkeit*.

6.1 Arbeitsweise der Turingmaschine

Das Speichermedium ist ein nach links und rechts unendliches Band, unterteilt in abzählbar viele Speicherzellen. Jede Zelle enthält zu jedem Zeitpunkt ein Symbol aus einem Bandalphabet Γ mit $\Sigma \subseteq \Gamma$. Γ enthält immer ein Blank-Symbol \square , das nicht in Σ liegt.

Die TM besitzt einen Lese-/Schreibkopf. Diesen bewegt sie nach links/rechts und steht zu jedem Zeitpunkt auf einer Zelle in einem gewissen Zustand. In einem Schritt liest die Maschine den Zelleneintrag unter dem Kopf, und tut abhängig vom Eintrag und dem aktuellen Zustand folgendes:

- Schreibt ein (potenziell neues) Symbol in die Zelle
- Geht in einen (potenziell neuen) Folgezustand
- Bewegt sich maximal um eine Zelle: L, N, R

6.2 Formale Definition

Definition

- a) Eine TM M ist ein 6-Tupel $M = (K, \Sigma, \Gamma, \delta, s, F)$ mit

K endliche Zustandsmenge

$s \in K$ Startzustand

$F \subseteq K$ Endzustände

Σ, Γ endliche Alphabete mit $\Sigma \subseteq \Gamma$ und

$\square \in \Gamma \setminus \Sigma$ als Blanksymbol

$\delta : K \setminus F \times \Gamma \rightarrow K \times \Gamma \times \{L, N, R\}$

- b) Für eine Eingabe $w \in \Sigma^*$ rechnet M wie folgt:

i) Als **Startkonfiguration** definiert man, dass der Kopf im Zustand s auf w_1 steht, und das Band links und rechts von w enthält ausschließlich \square (speziell $w = \lambda$: Kopf startet auf einem \square).

ii) Ein Rechenschritt wird durch δ beschrieben. Wird jemals ein $q \in F$ erreicht, hält M , sonst rechnet M unendlich lang weiter.

- c) Falls M für Eingabe w hält, dann akzeptiert M das Wort w , sonst nicht.

$L(M) := \{w \in \Sigma^* \mid M \text{ hält auf } w\}$ ist die von M akzeptierte Sprache

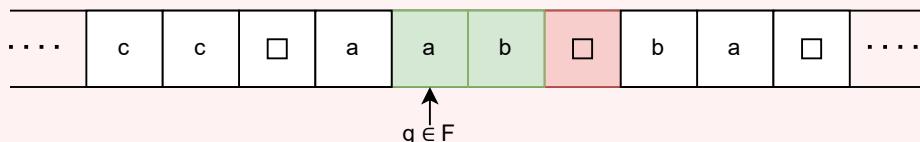
- d) Zu M gehört eine partielle Funktion $f_M : \Sigma^* \dashrightarrow \Sigma^*$

f_M ist so definiert:

Hält M auf Eingabe w in $q \in F$, so ist dasjenige Wort aus Σ^* das Resultat $f_M(w)$, welches in der Zelle beginnt, auf der der Kopf am Ende steht, und bis rechts bis zum ersten Symbol (exklusiv) aus $\Gamma \setminus \Sigma$ reicht.

Falls M auf w nicht hält, so gilt: $f_M(w) = \perp$ (undefiniert).

Beispielsweise folgendes Band:



Das Resultat hier wäre das Wort ab , da der Kopf auf dem a steht und nach dem b ein $\square \in \Gamma \setminus \Sigma$ kommt.

- e) Ein $L \subseteq \Sigma^*$ heißt Turing-akzeptierbar genau dann, wenn $\exists \text{TM } M$ mit $L(M) = L$.

Eine partielle Funktion $f : \Sigma^* \dashrightarrow \Sigma^*$ heißt Turing-berechenbar genau dann, wenn $\exists \text{TM } M$ mit $f_M(w) \equiv f(w)$ (d.h. $f(w) = f_M(w) \forall w \in L(M)$ und $f(w) = \perp \forall w \notin L(M)$).

einfache TMs

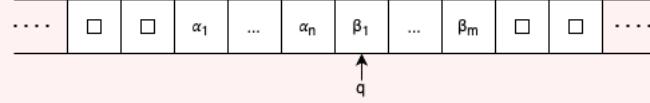
Definition

Sei M eine TM

- a) Eine Konfiguration von M ist ein Element aus $\Gamma^* \times K \times \Gamma^*$ (Band links vom

Kopf, aktueller Zustand, Band rechts vom Kopf)

Für $\alpha q \beta$ mit $\alpha = \alpha_1 \dots \alpha_n \in \Gamma^*$ und $\beta = \beta_1 \dots \beta_m \in \Gamma^*$ sieht das Band so aus:



- b) $\tilde{\alpha} p \tilde{\beta}$ ist (unmittelbare) Nachfolgekonfiguration von $\alpha q \beta$, wenn sie in (einem) mehreren Schritten aus $\alpha q \beta$ erreichbar ist

$$\alpha q \beta \vdash^* \tilde{\alpha} p \tilde{\beta}$$

[W | F 22](#) [W | F 23](#) [W | F 24](#)

6.3 Funktionskomposition

Lemma

Seien M_1, M_2 TM über Σ , dann gibt es eine TM M , die die Funktionskomposition $f_{M_2} \circ f_{M_1}$ berechnet, d.h. $f_M(w) \equiv f_{M_2} \circ f_{M_1}(w) \forall w \in \Sigma^*$ (Wenn $f_{M_1}(w)$ und $f_{M_2}(w)$ definiert sind, dann ist $f(w) := f_{M_2}(f_{M_1}(w))$).

→ die Turing-berechenbaren Fkt. sind abgeschlossen unter Komposition.

Beweis

Das Bandalphabet Γ von M sei $\Gamma_1 \cup \Gamma_2$ sowie zwei neue Symbole $\#_L$ und $\#_R$. Für Input $w_1 \dots w_n$ und Startkonfiguration $\square w_1 \dots w_n \square$ geht M zunächst einen Schritt nach links, setzt ein $\#_L$, läuft dann an das Ende von w und eine Zeile nach rechts, setzt dort $\#_R$ und läuft dann zurück zu w_1

q_0	a	q_1	a	L	$\forall a \in \Sigma$
q_0	\square	\dots	\dots	\dots	
q_1	\square	q_2	$\#_L$	R	
q_2	a	q_2	a	R	$\forall a \in \Sigma$
q_2	\square	q_3	$\#_R$	L	(laufe nach rechts)
q_3	a	q_3	a	L	$\forall a \in \Sigma$
q_3	$\#_L$	q_4	$\#_L$	R	

Jetzt simuliert $M M_1$. Immer wenn M auf ein $\#_L, \#_R$ stößt, wird es vor dem nächsten Schritt um eine Stelle verschoben, um Arbeitsplatz zu schaffen. Falls M_1 hält mit Resultat $v = f_{M_1}(w)$, dann muss M auf dem Band eine Startkonfiguration von M_2 mit Eingabe v „vorbereiten“. Dazu löscht M alles links von v bis $\#_L$ (inklusive) und rechts von v bis $\#_R$ (inklusive) und läuft zurück zu v_1 . Dann simuliert $M M_2$ auf v . □

ähnliche Routinen für TMs

- Eine TM simuliert eine andere TM
- Eine TM simuliert mehrere andere TMs, indem für jede simulierte TM ein eigener Bereich auf dem Band reserviert wird

6.4 Äquivalenz charakteristische Funktion und Turing-akzeptierbar

Definition

Folgende Eigenschaften für eine Sprache $L \subseteq \Sigma^*$ sind äquivalent:

- ① L wird von TM M_1 akzeptiert und \overline{L} wird von TM M_2 akzeptiert.
- ② \exists TM M mit zwei Haltezuständen q_{ja} und q_{nein} . M hält für $w \in L$ in q_{ja} und für $w \notin L$ in q_{nein} .
- ③ \exists TM \tilde{M} mit $f_{\tilde{M}} = \chi_L$ bzw. die charakteristische Funktion $\chi(w) = \begin{cases} 1, & w \in L \\ 0, & w \notin L \end{cases}$ ist berechenbar

Beweis

- (1) \Rightarrow (2)

Für Eingabe $w \in \Sigma^*$ simuliert M abwechselnd M_1 und M_2 (aus (1)) schrittweise auf w in zwei getrennten Bandbereichen, bis genau eine der Simulationen hält. Falls M_1 hält, dann stoppt M in q_{ja} , sonst in q_{nein} .

- (2) \Rightarrow (3)

Die TM \tilde{M} simuliert die TM M aus (2), und wenn M in q_{ja} hält, arbeitet \tilde{M} noch etwas weiter, um das Resultat „1“ zu schreiben. Andernfalls schreibt \tilde{M} das Resultat „0“.

- (3) \Rightarrow (1)

M_1 simuliert \tilde{M} . Bei Resultat 1 hält M_1 , bei Resultat 0 läuft M_1 künstlich in einem ∞ -Zyklus (Informationen werden weggeworfen).

Analog für M_2 .

□

6.5 Entscheidbarkeit

Definition

Sei $L \subseteq \Sigma^*$

- L heißt entscheidbar (recursive decidable), wenn χ_L berechenbar ist (d.h. alle Eigenschaften 1-3 gelten.)
- Eine Turing-akzeptierbare Sprache heißt auch semi-entscheidbar (semi-decidable)

Folgerung

$$\begin{aligned} L \text{ entscheidbar} &\Leftrightarrow L \text{ ist semi-entscheidbar und } \overline{L} \text{ ist semi-entscheidbar} \\ &\Leftrightarrow \overline{L} \text{ ist entscheidbar} \end{aligned}$$

(REC: Menge der entscheidbaren Sprachen)

TM die $L := \{w \in \{a,b\}^* \mid 2 \text{ teilt } |w|\}$ entscheidet

6.6 Rekursiv aufzählbar

Definition

$L \subseteq \Sigma^*$ heißt rekursiv aufzählbar, wenn es eine TM M gibt mit $f_M(\Sigma^*) = L$

(\mathcal{RE} : Menge der rekursiv aufzählbaren Sprachen)

Intuition (andere Definition)

Eine Sprache $L \subseteq \Sigma^*$ heißt rekursiv aufzählbar, wenn $L = \emptyset$ oder wenn es eine surjektive, berechenbare Abbildung $f : \mathbb{N} \rightarrow L$ gibt.

Rekursiv aufzählbar ist sehr eng verwandt mit **Abzählbarkeit**. Bei Abzählbarkeit gibt es eine bijektive Abbildung f von \mathbb{N} nach L , die Wörter in L werden also durch f „nummeriert“. f liefert somit eine gelistete Darstellung von L :

$$L = \{f(0), f(1), f(2), \dots\}$$

Bei Abzählbarkeit muss diese Funktion f lediglich existieren, nicht zwingend berechenbar sein!

Bei rekursiv aufzählbaren Sprachen muss diese Funktion aber berechenbar sein, daher kann man einen „*Aufzähler*“ für L bauen. Mit Aufzähler meint man eine TM, die unendlich läuft und nacheinander alle Wörter von L *aufzählt* (führt keine normale Rechnung mit Halt und Resultat aus), wobei die tatsächliche Reihenfolge der Wörter bei der Ausgabe irrelevant ist. Dies geht, indem sie fortlaufend $f(0), f(1), f(2), \dots$ berechnet und somit $w_1 w_2 w_3 \dots$ *aufzählt* (f ist berechenbar, also gibt es eine TM, die f berechnet, und diese kann von der Aufzähler-TM intern simuliert werden)

Ferner spielt bei rekursiv aufzählbar es keine Rolle, ob ein Element mehrfach aufgezählt wird, wichtig ist nur, dass jedes irgendwann von f berechnet wird. Daher enthält die Definition lediglich surjektiv (alle Elemente werden getroffen) und nicht bijektiv wie bei Abzählbarkeit.

Satz

L ist semi entscheidbar $\Leftrightarrow L$ ist rekursiv aufzählbar.

Beweis

- „ \Rightarrow “

Sei M TM mit $L(M) = L$. Folgende TM \tilde{M} zählt dann L rekursiv auf: \tilde{M} kopiert die Eingabe w in einen abgetrennten Bereich des Bands, und simuliert M auf der Kopie. Falls M hält, gibt \tilde{M} w aus. Dann folgt $f_{\tilde{M}}(\Sigma^*) = L$.

- „ \Leftarrow “

Sei \tilde{M} TM mit $f_{\tilde{M}}(\Sigma^*) = L$. Wir suchen TM M mit $L(M) = L$.

Sei $w \in \Sigma^*$ Eingabe für M . M soll herausfinden ob es ein Urbild $x \in \Sigma^*$ gibt, mit $f_{\tilde{M}}(x) = w$.

Naive Idee, um so ein x zu finden:

Zähle Σ^* auf (berechenbar) und prüfe der Reihe nach jedes Wort in der Aufzählung (x_1, x_2, x_3, \dots) , ob $f_{\tilde{M}}(x_i) = w, i = 1, 2, \dots$

Problem:

Es kann sein, dass $f_{\tilde{M}}(x_j)$ undefiniert ist und die Rechnung daher niemals hält, auch wenn $f_{\tilde{M}}(x_i) = w$ für ein $i > j$.

Lösung:

Verwende eine berechenbare Aufzählung von $\Sigma^* \times \mathbb{N}$, d.h.

$$g : \mathbb{N} \rightarrow \Sigma^* \times \mathbb{N}$$

M arbeitet so:

Für $i = 1, 2, \dots$ berechnet M den Wert $g(i) = (x_i, t_i)$, $x_i \in \Sigma^*$, $t_i \in \mathbb{N}$ und simuliert \tilde{M} für $\leq t_i$ Schritte auf x_i :

- Falls \tilde{M} auf x_i in $\leq t_i$ Schritten hält und w ausgibt ($f_{\tilde{M}}(x_i) = w$), dann hält M .
- Falls \tilde{M} in $\leq t_i$ Schritten zwar hält, aber $f_{\tilde{M}}(x_i) \neq w$ oder \tilde{M} hält nicht in t_i Schritten, dann erhöhe auf $i + 1$ und wiederhole alles.

Es folgt: $L(M) = L$, denn falls $w \in L$, dann gibt es x mit $f_{\tilde{M}}(x) = w$ und \tilde{M} rechnet t Schritte auf x .

(Zu (x, t) gibt es ein $i_0 \in \mathbb{N}$ mit $g(i_0) = (x, t)$ und M hält auf i_0 . Falls $w \notin L$, dann hält M auf keinem i .)

□

Intuition zum Beweis

- „ \Rightarrow “: Jede rekursiv aufzählbare Sprache L ist semi-entscheidbar

L ist rekursiv aufzählbar, also gibt es eine TM die alle Wörter in L nacheinander aufzählt. Die TM M , die nun die Semi-Entscheidbarkeit berechnen soll, kann diesen Aufzähler aufzählen lassen und halten, sobald das gesuchte Wort w auftritt. Dies geschieht für $w \in L$ nach endlich vielen Schritten, und wenn $w \notin L$, dann wird w nie in der Aufzählung auftauchen und M läuft unendlich.

- „ \Leftarrow “: Jede semi-entscheidbare Sprache L ist rekursiv aufzählbar

Ziel

L ist semi-entscheidbar, also gibt es eine TM M , die für $w \in L$ hält und für $w \notin L$ unendlich läuft. Wir müssen also M so ansteuern, dass wir insgesamt alle Elemente von L identifizieren, ohne in einer unendlichen Rechnung stecken zu bleiben.

Es hilft **Cantorsche Paarungsfunktion** $\pi : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, denn diese ist eine berechenbare, bijektive Funktion, also ist $\pi^{-1} : \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$ auch bijektiv und berechenbar. Wegen der **längen-lexikografischen Ordnung** gibt es eine berechenbare Funktion von $f^{-1} : \mathbb{N} \rightarrow \Sigma^*$, somit kann man insgesamt aus π^{-1} und f^{-1} eine berechenbare Funktion $g : \mathbb{N} \rightarrow \Sigma^* \times \mathbb{N}$ konstruieren.

Die TM \tilde{M} , die L nun rekursiv aufzählen soll, berechnet somit nun nacheinander $g(0), g(1), g(2), \dots$, wobei $g(i) = (x_i, t_i)$ $x_i \in \Sigma^*$, $t_i \in \mathbb{N}$ ist, und \tilde{M} simuliert dabei

M (die TM, die die Semi-Entscheidbarkeit berechnet) auf x_i mit t_i Schritten. Hält M innerhalb der t_i Schritte, dann zählt \tilde{M} dieses Resultat von f_M auf.

W | F 25 W | F 26

6.7 Äquivalenz beliebige Grammatik und semi-entscheidbar

Satz

Ein $L \subseteq \Sigma^*$ ist durch eine beliebige Grammatik G genau dann erzeugbar, wenn es Turing-akzeptierbar (semi-entscheidbar) ist.

Beweis

- „ \Rightarrow “
Sei $L = L(G)$ für eine Grammatik G mit Regeln $R = \{R_1, \dots, R_n\}$. Wir suchen TM M mit $L = L(M)$.
Wir zeigen stattdessen, dass L rekursiv aufzählbar ist:

Zähle R^* auf. Für jeden Satz von endlich vielen Regeln $(r_1, \dots, r_s) \in R^*$ werden alle möglichen Ableitungen (endlich viele) in G simuliert, die der Reihe nach r_1, \dots, r_s verwenden. Wenn eine dieser Ableitungen ein $w \in \Sigma^*$ ergibt, dann gibt $M w$ aus, andernfalls rechnet M unendlich lang.

- „ \Leftarrow “
Sei jetzt $L = L(M)$ für eine TM M . Konstruiere G mit $L(G) = L$. G simuliert eine akzeptierende Rechnungen von M rückwärts. Die Regeln von G behandeln Teile von Konfigurationen von M .

O.E. sei $\Sigma \cap K = \emptyset$ und Sondersymbole $S, \#_l, \#_r$ außerhalb von $\Sigma \cup K$. M akzeptiere immer in Konfiguration:

$$\begin{array}{c} \dots | \square | \#_l | \square | \#_r | \square | \dots \\ \uparrow \\ k \in F \end{array}$$

Wir schreiben diese Konfiguration als $\#_l k \square \#_r$ und nehmen als Start in G die Regel:

$$S \rightarrow \#_l k \square \#_r$$

Die NTS von G sind $S, K, \#_l, \#_r, \square$.

Sei $\delta(q, a) = (p, b, L)$ Operation in M :

$$\begin{array}{c} \dots | u_1 | a | u_2 | \dots \\ \uparrow \\ q \end{array} \quad \text{wird durch } \delta \text{ zu} \quad \begin{array}{c} \dots | u_1 | b | u_2 | \dots \\ \uparrow \\ p \end{array}$$

wird wird in G zu $p u_1 b \rightarrow u_1 q a$

(Analog für andere δ -Übergänge.)

Die Startkonfiguration von M $\square | w_1 | w_2 | \dots | w_n | \square$ entspricht dem Ende aller Ableitungen in G . Die Ableitungen führen auf Seiten der Grammatik also schlussendlich zum String $\#_l s w_1 \dots w_n \#_r$, daher benötigt G noch Regeln zum

Löschen von $\#_l s$ und $\#_r$:

$$\begin{aligned}\#_l S w_1 &\rightarrow w_1, \quad \forall w_1 \in \Sigma \\ \#_r &\rightarrow \lambda\end{aligned}$$

□

6.8 Halteproblem

Konstruktion einer universellen TM U

Sei M eine TM über einem endlichen Alphabet Γ .

Jede TM M kann als String über einem festen Alphabet Δ kodiert/interpretiert werden. Δ enthält z.B. die Symbole $0, 1, \square, \delta, q_1, \dots, q_m, (,), \rightarrow, \dots$

Jede TM M entspricht dann einem Code $C_M \in \Delta^*$. U arbeitet über Δ und erhält als Eingabe Paare $(x, w) \in \Delta^* \times \Delta^*$ ($\Sigma \subseteq \Delta$) und arbeitet so:

- Wenn x im obigen Sinn die Kodierung C_M einer TM M ist, dann simuliert $U M$ auf w
- Falls x kein korrekter Code ist, oder w kodiert keine korrekte Eingabe, dann hält U nicht auf (x, w)

Definition

Das **Halteproblem** H ist folgendes Entscheidungsproblem (EP):

Eingabe: Code C_M einer TM M und Eingabe w für M

Frage: Hält M auf w

$$H := \{(C_M, w) \mid M \text{ hält auf } w\}$$

Satz

H ist semi-entscheidbar

Beweis

U hält genau auf H

□

Wir zeigen nun, dass \overline{H} nicht semi-entscheidbar ist

Beweis

Indirekter Beweis:

Angenommen \overline{H} ist semi-entscheidbar. Dann wäre H entscheidbar. Ferner wäre auch folgendes Problem entscheidbar:

$$H_1 := \{w \mid w \text{ kodiert TM } M \text{ und } M \text{ hält auf } w\}$$

Wenn H_1 entscheidbar wäre, müsste $\overline{H_1}$ semi-entscheidbar sein, also es existiert eine Turingmaschine, die $\overline{H_1}$ akzeptiert.

Sei \tilde{M} eine TM mit $L(\tilde{M}) = \overline{H_1}$ und sei $C_{\tilde{M}}$ ihr Code. Wegen $H_1 \cap \overline{H_1} = \emptyset$ (disjunkt) besteht nun die Frage, ob $C_{\tilde{M}} \in H_1$ oder $C_{\tilde{M}} \in \overline{H_1}$.

Fall 1: $C_{\tilde{M}} \in \overline{H_1}$

Wegen $L(\tilde{M}) = \overline{H_1}$ (Definition von \tilde{M}) und $C_{\tilde{M}} \in \overline{H_1}$ muss \tilde{M} auf $C_{\tilde{M}}$ halten (Laut Definition von \tilde{M}).

Da dann aber \tilde{M} auf $C_{\tilde{M}}$ hält, folgt $C_{\tilde{M}} \in H_1$ (Definition von H_1), was im Widerspruch zu $C_{\tilde{M}} \in \overline{H_1}$ steht $\not\vdash$.

Fall 2: $C_{\tilde{M}} \in H_1$

Wegen $C_{\tilde{M}} \in H_1$ hält \tilde{M} auf $C_{\tilde{M}}$ (Definition von H_1).

Da dann aber \tilde{M} auf $C_{\tilde{M}}$ hält, und $L(\tilde{M}) = \overline{H_1}$ (Definition von \tilde{M} , \tilde{M} hält nur für $\overline{H_1}$), folgt $C_{\tilde{M}} \in \overline{H_1}$ was im Widerspruch zu $C_{\tilde{M}} \in H_1$ steht $\not\vdash$.

□

Satz von Turing, 1936

Das Halteproblem ist semi-entscheidbar, aber nicht entscheidbar.

Folgerung

Die Turing-akzeptierten Sprachen sind nicht abgeschlossen unter Komplementbildung.

Es gibt noch weitere unentscheidbare Probleme, z.B.:

- Das 10. Hilbertsche Problem
Gegeben ist ein multivariates Polynom $f(x_1, \dots, x_n)$ mit rationalen Koeffizienten.
Frage: Hat f eine ganzzahlige NST?
- Das Wortproblem für endlich präsentierte Gruppen

Beispiel zu $\text{Def}(f)$ und $\text{Bild}(f)$

6.9 Reduktionen

Definition

Seien $A, B \subseteq \Sigma^*$

A heißt many-one reduzierbar auf B , wenn es eine totale berechenbare Funktion $f : \Sigma^* \rightarrow \Sigma^*$ gibt, mit

$$\forall x \in \Sigma^* : x \in A \Leftrightarrow f(x) \in B$$

Notation: $A \leq^{mo} B$

Beachte: \leq^{mo} ist eine Relation auf der Menge der Sprachen
 \leq^{mo} ist reflexiv (mit $f(x) = id$) und transitiv

Lemma

1. Seien $A \leq^{mo} B$ und B entscheidbar, dann ist auch A entscheidbar.

2. Seien $A \leq^{mo} B$ und A unentscheidbar, dann ist auch B unentscheidbar.

Beweis

1. Um $w \in A$ zu entscheiden, wird $f(w)$ berechnet und dann das Entscheidungsverfahren für B angewandt.

2. Angenommen A ist unentscheidbar, aber B ist entscheidbar. Da B entscheidbar ist, kann man wie in 1. $f(w)$ berechnen, $f(w) \in B$ entscheiden, und somit auch $w \in A$ entscheiden. Dies ist ein Widerspruch, da A unentscheidbar ist!

□

Bemerkung

Man kann entscheidbar auf unentscheidbar reduzieren, aber nicht umgekehrt (würde Teil 2 des Lemma verletzen).

Lemma

1. Seien $A \leq^{mo} B$ und B rekursiv aufzählbar, dann ist auch A rekursiv aufzählbar.
2. Seien $A \leq^{mo} B$ und A nicht rekursiv aufzählbar, dann ist auch B nicht rekursiv aufzählbar.

(Gleiche Argumentation wie bei entscheidbar/unentscheidbar)

Intuition

$A \leq^{mo} B$ bedeutet, dass A auf B reduziert werden kann (im Sinne, dass zurückführen). „ \leq^{mo} “ zeigt bereits an, dass B ein viel größeres, schwereres Problem ist als A , und mittels der berechenbaren Funktion f kann man nun die Problemstellung von A in eine Problemstellung von B „übersetzen“. Da B entscheidbar ist, hat man so direkt die Entscheidbarkeit auch für A .

Even \leq^{mo} Odd (als Beispiel)

	Even	Odd
Ja-Instanz	n ist gerade	n' ist ungerade
Nein-Instanz	n ist ungerade	n' ist gerade

Angenommen Odd ist sehr schwer, und man hat einen Algorithmus zum entscheiden von Odd gefunden, und möchte nun Even entscheiden.

Naive Idee

n an den Algorithmus für Odd geben, bloß ist das Resultat immer verkehrt herum (wenn n gerade war (Ja-Instanz von Even), dann gibt Odd falsch (oder 0) aus, also eine Nein-Instanz von Odd, und umgekehrt).

Wegen $x \in A \Leftrightarrow f(x) \in B$ soll aber jede Ja-Instanz zu einer Ja-Instanz führen, und jede Nein-Instanz zu einer Nein-Instanz.

Als „Übersetzungsfunction“ wählt man $f(n) = n + 1$. Wenn dann n gerade ist, wird an Odd jedoch $n + 1$ übergeben, also eine ungerade Zahl, und Odd gibt Wahr (bzw. 1) zurück, also $n \in A \checkmark$.

Reduktion beim Halteproblem

Beispiel zu $a^n b^n$

Reduktion von Subset-Sum auf Lösen eines LGS

W | F 27 W | F 28 W | F 29

6.10 Satz von Rice

Ziel

Finde zahlreiche weitere unentscheidbare Probleme

Struktur all dieser Probleme:

Eingabe: Code C_M einer TM

Frage: Hat $L(M)$ eine bestimmte Eigenschaft?

Sei \mathcal{RE} die Menge der rekursiv aufzählbaren Sprachen

Definition

Eine Eigenschaft E von \mathcal{RE} ist eine Teilmenge von \mathcal{RE}

$$E \subseteq \mathcal{RE}$$

Beispiele für Eigenschaften

Satz von Rice

Sei $E \subseteq \mathcal{RE}$ eine nicht triviale Eigenschaft. Dann ist folgendes EP C_E unentscheidbar:

Eingabe: Code C_M von TM M

Frage: Gilt $L(M) \in E$?

Beweis

Ziel

Reduktion von H auf C_E . Dann folgt Unentscheidbarkeit von C_E .

O.E. nehmen wir $\emptyset \notin E$ an. (Sonst betrachte statt E die Eigenschaft \overline{E} . Mit C_E ist auch $C_{\overline{E}}$ entscheidbar). Sei $A \in E$ eine feste Sprache und sei M_A eine TM mit $L(M_A) = A$.

Wir reduzieren H auf C_E :

$z = (C_M, w)$ (Eingabe auf H)

$\rightarrow f(z) = \text{Code } C_{M_z} \text{ einer TM } M_z \text{ mit folgenden Verhalten: } U(z) \text{ hält} \Leftrightarrow L(M_z) \in E$

Sei $z = (C_M, w)$ eine Eingabe für das Halteproblem. Wir beschreiben die Arbeitsweise der TM M_z , deren Code das Ergebnis der Reduktion $f(z)$ sein soll:

M_z ist eine TM, die auf Eingabe y so arbeitet:

(1) M_z simuliert die universelle TM U auf z (unabhängig von y)

(2) falls $U(z)$ hält, dann simuliert M_z die TM M_A auf y

Die Konstruktion liefert eine Reduktion von H auf C_E (überprüfe ob gilt $\forall z \in \Sigma^* : z \in H \Leftrightarrow f(z) \in C_E$):

$z \notin H$: U hält nicht auf z , somit tritt das Verhalten von (2) nie in Kraft und M_z hält auf keinem y .

Also folgt: $L(M_z) = \emptyset \notin E$ (auch Nein-Instanz von E)

$z \in H$: M_z beendet auf jeder Eingabe von y den Teil (1) seiner Rechnung (U hält auf z) und arbeitet danach wie M_A auf y .

Also folgt: $L(M_z) = L(M_A) = A \in E$ (auch Ja-Instanz von E)

Somit ist die Eigenschaft einer Reduktion bewiesen und C_E ist unentscheidbar. \square

Bemerkung

Implizit verwenden wir oft folgendes Argument:

Definition

Für $A \subseteq B \subseteq \Sigma^*$ nennen wir A entscheidbar in B , wenn gilt:

Es gibt eine Funktion

$$f : B \rightarrow \{0, 1\}$$

die für Eingaben aus B berechenbar und total ist und

$$f(x) = \begin{cases} 1, & x \in A \\ 0, & x \in B \setminus A \end{cases}$$

Dann gilt: Wenn B entscheidbar und A entscheidbar in B , dann ist A entscheidbar (für eine Eingabe $x \in \Sigma^*$ entscheiden wir zunächst, ob $x \in B$. Falls nein, dann folgt $x \notin A$. Falls ja, dann berechne $f(x)$).

Beispiel: Problem entscheidbar? Satz von Rice anwendbar?

W | F 30 W | F 31

6.11 Varianten von TMs

6.11.1 k-Band-TMs

Definition

Sei $k \in \mathbb{N}$. Eine k -Band-TM M ist eine 6-Tupel $(K, \Sigma, \Gamma, \delta, q_0, F)$, wobei $K, \Sigma, \Gamma, q_0, F$ wie bei einer 1-Band-TM definiert sind. δ hingegen ist eine Transitionsfunktion, die präzisiert, dass M auf k Bändern unabhängig voneinander arbeiten kann, d.h. M hat auch k viele Lese-/Schreibköpfe.

Start-, Schlusskonfiguration, Rechnung sowie Resultat sind analog zu einer 1-Band-TM definiert.

Lemma

Eine 1-Band-TM kann alle Funktionen berechnen, die auch eine k -Band-TM berechnen kann.

Beweis

Eine TM kann die k Bändern simulieren, indem sie die k vielen, durch Sonderzeichen getrennten, Bereiche auf dem einen Band reserviert und dann die Operationen der k vielen Lese/-Schreibköpfe nacheinander simuliert. \square

Bemerkung

k-Band-TMs spielen eine gewisse Rolle in der Komplexitätstheorie, da sie die Eingaben der Größe n auf dem ersten Band speichern, aber auf den anderen Bändern für eine Rechnung weniger Platz als n benötigen (z.B. bei der Klasse LOGSPACE).

6.11.2 Nichtdeterministische TMs

Definition

Eine nichtdeterministische TM (NDTM) M ist eine 6-Tupel $(K, \Sigma, \Gamma, s, \Delta, F)$, wobei K, Σ, Γ, s, F wie bei einer deterministischen TM definiert sind. Δ hingegen ist nun eine Übergangsrelation.

Außerdem ändert sich die Definition von $L(M)$:

M akzeptiert $w \Leftrightarrow$ es gibt mindestens eine akzeptierende Rechnung

Lemma

Zu jeder nichtdeterministischen TM M gibt es eine deterministische TM \tilde{M} mit $L(M) = L(\tilde{M})$

Beweis

Für Eingabe w simuliert \tilde{M} für $t = 1, 2, 3, \dots$ alle möglichen Rechnungen von M mit $\leq t$ Schritten. \square

Bemerkung

Nichtdeterministische TMs spielen eine fundamentale Rolle in der Komplexitätstheorie!

6.11.3 Linear-beschränkte TMs

Definition

Eine nichtdeterministische TM M heißt linear-beschränkt, wenn sie für die Eingabe w mit Länge $n := |w|$ nur auf $a \cdot n$ viele Zellen auf dem Band arbeitet (a muss konstant sein!).

Um dies zu gewährleisten, werden zu Beginn beide Enden der Eingabe mit Sonderzeichen aus Γ markiert, und die TM darf diese Sonderzeichen nicht überschreiten oder verschieben.

6.12 Wortproblem für linear-beschränkte TMs

Gegeben: nichtdeterministische, linear-beschränkte TM M , Eingabe $w \in \Sigma^*$

Frage: Ist $w \in L(M)$?

Algorithmus: Für die Eingabe w ist die Anzahl an verfügbaren Zellen für eine Rechnung von M mit $a \cdot n$ (a konstant) fest, und endlich. Wegen der endlich vielen Zellen (und K, δ endlich) gibt es auch nur endlich viele Konfigurationen, die M annehmen kann. Somit kann es auch nur endlich viele Rechnungen geben, welche alle simuliert werden können.

Damit ist das Wortproblem entscheidbar.

6.13 Kontextsensitive Grammatiken

Definition

Eine Grammatik G heißt kontextsensitiv, wenn jede Regel die Form $A \rightarrow B$ hat, wobei $A \in (V \cup \Sigma)^*$, $B \in (V \cup \Sigma)^+$ und $|A| \leq |B|$, d.h. Regeln sind nicht verkürzend.

Lemma

Die kontextsensitiven Sprachen sind genau die, die von nichtdeterministischen, linear-beschränkten TMs akzeptiert werden.

Beispiel zu 6.13 Kontextsensitive Grammatiken

6.14 Wortproblem für kontextsensitive Grammatiken

Gegeben: Kontextsensitive Grammatik G , Eingabe $w \in \Sigma^*$

Frage: Ist $w \in L(G)$?

Variante 1: Untersuche alle möglichen Ableitungen mit einer (berechnbaren) Höchstzahl an Regelanwendungen, um in G ein Wort der Länge $n := |w|$ zu erzeugen.

Wegen $|A| \leq |B|$ muss jede weitere Ableitung entweder ein Zeichen hinzufügen (1 Schritt näher zur Lösung), oder es erfolgt eine gleich-lange Ersetzung. Da V, Σ, R endlich sind, gibt es nur endlich viele solcher gleich-langer Ersetzungen, womit man alle diese Fälle durchgehen kann, und sich dabei die jeweils ersetzenen Strings merkt, um Zyklen zu erkennen.

Insgesamt ist so das Wortproblem entscheidbar.

Variante 2: Simuliere w auf der nichtdeterministischen, linear-beschränkten TM, die G akzeptiert.

6.15 Sprachen $L \notin RE \wedge L \notin co - RE$

$(\{1\}.H_1 \cup \{0\}.\overline{H_1})$

Beweis

$H_1 \times \overline{H_1}$

Beweis

6.16 Abschlusseigenschaften von RE

Abschluss

- Vereinigung: auf beiden TMs parallel semi-entscheiden, und eine müssen halten, damit $w \in L_1 \cup L_2$ (oder beide Aufzähler nacheinander aufzählen lassen)
- Schnitt: auf beiden TMs parallel semi-entscheiden, und beide müssen halten, damit $w \in L_1 \cap L_2$
- Konkatenation: Aufzähler-Funktion f' mit $f'(k) = f(i, j) = f_1(i).f_2(j)$ und $\mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$ mittels **Cantorsche Paarungsfunktion**
- Kleene-Stern: auf allen möglichen Zerlegungen von w in w_1, \dots, w_n parallel semi-entscheiden, und mindestens eine muss halten

- Homomorphismus: Aufzähler-Funktion $f'(i) = h(f(i))$
- Inverse Homomorphismus: $h(x)$ berechnen und dann $h(x) \in L(M)$ semi-entscheiden

Kein Abschluss

- Komplement: $H \in RE$, aber $\overline{H} \notin RE$ (**Satz von Turing**)
- Differenz: bereits nicht unter Komplement, was Differenz von Σ^* entspricht

6.17 Abschlusseigenschaften von REC

Abschluss

- Vereinigung: in beiden entscheiden, und gibt eine „Ja“ zurück
- Schnitt: in beiden entscheiden, und geben beide „Ja“ zurück
- Konkatenation: nichtdeterministisch Zerlegung wählen, und dann beide Wörter jeweils in L_1 und L_2 entscheiden
- Kleene-Stern: nichtdeterministisch Zerlegung von w in w_1, \dots, w_n wählen, und dann alle w_i in L_1 entscheiden
- Komplement: Definition von **Entscheidbarkeit**
- Differenz: in beiden entscheiden, und erste gibt „Ja“ und zweite „Nein“ zurück
- Inverse Homomorphismus: $h(x)$ berechnen und dann $h(x) \in L(M)$ entscheiden

Kein Abschluss

- Homomorphismus:
 - $L = \{(C_M, w, t) \mid C_M \text{ ist Code der TM } M \text{ und } M \text{ hält auf } w \text{ innerhalb von } t \text{ Schritten}\} \in REC$, mit t unär kodiert ($t \rightarrow 1^n$)
 - Homomorphismus $h : \Sigma^* \rightarrow \Sigma^*$ und $h(w) = w \quad \forall w \in \Sigma \setminus \{1\}$, $h(1) = \lambda$
 - $\rightarrow h(L) = H$ (Halteproblem) und $H \notin REC$ (**Satz von Turing**) (C_M, w ohne 1 kodiert)

6.18 Chomsky-Hierarchie

Die Chomsky-Hierarchie ist eine Klassifikation von formalen Sprachen:

Klasse	Automat	Grammatik	Chomsky-Hierarchie
regulär	(N)DEA	rechts-linear	CH-3
kontextfrei	KA	kfr. Grammatik	CH-2
kontext-sensitiv	lin.-beschr. TM	kontext-sensitiv	CH-1
rek. aufzählbar	TM	allg. Grammatik	CH-0

Kapitel 7 Komplexitätstheorie

7.1 Komplexitätsklasse P

Bisher lauteten unsere Fragen, ob eine Sprache / eine Funktion überhaupt entscheidbar / berechenbar ist. Jetzt betrachten wir ausschließlich entscheidbare Sprachen bzw. berechenbare Funktionen und wollen mehr über die Ressourcen wissen, die zugehörige Algorithmen benötigen. Zentral ist dabei:

- Welche Ressourcen interessieren?
 - Laufzeit ($\#Schritte$)
 - Platzverbrauch
- Angabe eines konkreten Algorithmus für ein Problem und seine Analyse
 - obere Schranken
- Welche Ressourcen benötigt prinzipiell jeder Algorithmus für ein Problem?
 - untere Schranken

Ist die obere Schranke \approx die untere Schranke eines Algorithmus, so ist der Algorithmus optimal.

Definition

- a) Sei M TM, die eine totale Funktion $f_M : \Sigma^* \rightarrow \Sigma^*$ berechnet und sei $t : \mathbb{N} \rightarrow \mathbb{N}$. Wir sagen, dass M eine durch t beschränkte Laufzeit hat, wenn $\forall w \in \Sigma^*$ das Ergebnis $f_M(w)$ in $\leq t(|w|)$ Schritten berechnet wird.
- b) M läuft in Polynomialzeit, wenn M durch ein Polynom laufzeitbeschränkt ist.

$$p \text{ Polynom: } p : \mathbb{R} \rightarrow \mathbb{R}, \forall x \in \mathbb{R} : p(x) = \sum_{i=0}^d a_i x^i$$

mit $d \in \mathbb{N}$ Grad, $a_i \in \mathbb{R}$, $a_d \neq 0$

- c) Ein $L \subseteq \Sigma^*$ gehört zur Klasse P (in Polynomialzeit entscheidbar), wenn ihre charakteristische Funktion χ_L in Polynomzeit berechenbar ist.

Beispiel zu 7.1 Komplexitätsklasse P

Lemma

- a) Die Klasse P ist abgeschlossen unter Vereinigung, Schnitt, Komplement, Konkatenation.
- b) Die Komposition zweier in Polynomialzeit berechenbarer Funktionen ist wieder in Polynomialzeit berechenbar.

Beweis für b)

Seien die Funktionen f_1, f_2 durch die TMs M_1, M_2 in Zeit p_1, p_2 berechenbar.

Dann ist die Komposition $f_2 \circ f_1$ bzw. $f_2(f_1(x))$ in Zeit $p_2(p_1(|x|) + |x|)$ berechenbar, was durch einen Polynom mit Grad $d_1 \cdot d_2$ beschränkt ist. \square

Für viele wichtige Entscheidungsprobleme weiß man nicht ob sie in P liegen. Viele davon haben eine spezielle Struktur, die zu einer weiteren Klasse NP führt.

W | F 34 **W | F 35**

Hamiltonkreis

7.2 Erfüllbarkeitsproblem

Sei $X := \{x_1, x_2, \dots\}$ Menge von Booleschen Variablen, d.h. jedes x_i kann die Werte 0 und 1 annehmen ($0 \stackrel{\Delta}{=} \text{falsch}, 1 \stackrel{\Delta}{=} \text{wahr}$).

- **Literal:** x_i oder $\overline{x_i}$ (Negation von x_i)
- **Negation von x_i :** $\overline{x_i}$ (bzw. $\neg x_i$), d.h. $\overline{0} = 1$ und $\overline{1} = 0$.
- **Klausel C :** Disjunktion von Literalen, z.B. $x_1 \vee \overline{x_2} \vee x_5 \vee \overline{x_7}$.
- **boolesche Funktion ϕ in CNF:** Konjunktion von endlich vielen Klauseln: $C_1 \wedge \dots \wedge C_m$,

z.B. $\phi(x_1, x_2, x_3, x_4) \equiv (\overline{x_1} \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3} \vee \overline{x_4}) \wedge x_4$.

Beispiel zu 7.2 Erfüllbarkeitsproblem

Bemerkung

ϕ stellt eine boolesche Funktion von $\{0,1\}^n$ nach $\{0,1\}$ dar. $x \in \{0,1\}^n$ heißt Belegung und falls $\phi(x) = 1$, dann ist x eine erfüllende Belegung für ϕ .

7.2.1 SAT

Definition

Das Erfüllbarkeitsproblem SAT lautet wie folgt:

Gegeben: boolesche Funktion ϕ in CNF, in $n \in \mathbb{N}$ Variablen.

Frage: Gibt es eine erfüllende Belegung für ϕ , d.h. $\exists x^* \in \{0,1\}^n : \phi(x^*) = 1$?

Dann heißt ϕ erfüllbar.

Variante SAT₁:

Gegeben: ϕ in CNF in n Variablen, $x^* \in \{0,1\}^n$.

Frage: Gilt $\phi(x^*) = 1$?

Behauptung: **SAT₁ ∈ P**

Eingabegröße: Was ist ca. die Eingabegröße einer Instanz (ϕ, x^*) von **SAT₁**?

→ Kodierungslänge für ϕ : $\mathcal{O}(n \cdot m \cdot \log(n))$

(ϕ hat m Klauseln, und jede Klausel hat maximal n viele Literale (Doppelte ergeben Tautologie und können gekürzt werden). Für die Kodierung eines Literalen werden noch $\log(n)$ benötigt.)

→ Kodierungslänge für x^* : $\mathcal{O}(n)$

Algorithmus: Gehe nacheinander durch jede Klausel und setze die Belegung, um zu überprüfen, ob x^* die Klausel erfüllt.

↔ polynomiell in $|(\phi, x^*)|$

Entscheidungsalgorithmus für SAT:

→ Zähle $\{0, 1\}^n$ auf und wende für jede Belegung den Algorithmus für **SAT₁** an.

↔ Im Worst-Case benötigt dieser Algorithmus $\geq 2^n$ Schritte → exponentielle Laufzeit.

SAT ∈ P ?

7.2.2 k-SAT

Definition

Sei $k \in \mathbb{N}$.

Das k-SAT-Problem ist folgende Variante von SAT:

Gegeben: boolesche Funktion ϕ in CNF, in $n \in \mathbb{N}$ Variablen. Jede Klausel hat maximal k Literale.

Frage: Ist ϕ erfüllbar?

Beispiel zu 7.2.2 k-SAT

7.3 Komplexitätsklasse NP

Strukturelle Gemeinsamkeiten zwischen HK/HK₁ und SAT/SAT₁

Zu einer Probleminstanz fragen wir nach der Existenz eines Objekts, das bzgl. der Instanz eine gewisse Eigenschaft erfüllt. Dabei ist der Suchraum für diese Objekte exponentiell groß in der Eingabegröße. Hat man ein konkretes solches Objekt gegeben, dann kann man effizient prüfen, ob es die gewünschte Eigenschaft hat

Definition NP erste Variante

$L \subseteq \Sigma^*$ liegt in NP (L ist in nicht-deterministischer Polynomzeit verifizierbar), wenn es eine NDTM M und ein Polynom p gibt mit:

- i) M hält immer mit Resultat 0 oder 1
- ii) $\forall x \notin L$ gibt M immer 0 als Resultat aus
- iii) $\forall x \in L$ gibt es mindestens eine Rechnung mit Resultat 1

iv) Alle Rechnungen von M auf $x \in \Sigma^n$ sind in $\leq p(n)$ Schritten beendet

$HK \in NP$ und $k-SAT \in NP$

Bemerkung

- $P \subseteq NP$

Wähle als NDTM in der Definition von NP die deterministische TM, die ein $L \in P$ entscheidet.

- $L \in NP \implies L$ ist entscheidbar

Jedes $L \in NP$ ist entscheidbar, da ein det. Entscheidungsverfahren alle Rechnungen von M in der Definition von $L \in NP$ simulieren kann.

Definition NP zweite Variante

Ein Problem L liegt in NP, wenn es ein Problem $L_1 \in P$ und ein Polynom p gibt mit:

$$L := \{x \in \Sigma^* \mid \exists y \in \Sigma^* \text{ mit } |y| \leq p(|x|) \wedge (x, y) \in L_1\}$$

Ein passendes y für Eingabe x kann als Beweis bzw. Zertifikat für $x \in L$ gedeutet werden. Es darf nicht zu lang sein.

Lemma

Beide Definitionen liefern dieselbe Komplexitätsklasse

Beweis

(1) \implies (2) Sei $L \in NP^{(1)}$ nach erster Variante, M die zugehörige NDTM mit Laufzeit p .

Dann entspricht y in Variante (2) einem Rechenweg von M . $(x, y) \in L_1$ heißt dann, M akzeptiert x entlang dem Rechenweg $y \rightsquigarrow$ geht in $p(n)$ Schritten.

Es folgt also $L_1 \in P$ und L erfüllt auch die 2. Version der Definition von NP.

(2) \implies (1) Sei umgekehrt $L \in NP^{(2)}$ nach 2. Version, L_1 und p wie in der Definition.

Eine NDTM für L rät nicht-deterministisch ein y und prüft danach deterministisch mit dem Algorithmus für L_1 , ob $(x, y) \in L_1$. □

Es wird vermutet, dass $P \not\subseteq NP$ gilt. Auch wenn das unbewiesen ist, kann man einige Probleme identifizieren, die die Komplexität der Klasse NP repräsentieren.

W | F 36

W | F 37

W | F 38

7.4 NP-vollständige Probleme, Polynomzeit-Reduktion

Definition

a) Seien $L_1, L_2 \subseteq \Sigma^*$

L_1 heißt *in Polynomzeit many-one reduzierbar* auf L_2 , wenn es eine totale berechenbare Funktion $f : \Sigma^* \rightarrow \Sigma^*$ und einen Polynom p gibt, mit:

$$\forall x \in \Sigma^* : x \in L_1 \Leftrightarrow f(x) \in L_2$$

und die TM, die f berechnet, benötigt für die Rechnung auf jeden $x \leq p(|x|)$ Schritte

Notation: $L_1 \preceq_p^{mo} L_2$

b) L heißt NP-schwer (oder NP-hart), wenn $\forall A \in NP : A \preceq_p^{mo} L$

(d.h. L ist mindestens so schwer wie jedes Problem in NP , aber muss nicht in NP liegen, z.B. unentscheidbare Probleme)

c) L heißt NP-vollständig, wenn $L \in NP$ und L NP-schwer ist

d) NPC ist die Menge aller NP-vollständigen Probleme.

Polynomialzeit Reduktion von Subset Sum auf Bin-Packung

Lemma

Seien $L_1, L_2, L_3 \subseteq \Sigma^*$

a) Falls $L_1 \preceq_p^{mo} L_2$ und $L_2 \in P$, dann $L_1 \in P$

Analog für NP

b) Sei $L_2 \in NP$, $L_1 \preceq_p^{mo} L_2$, $L_1 \in NPC$, dann folgt $L_2 \in NPC$

c) $P = NP \Leftrightarrow \exists L \in NPC \cap P$

Beweis

a) Zum Entscheiden von $x \in L_1$ berechne Reduktion f (in Polynomzeit möglich nach Definition von \preceq_p^{mo}) und entscheide dann ob $f(x) \in L_2$ (auch in Polynomzeit möglich, da $L_2 \in P$)

b) \preceq_p^{mo} ist transitiv:

$$L_1 \in NPC \implies \forall A \in NP : A \preceq_p^{mo} L_1$$

$$\forall A \in NP : A \preceq_p^{mo} L_1 \preceq_p^{mo} L_2 \implies \forall A \in NP : A \preceq_p^{mo} L_2 \implies L_2 \in NPC \quad \checkmark$$

c)

„ \Leftarrow “ $L \in NPC \implies \forall A \in NP : A \preceq_p^{mo} L$ (Definition von NP-vollständig)

$$\forall A \in NP : \underbrace{A \preceq_p^{mo} L}_{\text{Polynomzeit}} \wedge \underbrace{L \in P}_{\text{Polynomzeit determ.}} \implies \underbrace{A \in P}_{\text{Polynomzeit determ.}}$$

$$\forall A \in NP : A \in P \implies NP \subseteq P$$

$$NP \subseteq P \wedge P \subseteq NP \implies P = NP$$

„ \Rightarrow “ $P = NP \implies \forall L \in NP : L \in P \implies \forall L \in NPC : L \in P (L \in NPC \implies L \in NP)$
 $\implies NPC = P \setminus \{\emptyset, \Sigma^*\} (\emptyset, \Sigma^* \notin NPC) \implies \exists L \in NPC \cap P (NPC \neq \emptyset)$

□

Bemerkung

Die trivialen Sprache \emptyset und Σ^* sind nicht NP-vollständig ($\emptyset \notin NPC$ und $\Sigma^* \notin NPC$)

Beweis für \emptyset und Σ^* sind nicht NP-vollständig

7.4.1 Beispiele

Beispiel 1 Beispiel 2 Beispiel 3

7.4.2 Existenz von NP-vollständigen Problemen

→ Konstruktion eines NPC -Problems, indem man das Halteproblem variiert:

Eingabe: Code C_M einer NDTM M , $x \in \Sigma^*, t \in \mathbb{N}$ in unärer Kodierung

Frage: Akzeptiert $C_M x$ in $\leq t$ Schritten?

Behauptung: Das Problem ist NP-vollständig

Beweis:

$L \in NP$

Der nicht-det. Algorithmus rät einen Rechenweg von M und simuliert M auf x für t Schritte entlang dieses Weges und gibt dieselbe Antwort wie M . Die Simulation dauert t Schritte, was wegen der unären Kodierung polynomiell ist. □

$\forall A \in NP : A \preceq_p^{mo} L$

Sei $L \in NP$ mit zugehöriger NDTM M_L , deren Laufzeit durch den Polynom p beschränkt ist, und x die Eingabe für L .

Die Reduktion f berechnet aus x die Instanz $(C_{M_L}, x, p(|x|))$.

f ist polynomieller Zeit berechenbar, denn C_M ist für alle x konstant und kann daher in konstanter Zeit berechnet werden. x ist außerdem schon als Eingabe für f vorhanden und den Wert $p(|x|)$ kann man in der Tat in polynomiell vielen Schritten bzgl. $|x|$ berechnen. □

W | F 39

W | F 40

W | F 41

7.5 Satz von Cook, Levin

Satz von Cook, Levin 1971

Das Problem 3 – SAT ist NP-vollständig.

Beweis

(Wir zeigen nur $SAT \in NPC$, $SAT \in NP$ siehe **oben**, und eine \preceq_p^{mo} -Reduktion von SAT auf $3-SAT$ ist relativ einfach.)

Sei $L \in NP$ und M eine det. TM, die $L \in NP$ belegt, d.h. M arbeitet auf Paaren (x, y) :

- i) $\forall x, y : M(x, y) \in \{0, 1\}$
- ii) $\forall x \in L \exists y : M(x, y) = 1$
- iii) $\forall x \notin L \forall y : M(x, y) = 0$
- iv) $\forall x, y$ ist die Laufzeit $T_M(x, y) \leq p(|x|)$ (p Polynom)

O.E. habe M einen akzeptierenden Endzustand q_+ und einen verwerfenden q_- .

Aufgabe

Konstruieren für jedes $w \in \Sigma^*$ in Polynomzeit in $|w|$ eine SAT -Formel $f(w)$ so, dass es für M eine akzeptierende Rechnung auf w genau dann gibt, wenn $f(w)$ erfüllbar ist. Dazu müssen Rechnungen von M aussagenlogisch beschrieben werden.

Sei $w \in \Sigma^*$ fest, $|w| =: n$

M verwendet bei Rechnung auf (w, y) maximal die Zeilen mit Nummern $-p(n), -p(n) + 1, \dots, 0, 1, \dots, p(n)$ (M ist durch $p(n)$ beschränkt, kann also von Zelle 0 mit seinen $p(n)$ vielen Schritten maximal $p(n)$ nach links oder rechts):

$-p(n)$	-1	0	1	$n-1$	$p(n)$
	\dots		w_1	w_2	\dots
			\uparrow s		w_n

maximaler Arbeitsbereich von M

Wir schauen uns jetzt ein Tableau mit $p(n) + 1$ vielen Spalten (Schritt 0 bis letzter Schritt $p(n)$) und $2p(n) + 1$ Spalten (die nummerierten Zellen) an. Im Idealfall repräsentiert die Zeile für $0 \leq t \leq p(n)$ die korrekte Konfiguration von M nach t Schritten bei Rechnung auf (w, y) .

Sei $K = \{q_1, \dots, q_l\}$ die Menge der Zustände von M , $\Gamma = \{a_1, \dots, a_m\}$ das Bandalphabet. Wir finden nun Gruppen von Booleschen Variablen und geben ihre gewünschte Interpretation bzgl. M an:

- 1) Für jede Zeile i mit $-p(n) \leq i \leq p(n)$, jedes $a \in \Gamma$ und jede Schrittzahl $0 \leq t \leq p(n)$ verwenden wir eine Variable $C_{i,a,t}$ (\sim Cell) mit der Interpretation:

$$C_{i,a,t} = 1 \Leftrightarrow \text{Zelle } i \text{ hat zur Zeit } t \text{ Eintrag } a$$

$$\rightarrow \# \text{ Variablen: } (2p(n) + 1) \cdot (p(n) + 1) \cdot |\Gamma|$$

- 2) Für $0 \leq t \leq p(n)$ und $\forall q \in K$ eine Variable $s_{q,t}$ (\sim State) mit Interpretation:

$$s_{q,t} = 1 \Leftrightarrow \text{Zur Zeit } t \text{ befindet sich } M \text{ im Zustand } q$$

$$\rightarrow \# \text{ Variablen: } (p(n) + 1) \cdot |K|$$

- 3) Für $0 \leq t \leq p(n)$ und $-p(n) \leq i \leq p(n)$ Variablen $h_{i,t}$ (\sim Head) mit:

$$h_{i,t} = 1 \Leftrightarrow \text{Zur Zeit } t \text{ steht der Kopf auf Zelle } i$$

$$\rightarrow \# \text{ Variablen: } (2p(n) + 1) \cdot (p(n) + 1)$$

Wir wollen jetzt aussagenlogische Formeln in diesen (und weiteren) Variablen so aufstellen, dass die Erfüllbarkeit der Formeln etwas über Rechnungen von M aussagt. Für diese Formeln gibt es zunächst grundsätzliche Bedingungen, z.B. darf zu jedem Zeitpunkt nur ein (und genau ein) Buchstabe in jeder Zeile stehen.

Zu booleschen Variablen x_1, \dots, x_r definiere:

$$U(x_1, \dots, x_r) \equiv (x_1 \vee \dots \vee x_r) \wedge \bigwedge_{1 \leq i < j \leq r} (\neg x_i \vee \neg x_j)$$

Dann gilt:

$$U(x_1, \dots, x_r) = 1 \Leftrightarrow \text{genau ein } x_i \text{ hat den Wert 1}$$

Wir fügen der Konstruktion folgende Formeln hinzu:

$$\begin{aligned} U(h_{-p(n),t}, h_{-p(n)+1,t}, \dots, h_{p(n),t}) &\quad \forall 0 \leq t \leq p(n) \\ U(s_{q_1,t}, \dots, s_{q_l,t}) &\quad \forall 0 \leq t \leq p(n) \\ U(C_{i,a_1,t}, \dots, C_{i,a_n,t}) &\quad \forall 0 \leq t \leq p(n), -p(n) \leq i \leq p(n) \end{aligned}$$

(Diese werden mit *UND* verknüpft)

Jetzt drücken wir korrekte Rechenschritte von M aus. Definiere Formeln $D_t, 0 \leq t \leq p(n) - 1$

$$D_t := \bigwedge_{\substack{-p(n) \leq i \leq p(n), \\ a \in \Gamma}} [h_{i,t} \vee (C_{i,a,t} \Leftrightarrow C_{i,a,t+1})]$$

D_t erfüllt g.d.w. alle Zellen auf denen der Kopf steht, im nächsten Schritt unverändert bleiben. Für den Fall, dass $h_{i,t} = 1$, muss Zelle i entsprechend angepasst werden. Wir bilden:

$$E_t = \bigwedge_{\substack{-p(n) \leq i \leq p(n), \\ a \in \Gamma, q \in K}} \underbrace{E_{i,a,q,t}}_{(1)} \vee \underbrace{E_{i,a,q,t}}_{(2)}$$

$$(1) E_{i,a,q,t} \equiv (\neg C_{i,a,t}) \vee (\neg h_{i,t}) \vee (\neg s_{q,t})$$

$$(2) E_{i,a,q,t} \equiv s_{p,t+1} \wedge C_{i,b,t+a} \wedge h_{i+\epsilon_\alpha,t+1}$$

für $\delta(q, a) = (p, b, \alpha)$ Befehl in M mit $\alpha \in \{R, L, N\}, \epsilon_\alpha \in \{1, -1, 0\}$

D.h. $\neg(1)$ g.d.w. der Kopf in Zustand q auf Zelle i steht und ein a liest. $E(2)$ ist erfüllt g.d.w. der Schritt von M korrekt beschrieben ist.

Wir bilden die Konjunktion aller dieser D_t, E_t und U -Formeln für die nötigen Parameterwahlen. Diese Konjunktion ist erfüllbar g.d.w. die betreffende Variablenbelegung eine korrekte Rechnung von M beschreibt. Es fehlt noch eine korrekte Startkonfiguration für w

$$B \equiv s_{q_1,0} \wedge h_{1,0} \wedge C_{1,w_1,0} \wedge \dots \wedge C_{n,w_n,0} \wedge \bigwedge_{-p(n) \leq i \leq n} C_{i,\square,0} \wedge \bigwedge_{n+1 \leq i \leq p(n)} \text{Zelle } i \text{ enthält } a \in \Sigma$$

Sowie am Ende: $s_{q_+,p(n)}$ mit $q_+ :=$ einziger akzeptierender Endzustand

Insgesamt erhalten wir eine Formel:

$$\phi(\text{ alle Variablen }) \equiv B \wedge \bigwedge_t D_t \wedge E_t \wedge U_t \wedge s_{q_+,p(n)}$$

ϕ erfüllbar g.d.w. M eine akzeptierende Rechnung auf w hat

ϕ hat polynomiale Länge in $n = |w|$

ϕ hat noch nicht KNF, z.B. $D_t, E_t \rightarrow$ Wir bauen die Formel D_t um

$$\begin{aligned} & x \vee (y \Leftrightarrow z) \\ \Leftrightarrow & x \vee (\neg y \vee z \wedge \neg z \vee y) \\ \Leftrightarrow & (x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \end{aligned}$$

Analog für E_t : Hier erhalten wir u.U. mehr als 3 Literale

Letzter Schritt: Erzeuge $3-SAT$ -Formeln

$$k = x_1 \vee \dots \vee x_r \quad , r > 3$$

Führe neue Variable y ein mit Bedingung $x_1 \vee x_2 \Leftrightarrow y$

Verwandle diese in KNF-Formel und ersetze zusätzlich K durch $k \equiv y \vee x_3 \vee \dots \vee x_r$

Fahre fort, bis aus k eine Klausel mit 3 Literalen wird. \square

Lemma

Die Probleme $1-SAT$ und $2-SAT$ liegen in P .

Beweis

$2-SAT$ sei $\Phi(x_1, \dots, x_n)$

Setze $x_1 = 1$. Wir setzen x_1 in alle Klauseln, die x_1 oder \bar{x}_1 enthalten.

a) Klausel wird erfüllt \rightarrow Streichen

b) Klausel enthält $\bar{x}_1 \rightarrow$ Belege das zweite Literal mit dem nötigen Wert, um die Klausel zu erfüllen, mache mit dieser Variable analog weiter

Mögliche Fälle:

a) Man findet erfüllende Belegung

b) Man erhält einen Widerspruch \rightarrow Wiederhole alles mit $x_1 = 0$

c) Es bleibt eine kleinere $2-SAT$ Instanz, die nur aus Klauseln von Φ besteht. Jetzt gilt: Φ' erfüllbar g.d.w. Φ erfüllbar und die ersten Variablen können wie bei 1), 2) gewählt werden.

\square

7.6 Asymptotisches Wachstumsverhalten

7.6.1 Landau-Notation

$f = \mathcal{O}(g)$	Bedeutung: f wächst höchstens so schnell wie g Grenzwert: $\limsup_{n \rightarrow \infty} \left \frac{f(n)}{g(n)} \right < \infty$ Quantoren: $\exists C > 0 \exists n_0 \forall n > n_0 : f(n) \leq C \cdot g(n) $
$f = o(g)$	Bedeutung: f wächst langsamer als g Grenzwert: $\lim_{n \rightarrow \infty} \left \frac{f(n)}{g(n)} \right = 0$ Quantoren: $\forall C > 0 \exists n_0 \forall n > n_0 : f(n) < C \cdot g(n) $

$f = \Omega(g)$ $(\Leftrightarrow g = \mathcal{O}(f))$	Bedeutung: f wächst mindestens so schnell wie g Grenzwert: $\liminf_{n \rightarrow \infty} \left \frac{f(n)}{g(n)} \right > 0$ Quantoren: $\exists c > 0 \exists n_0 \forall n > n_0 : c \cdot g(n) \leq f(n) $
$f = \omega(g)$	Bedeutung: f wächst schneller als g Grenzwert: $\lim_{n \rightarrow \infty} \left \frac{f(n)}{g(n)} \right = \infty$ Quantoren: $\forall c > 0 \exists n_0 \forall n > n_0 : c \cdot g(n) < f(n) $
$f = \Theta(g)$	Bedeutung: f wächst genauso schnell wie g Grenzwert: $0 < \liminf_{n \rightarrow \infty} \left \frac{f(n)}{g(n)} \right \leq \limsup_{n \rightarrow \infty} \left \frac{f(n)}{g(n)} \right < \infty$ Quantoren: $\exists C > 0 \exists c > 0 \exists n_0 \forall n > n_0 : c \cdot g(n) \leq f(n) \leq C \cdot g(n) $

7.6.2 Regel von L'Hospital

Definition

$$\lim_{x \rightarrow \infty} \frac{f(x)}{g(x)} = \lim_{x \rightarrow \infty} \frac{f'(x)}{g'(x)}$$

Umformungstabelle:

Grenzwert $\lim_{x \rightarrow a} f(x)$	Funktion $f(x)$	Umformung
$\frac{0}{0}, \frac{\infty}{\infty}$	$\frac{g(x)}{h(x)}$	entfällt
$0 \cdot \infty$	$g(x) \cdot h(x)$	$\frac{g(x)}{\frac{1}{h(x)}}$
$\infty - \infty$	$g(x) - h(x)$	$\frac{\frac{1}{h(x)} - \frac{1}{g(x)}}{\frac{1}{g(x) \cdot h(x)}}$
$0^\infty, \infty^0, 1^\infty$	$g(x)^{h(x)}$	$e^{h(x) \cdot \ln(g(x))}$

7.6.3 Stirling-Formel

$$n! = \mathcal{O} \left(\sqrt{2\pi n} \cdot \left(\frac{n}{e} \right)^n \right) \text{ für } n \rightarrow \infty$$

7.6.4 Logarithmen-Gesetze

Lemma

$$\begin{aligned} \log_a(x \cdot y) &= \log_a(x) + \log_a(y) \\ \log_a\left(\frac{x}{y}\right) &= \log_a(x) - \log_a(y) \end{aligned}$$

$$\begin{aligned}\log_a(x^y) &= y \cdot \log_a(x) \\ \log_a(b) &= \frac{\log_c(b)}{\log_c(a)} \\ \rightarrow \log_c(a) &= \frac{\log_c(b)}{\log_a(b)} \\ \log_a(x) &= \frac{1}{\log_x(a)}\end{aligned}$$

7.6.5 Beispielrechnungen

Beispiel 1 Beispiel 2 Beispiel 3

W | F 42

W | F 43

W | F 44

W | F 45

W | F 46

Kapitel 8 Wahr oder Falsch

Zusätzliche „Wahr oder Falsch“-Fragen, die im Wintersemester 2024/2025 nicht behandelt wurden, aber dennoch interessant/wichtig sind:

[W | F 47](#)

[W | F 48](#)

[W | F 49](#)

[W | F 50](#)

[W | F 51](#)

[W | F 52](#)

[W | F 53](#)

[W | F 54](#)

[W | F 55](#)

[W | F 56](#)

[W | F 57](#)

[W | F 58](#)

[W | F 59](#)

[W | F 60](#)

[W | F 61](#)

[W | F 62](#)

[W | F 63](#)

[W | F 64](#)

[W | F 65](#)

[W | F 66](#)

[W | F 67](#)

[W | F 68](#)

[W | F 69](#)

[W | F 70](#)

Kapitel 9 Methodenblatt ↴

9.1 Definitionen/Sätze/Lemma

Muss man formal korrekt wiedergeben können:

- Reguläre Ausdrücke
- Reguläre Pumping-Eigenschaft
- Reguläres Pumping-Lemma (mit Beweis)
- Rechtskongruenzrelation auf L (\approx_L)
- Rechtskongruenzrelation auf M (\sim_M)
- Satz von Myhill-Nerode (mit Beweis)
- Satz von Kleene
- Satz von Robin & Scott
- Kontextfreie Pumping-Eigenschaft
- Kontextfreies Pumping-Lemma

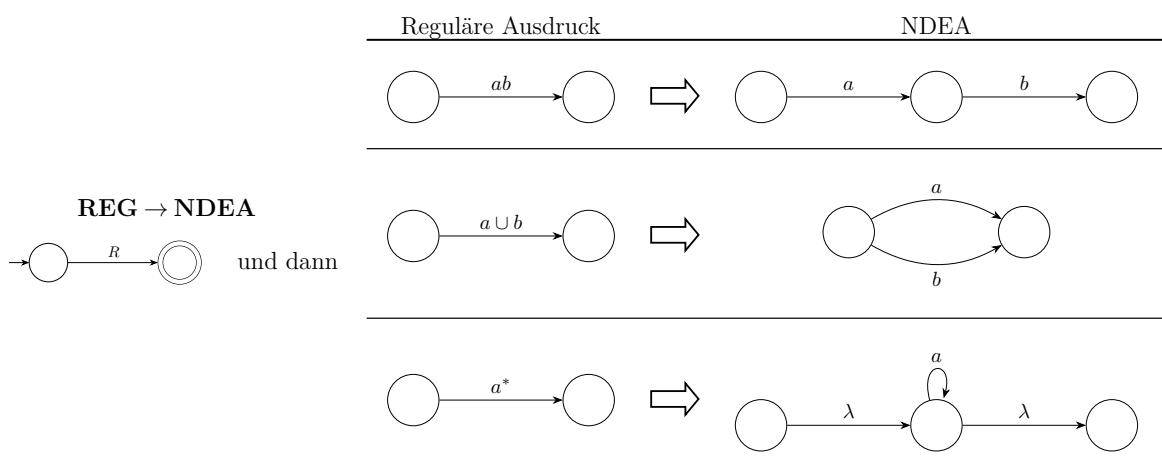
9.2 Reguläre Sprachen (REG)

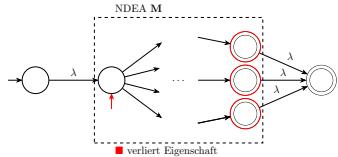
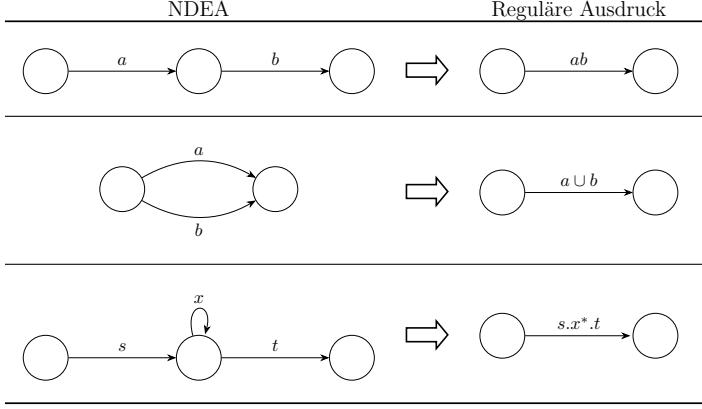
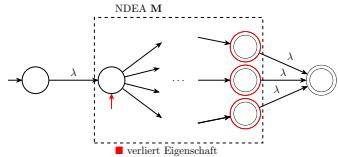
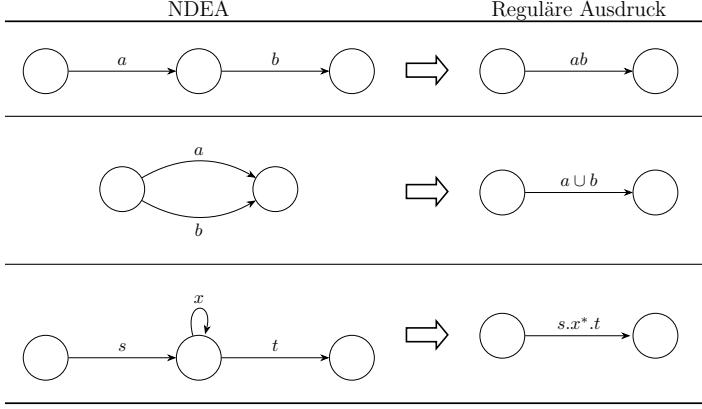
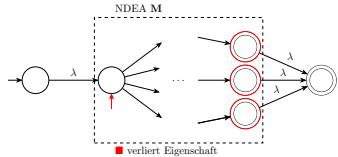
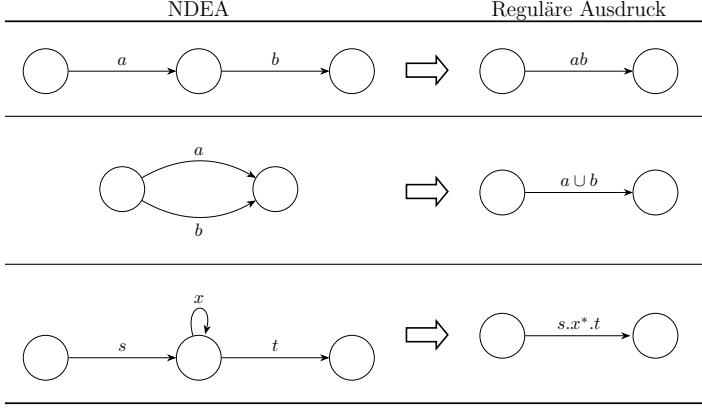
9.2.1 Beweise/Widerlege $L \in REG$

- | | |
|-------------|----------------|
| $L \in REG$ | $L \notin REG$ |
|-------------|----------------|
- Regulären Ausdruck angeben
 - NDEA konstruieren
 - keine reguläre Pumping-Eigenschaft (hat jede reguläre Sprache nach reg. Pumping-Lemma)
 - Unendlich viele Äquivalenzklassen bzgl. \approx_L (reguläre Sprachen haben endlich viele ÄK nach Satz von Myhill-Nerode)

Abschluss: unter Vereinigung, Schnitt, Komplement, Differenz, Konkatenation, Kleene-Stern, Homomorphismus, Inverse Homomorphismus

9.2.2 NDEA \leftrightarrow REG



NDEA	Reguläre Ausdruck
	
	
	

9.2.3 NDEA → DEA

Aus dem NDEA $M = (K, \Sigma, \Delta, s, F)$ konstruiert man den DEA (Potenzautomat) $M' = (K', \Sigma, \delta', s', F')$ mit:

- $K' = \mathcal{P}(K)$ (Potenzmenge)
- $s' = E(s) \in K'$
- $F' = \{Q \subset K \mid Q \cap F \neq \emptyset\}$
- $\delta'(Q, a) := \bigcup_{q \in Q} \{E(p) \mid p \in K \wedge (q, a, p) \in \Delta_M\}$

$E(q) := \{p \in K \mid (q, \lambda) \xrightarrow{M^*} (p, \lambda)\} \cup \{q\}$ (Alle von q mit λ -Übergänge erreichbaren Zustände)

9.2.4 2 DEA → DEA

Zum DEA $M_1 = (K_1, \Sigma, \delta, s_1, F_1)$ mit $L(M_1) = L_1$, und DEA $M_2 = (K_2, \Sigma, \delta, s_2, F_2)$ mit $L(M_2) = L_2$ konstruiert man den DEA (Produktautomat) $M = (K, \Sigma, \delta, s, F)$ mit

- $K = K_1 \times K_2$
- $s = (s_1, s_2)$
- $F = \begin{cases} F_1 \times F_2 & \text{für } L_1 \cap L_2 \\ K_1 \times F_2 \cup K_2 \times F_1 & \text{für } L_1 \cup L_2 \end{cases}$
- $\delta : ((q_1, q_2), a) \rightarrow (\delta_1(q_1, a), \delta_2(q_2, a))$

9.2.5 DEA minimieren

- Nicht erreichbare Zustände entfernen
- \equiv_0 bilden: F und $K \setminus F$
- \equiv_{n+1} bilden: $p \equiv_{n+1} q \Leftrightarrow \begin{array}{l} 1) \quad q \equiv_n p \\ 2) \quad \forall a \in \Sigma : \delta(q, a) \equiv_n \delta(p, a) \end{array}$
- Gilt $\equiv_{n+1} = \equiv_n \rightarrow$ Fertig ($\equiv_n = \equiv_M$)
- Minimalen Zustandsautomaten konstruieren: Jede Klasse ein Zustand, Startzustand wird die Klasse mit dem ehemaligen Startzustand, Endzustände sind die Klassen,

die aus F hervorgeringen, für die Transitionen einen Zustand q^* der Klasse wählen, und $\delta(q^*, a) \forall a \in \Sigma$ betrachten.

9.2.6 ÄK bzgl. $\sim_M \rightarrow \text{REG}$

Zur Äquivalenzklasse $A = [a]$ bestimmt wie folgt den regulären Ausdruck:

- Zu M den minimalen Automaten M' bestimmen
- Bestimme Zustand $q \rightarrow$ Zustand wo die Rechnung von M' auf a endet
- Der reguläre Ausdruck α für A ist dann:

$$\alpha = (\bigcup \{\text{Pfad von } s \text{ zu } q\}) \cdot (\bigcup \{\text{Zyklus von } q \text{ zu } q\})^*$$

9.3 Kontextfreie Sprachen (CFL)

9.3.1 Beweise/Widerlege $L \in CFL$

$L \in CFL$

- kfr. Grammatik angeben
- PDA konstruieren
- regulären Ausdruck angeben ($REG \subset CFL$)
- NDEA konstruieren

$L \notin CFL$

- keine kfr. Pumping-Eigenschaft (hat jede kfr. Sprache nach kfr. Pumping-Lemma)

Abschluss: unter Vereinigung, Konkatenation, Kleene-Stern, Homomorphismus, Inverse Homomorphismus, Schnitt mit einer regulären Sprache

!kein! Abschluss: unter Schnitt, Komplement, Differenz

9.3.2 $CFL \leftrightarrow NDEA$

$CFL \rightarrow NDEA$

- $K := V$
- $s := S$
- $A \rightarrow wB$ wird zu $(A, w_1, A_1), \dots, (A_{n-1}, w_n, B) \in \Delta$
- $A \rightarrow w$ wird zu $(A, w_1, A_1), \dots, (A_{n-1}, w_n, A_n) \in \Delta$ und $A_n \in F$
(Für $A \rightarrow \lambda$ nur $A \in F$)

$NDEA \rightarrow CFL$

- $V := K$
- $S := s$
- $\delta(q, a) = p$ wird zu $Q \rightarrow aP$
- für alle $q \in F$ wird $Q \rightarrow \lambda$ hinzugefügt

9.3.3 CFL \leftrightarrow PDA

CFL \rightarrow PDA

$$K := \{p, q\}$$

$$\Gamma := V \cup \Sigma$$

$$s := p$$

$$F := \{q\}$$

$$\Delta := \{(p, \lambda, \lambda) \rightarrow (q, S)$$

$$(q, \lambda, A) \rightarrow (q, B) \quad \forall (A \rightarrow B) \in R$$

$$(q, a, a) \rightarrow (q, \lambda) \quad \forall a \in \Sigma$$

{}

PDA \rightarrow CFL

$$1) \quad S \rightarrow [s, z_0, q] \quad \forall q \in K$$

$$2) \quad [q, A, q_{m+1}] \rightarrow a[q_1, B_1, \overbrace{q_2}^{\text{q}_2}, B_2, q_3] \dots$$

 $[q_{m-1}, B_{m-1}, \overbrace{q_m}^{\text{q}_m}, B_m, q_{m+1}]$ für jedeTransition $(q, a, A) \xrightarrow[M]{} (q_1, B_1 \dots B_m)$ in M und jede Wahl von $q_2, \dots, q_{m+1} \in K$

$$3) \quad [q, A, q_1] \rightarrow a \text{ für } a \in \Sigma \cup \{\lambda\}, \text{ falls}$$

$$(q, a, A) \xrightarrow[M]{} (q_1, \lambda) \text{ in } M$$

9.3.4 CFL \rightarrow CNF

- 1) Falls $\lambda \in L(G)$: füge $S_{neu} \rightarrow \lambda$ und $S_{neu} \rightarrow S$ hinzu
- 2) Entferne Regeln $B \rightarrow \lambda$, indem für alle Regeln mit B auf der rechten Seite, bspw. $A \rightarrow \alpha B \beta B \gamma$, zusätzlich die Regeln $A \rightarrow \alpha \beta B \gamma$, $A \rightarrow \alpha B \beta \gamma$ sowie $A \rightarrow \alpha \beta \gamma$ hinzugefügt werden, und $B \rightarrow \lambda$ entfernt wird
- 3) In Regeln $A \rightarrow w_1 w_2 \dots w_k$ mit $k \geq 2$ ersetze alle $w_i \in \Sigma$ durch T_{w_i} (neue NTS) und füge die Regeln $T_{w_i} \rightarrow w_i$ hinzu
- 4) Elimination von $B \rightarrow C$ durch Untersuchung von Ableitungssequenzen von C aus:

- $C \xrightarrow[G]{*} a \in \Sigma^*$: Füge $B \rightarrow a$ hinzu
- $C \xrightarrow[G]{*} E_1 E_2 \dots E_k \ (k \geq 2)$: Füge $B \rightarrow E_1 E_2 \dots E_k$ hinzu

Lösche $B \rightarrow C$

- 5) Elimination von Regeln $A \rightarrow B_1 B_2 \dots B_k \ k \geq 3$ mit neuen NTS N_1, \dots, N_{k-2}

$$\begin{array}{rcl} A & \rightarrow & B_1 N_1 \\ N_1 & \rightarrow & B_2 N_2 \\ & \vdots & \\ N_{k-2} & \rightarrow & B_{k-1} B_k \end{array}$$

9.3.5 Prüfe $w \in L(G)$ (CYK-Alg.)

 G muss in CNF sein!

- Erstelle eine Matrix \mathcal{N} der Größe $n+1 \times n+1$ mit $n := |w|$, die Einträge sind Teilmengen von V mit:

$$A \in \mathcal{N}[i, j] \Leftrightarrow A \xrightarrow[G]{*} w_i w_{i+1} \dots w_{j-1}, i < j$$

 \rightarrow alle Einträge unterhalb der Diagonalen (eingeschlossen) sind \emptyset (da $i \geq j$)

- Berechne erste Nebendiagonale: $A \in \mathcal{N}[i, i+1] \stackrel{!}{\Leftrightarrow} A \xrightarrow[G]{*} w_i$ muss Regel in G sein

- Berechne die restlichen Nebendiagonalen $\mathcal{N}[i,j]$ nacheinander ($j = i+2, \dots, n+1$):

$$\mathcal{N}[i,j] = \bigcup_{k=i+1}^{j-1} \mathcal{N}[i,k] \odot \mathcal{N}[k,j]$$

mit $M_1 \odot M_2 := \{A \in V \mid A \xrightarrow{G} BC \wedge B \in M_1, C \in M_2\}$

- $w \in L(G) \Leftrightarrow S \in \mathcal{N}[1, n+1]$

9.4 Rekursiv aufzählbare Sprachen (RE)

$L \in RE$

- TM konstruieren, die L semi-entscheidet o. entscheidet
- Eine Funktion bzw. TM angeben, die L aufzählt
- Many-One Reduktion $L \leq_{mo} A$ angeben, wobei A rekursiv aufzählbar ist

$L \notin RE$

- Many-One Reduktion von $A \leq_{mo} L$ angeben, wobei A nicht rekursiv aufzählbar ist

Abschluss: unter Vereinigung, Schnitt, Konkatenation, Kleene-Stern, Homomorphismus, Inverse Homomorphismus

!kein! Abschluss: unter Komplement, Differenz

9.5 Entscheidbare Sprachen (REC)

$L \in REC$

- TM konstruieren, die L entscheidet
- TM konstruieren, die L semi-entscheidet, und eine zweite, die \overline{L} semi-entscheidet
- Many-One Reduktion $L \leq_{mo} A$ angeben, wobei A entscheidbar ist

$L \notin REC$

- Many-One Reduktion $A \leq_{mo} L$ angeben, wobei A nicht entscheidbar ist (z.B. Halteproblem H)
- Zeigen, dass L oder \overline{L} nicht rekursiv aufzählbar ist

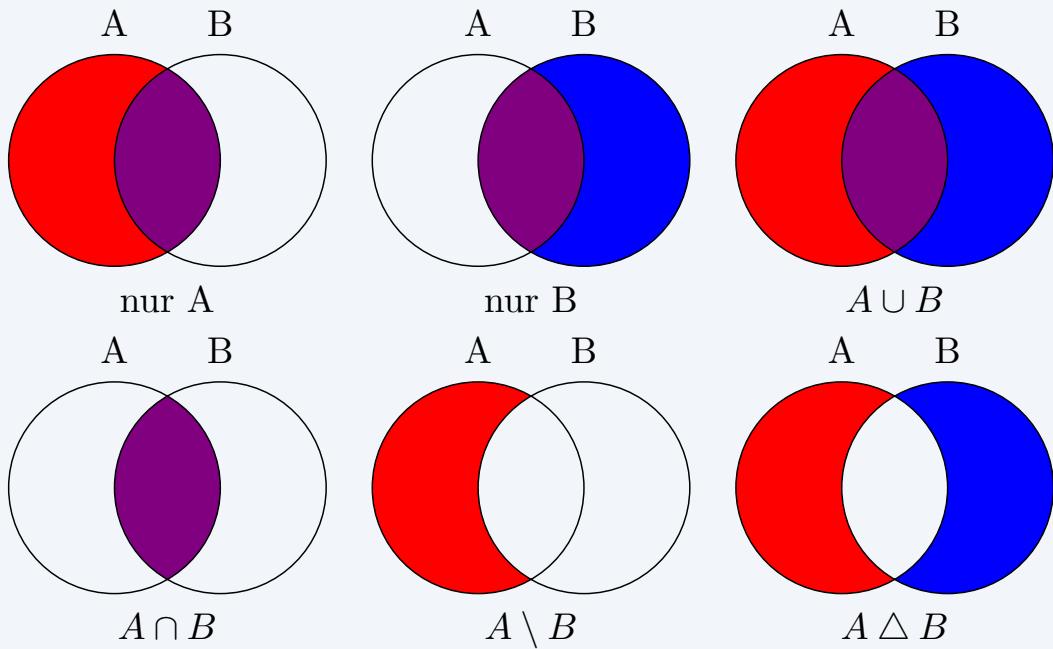
Abschluss: unter Vereinigung, Schnitt, Komplement, Differenz, Konkatenation, Kleene-Stern, Inverse Homomorphismus

!kein! Abschluss: unter Homomorphismus

Kapitel 10 Anhang

10.1 Beispiele

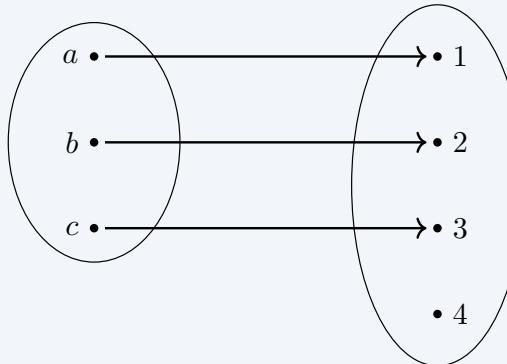
Beispiel zu 1.1.5 Operationen auf Mengen



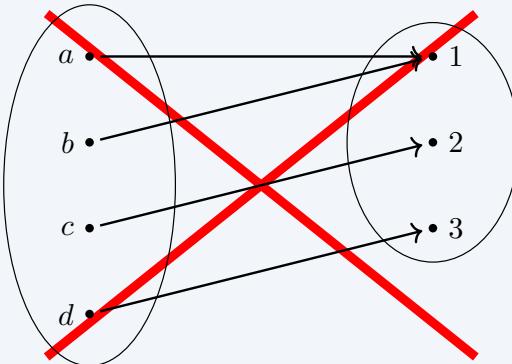
[Zum Text](#)

Beispiel zu 1.3.4 Eigenschaften von Abbildungen

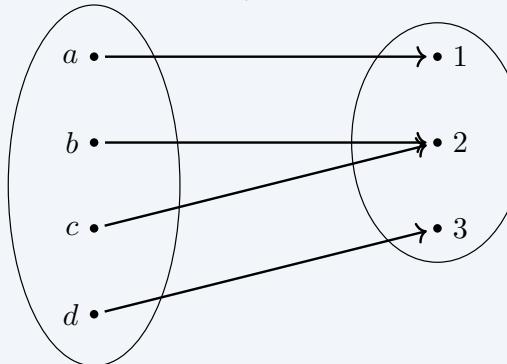
injektiv:



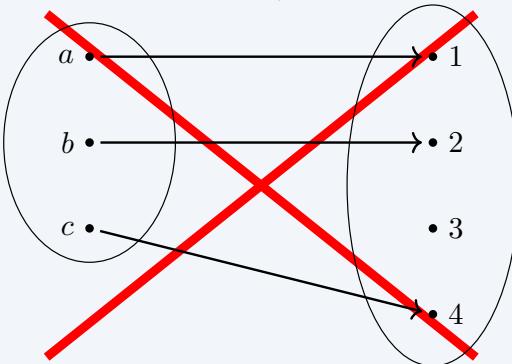
nicht injektiv:



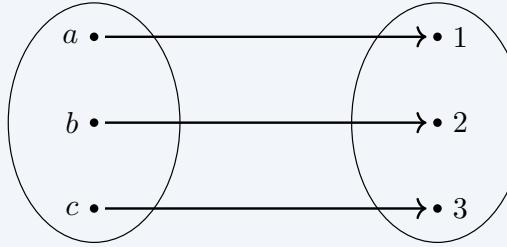
surjektiv:



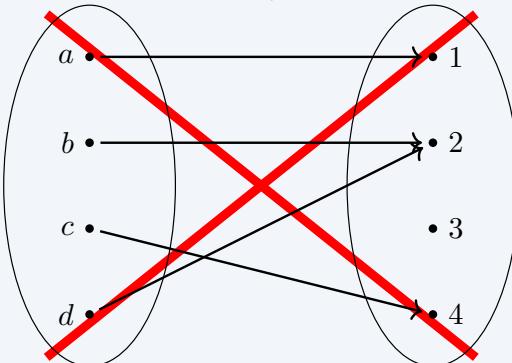
nicht surjektiv:



bijektiv:



nicht bijektiv:

[Zum Text](#)

Beispiel zu 1.4.2 Gleichheitmächtigkeit

Gleichmächtig sind:

- \mathbb{N} , $\mathbb{N} \times \mathbb{N}$, \mathbb{Z} und \mathbb{Q}
- \mathbb{R} , $]0, 1[$, und $\mathcal{P}(\mathbb{N})$ (Potenzmenge)

!Nicht! gleichmächtig sind:

- \mathbb{N} und \mathbb{R} , da \mathbb{R} mächtiger ist als \mathbb{N} (\mathbb{R} ist Überabzählbar, \mathbb{N} nur abzählbar unendlich)

[Zum Text](#)

Beispiel zu 1.4.3 Abzählbar unendlich

- \mathbb{N}
- $\mathbb{N} \times \mathbb{N}$

- \mathbb{Z}
- \mathbb{Q}
- Σ^*

[Zum Text](#)

Beispiel zu 1.4.4 Überabzählbar unendlich

- \mathbb{R}
- $]0,1[$
- $\mathcal{P}(\mathbb{N})$ (Potenzmenge)
- $\{f \mid f : \mathbb{N} \rightarrow \{0,1\}\}$ Menge aller Abbildungen von \mathbb{N} nach $\{0,1\}$ (äquivalent zu $\mathcal{P}(\mathbb{N})$)
- $\mathbb{N}^\mathbb{N} (= \{f \mid f : \mathbb{N} \rightarrow \mathbb{N}\},$ Menge aller Abbildungen von \mathbb{N} nach \mathbb{N})
- $\mathcal{P}(\Sigma^*)$ Menge aller Sprachen

[Zum Text](#)

Beispiel zu 3.1.7 Funktionen auf Sprachen

Sei $\Sigma := \{0,1\}$

1. $L := \Sigma^*$ ist die Menge aller Bitstrings
2. $L := \{0\}^*. \{1\}^+$ ist die Menge aller Wörter die mit beliebig vielen 0'en starten und mindestens einer 1 enden. Dies ist äquivalent zu: $L = \{0\}^*. 1. \{1\}^*$
→ Darstellung nicht eindeutig!
3. $L := \{x \in \{0,1\}^* \mid x \text{ ist Binärdarstellung einer Primzahl}\}$
4. Sei $S \subseteq \mathbb{N}, L_S := \{x \in \{0,1\}^* \mid x \text{ ist Binärdarstellung von } n \in S\}$

[Zum Text](#)

Beispiel zu 3.2 Längen-lexikografische Ordnung

Für das Alphabet $\Sigma = \{a,b\}$ ergibt sich die Ordnung:

$$\lambda, a, b, aa, ab, ba, bb, aaa, aab, \dots$$

Dabei steht λ (das leere Wort) an erster Stelle, gefolgt von den Wörtern der Länge 1, dann der Länge 2, und so weiter.

[Zum Text](#)

Beispiel zu 3.3.1 Reguläre Ausdrücke

- 1.

$L := \{w \in \Sigma^* \mid w \text{ startet mit } a \text{ und endet mit zwei } b\}$

$$x := a(a \cup b)^*bb$$

$$\mathcal{L}(x) = L$$

- 2.

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

$$x := \underbrace{(0^*1)^*}_{\text{Alle Wörter die auf 1 enden}} \cup \underbrace{(1^*0)^*}_{\text{Alle Wörter die auf 0 enden}}$$

$\mathcal{L}(x) = \Sigma^*$

[Zum Text](#)

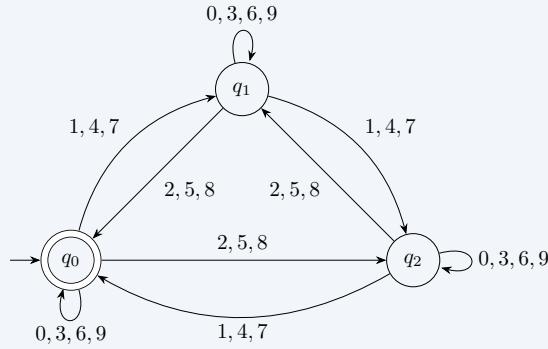
Beispiel zu 4.1.2 Rechnung

$$\Sigma := \{0, 1, \dots, 9\}$$

$$L := \{x \in \Sigma^* \mid x \text{ ist als Dezimalzahl durch 3 teilbar}\}$$

Für $x := x_1 x_2 \dots x_n, x_i \in \Sigma$ gilt:

$$x \text{ ist durch 3 teilbar} \Leftrightarrow \text{Quersumme}(x) := \sum_{i=1}^n x_i \text{ ist durch 3 teilbar.}$$



Interpretation: Automat liest $x_1 \dots x_n$ von links nach rechts. Er befindet sich in q_i , wenn die Quersumme des bisher gelesenen Teils von x durch i teilbar ist.

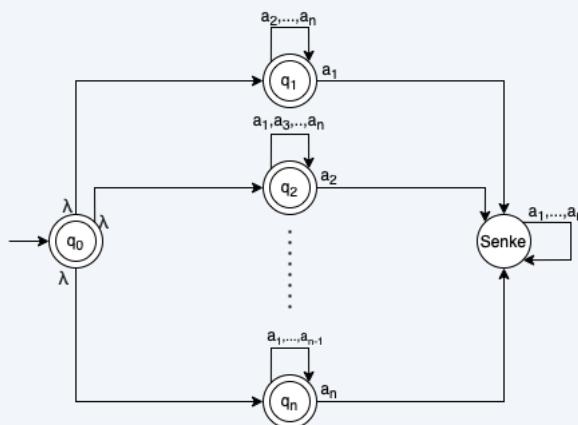
Der Automat startet in q_0 und x ist durch 3 teilbar, genau dann wenn er sich nach Lesen von x in q_0 befindet.

[Zum Text](#)

Beispiel zu 4.2.1 Nicht-Deterministische Übergänge

Sei $\Sigma := \{a_1, \dots, a_n\}$ und $L := \{w \in \Sigma^* \mid w \text{ enthält } \underline{\text{nicht alle }} a\}$

Wenn wir bei endlichen Automaten gewisse nicht-deterministische Übergänge erlauben, lässt sich L gut behandeln. Wir erlauben λ -Übergänge (und auch weitere nicht-det. Übergänge):

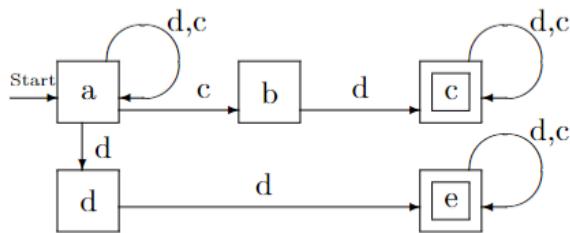


M kann im Startzustand nicht-deterministisch wählen, wohin mit Lesen von λ gewechselt wird.

$w \in L$ genau dann, wenn es mindestens eine Rechnung von M auf w gibt, die in einem $q \in F$ endet.

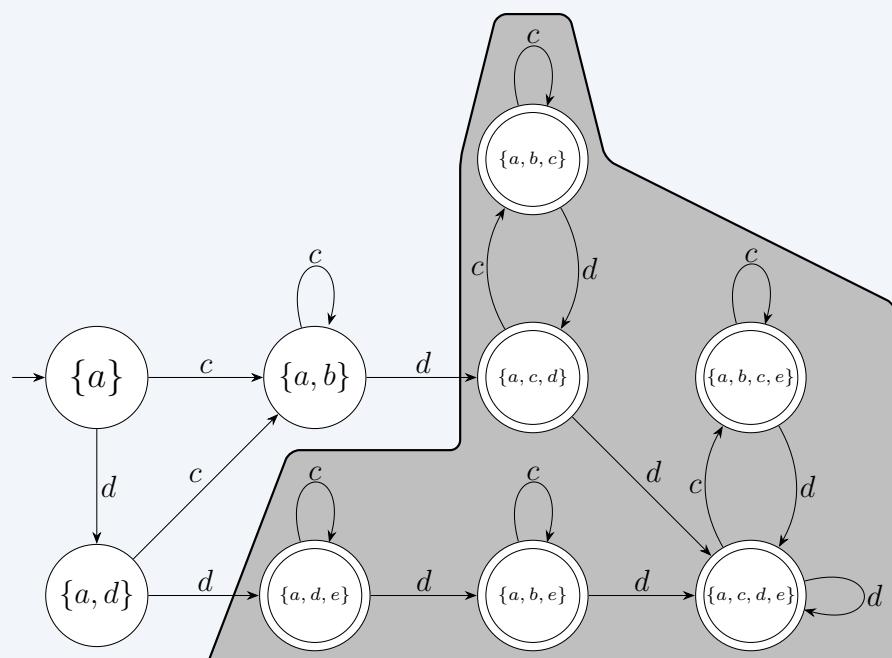
[Zum Text](#)

Beispiel zu NDEA zu DEA



NDEA

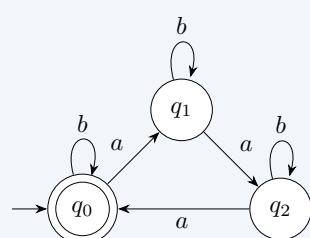
↓



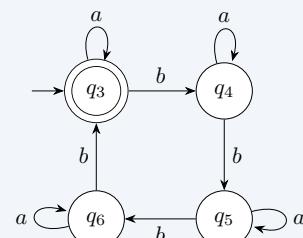
DEA

Zum Text

Beispiel zu 4.4 2 DEA simultan simulieren: Produktautomat

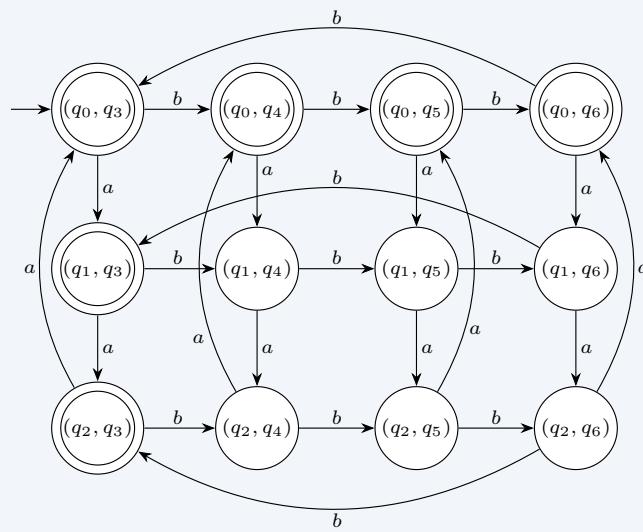


DEA 1



DEA 2

↓

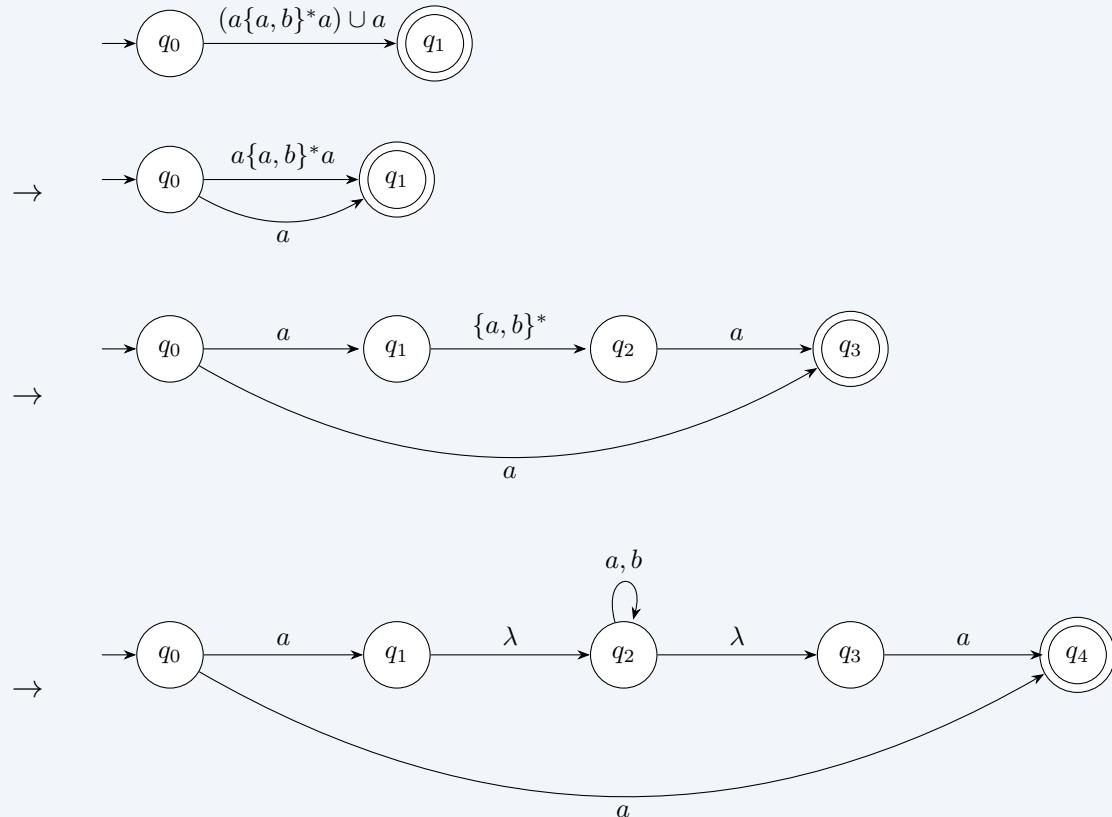


Produktautomat (simuliert beide gleichzeitig)

[Zum Text](#)

Beispiel zu 4.5.1 Top-Down

reguläre Ausdruck: $(a\{a,b\}^*a) \cup a$

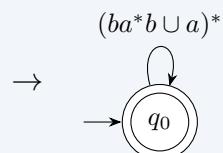
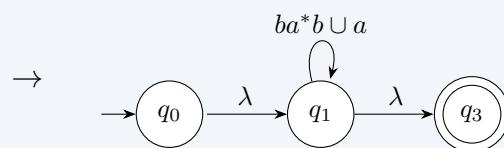
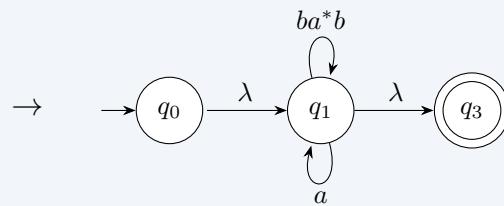
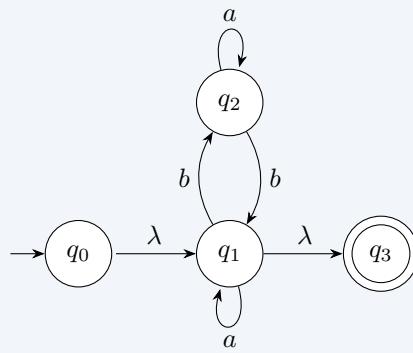
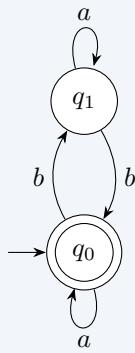


[Zum Text](#)

Beispiel zu 4.6.2 Umkehrung von Top-Down

NDEA:

(erkennt alle Wörter mit gerader Anzahl b's)



$$\rightarrow R = (ba^*b \cup a)^*$$

Zum Textreg. PE: $a^{4n}b^m$

$L = \{a^{4n}b^m \mid n, m \in \mathbb{N}\}$ hat die reguläre Pumping-Eigenschaft, was wie folgt gezeigt werden:

$$\exists n \in \mathbb{N} \rightarrow \mathbf{n = 4}$$

$$\forall w \in L \text{ mit } |w| \geq 4 \quad (\text{muss gezeigt werden...})$$

Wahl der Zerlegung anhand Fallunterscheidung:

- (Angenommen $0 \in \mathbb{N}$) Fall 1: $\#\mathbf{a}(\mathbf{w}) = \mathbf{0}$ ($\mathbf{w} = \mathbf{a}^{4\mathbf{n}}\mathbf{b}^{\mathbf{m}}, \mathbf{n} = \mathbf{0}$)

Aus $\#_a(w) = 0$ folgt, dass w ausschließlich b 's enthält bzw. $w = b^{|w|}$ ($|w| \geq 4$):

$$\exists x, y, z \in \Sigma^* \rightarrow \mathbf{x} = \lambda, \mathbf{y} = \mathbf{b}, \mathbf{z} = \mathbf{b}^{|w|-1}$$

Für die Reguläre Pumping-Eigenschaft müssen nun die Eigenschaften 1. bis 4. gelten:

1. $w = xyz = \lambda.b.b^{|w|-1} = b^{|w|} \quad \checkmark$
2. $|xy| = |\lambda b| = \underbrace{|\lambda|}_{=0} + \underbrace{|b|}_{=1} = 1 \leq 4 \quad \checkmark$
3. $|y| = |b| = 1 \geq 1 \quad \checkmark$
4. $\forall i \in \mathbb{N}_0 : xy^i z \in L \quad \checkmark$

denn: $\rightarrow i = 0 : xz = \lambda.b^{|w|-1} \in L \quad (|w| \geq 4 \implies |w| - 1 \geq 3 \in \mathbb{N})$

$$i = 1 : xy^1 z = \lambda.b.b^{|w|-1} = b^{|w|} = w \in L$$

$$i > 1 : xy^i z = \lambda.b^i.b^{|w|-1} = b^{|w|+i-1} \in L$$

$$(|w| \geq 4 \wedge i > 1 \implies |w| + i - 1 \geq 5 \in \mathbb{N})$$

- Fall 2: $\#\mathbf{a}(\mathbf{w}) \neq \mathbf{0}$ ($\mathbf{w} = \mathbf{a}^{4\mathbf{n}}\mathbf{b}^{\mathbf{m}}, \mathbf{n} \neq \mathbf{0}$)

Aus $\#_a(w) \neq 0$ und $w = a^{4n}b^m$ folgt, dass w mindestens 4 a 's am Anfang enthält. Daher kann man w bei der Zerlegung in seinen ersten 4 Buchstaben (nur a 's) $w_{1-4} = aaaa$ ($|w_{1-4}| = 4$) und seinen Rest $w_{\text{Rest}} \in (aaaa)^{n-1}.b^m$ ($|w_{\text{Rest}}| = |w| - 4 \geq 0$) aufteilen ($w = w_{1-4}.w_{\text{Rest}}$):

$$\exists x, y, z \in \Sigma^* \rightarrow \mathbf{x} = \lambda, \mathbf{y} = \mathbf{w}_{1-4}, \mathbf{z} = \mathbf{w}_{\text{Rest}}$$

Für die Reguläre Pumping-Eigenschaft müssen nun die Eigenschaften 1. bis 4. gelten:

1. $w = xyz = \lambda.\underbrace{w_{1-4}.w_{\text{Rest}}}_{=w} \quad \checkmark$
2. $|xy| = |\lambda w_{1-4}| = |\lambda aaaa| = \underbrace{|\lambda|}_{=0} + \underbrace{|aaaa|}_{=4} = 4 \leq 4 \quad \checkmark$
3. $|y| = |w_{1-4}| = |aaaa| = 4 \geq 1 \quad \checkmark$
4. $\forall i \in \mathbb{N}_0 : xy^i z \in L \quad \checkmark$

denn: $\rightarrow i = 0 : xz = \lambda.w_{\text{Rest}} = (aaaa)^{n-1}.b^m \in L$

$$(n \neq 0 \wedge n \in \mathbb{N} \implies n \geq 1 \implies n - 1 \geq 0 \in \mathbb{N})$$

$$i = 1 : xy^1 z = w \in L$$

$$i > 1 : xy^i z = \lambda.(w_{1-4})^i.w_{\text{Rest}} = (aaaa)^i.(aaaa)^{n-1}.b^m \in L$$

$$((aaaa)^{i+n-1} \wedge i > 1 \implies (aaaa)^x \text{ mit } x \geq n+1 \in \mathbb{N})$$

[Zum Text](#)

keine reg. PE: $a^n b^n$

$L = \{a^n b^n \mid n \in \mathbb{N}\}$ hat nicht die reguläre Pumping-Eigenschaft, was wie folgt bewiesen werden kann:

- Sei ein beliebiges, aber festes $p \in \mathbb{N}$ gegeben (die Pumping-Konstante)
- Setze $w = a^p b^p$ ($w \in L_1$ und $|w| = 2p \geq p \checkmark$)
- Sei eine beliebige, aber feste Zerlegung $w = xyz$ mit $x, y, z \in \Sigma^*$, $|xy| \leq p$ und $|y| \geq 1$ gegeben (Zerlegung erfüllt i)-iii))
- $\exists i \in \mathbb{N}_0 : xy^i z \notin L_1 :$
 - Fall 1: y enthält ab
→ Setze $i = 2$, dann folgt $xy^2 z \notin L$.
Denn y enthält ab , und $xy^2 z$ enthält somit 2 mal ab , aber $a^n b^n$ (L) darf ausschließlich ein einziges ab (in der Mitte) enthalten
 - Fall 2: y enthält kein ab
→ Setze $i = 0$, dann folgt $xy^0 z = xz \notin L$.
Denn y enthält kein ab , daher enthält y entweder ausschließlich a 's oder b 's, und durch Abpumpen ($i = 0$) hat xz nicht mehr gleich viele a 's wie b 's

[Zum Text](#)

keine reg. PE: a^i mit i Primzahl

$L = \{w \in \{a\}^* \mid |w| \in \mathbb{P}\}$ (Primzahl) hat nicht die reguläre Pumping-Eigenschaft, was wie folgt bewiesen werden kann:

- Sei $n \in \mathbb{N}$ gegeben (Pumping-Zahl)
- Setze $w = a^m$ mit $m \in \mathbb{N}$, $m \in \mathbb{P}$ (m ist Primzahl), $m \geq n + 2$ ($w \in L \checkmark$)
- Sei die Zerlegung $w = xyz$ mit $|y| \geq 1$ und $|xy| \leq n$ gegeben
- Setze $i = |xz| = |x| + |z|$, dann:

$$|xy^i z| = |x| + (\underbrace{|x| + |z|}_i) \cdot |y| + |z| = (\underbrace{|x| + |z|}_{\geq 2}) \cdot (\underbrace{|y| + 1}_{\geq 2})$$

$$(|y| \geq 1 \implies |y| + 1 \geq 2)$$

$$(|xyz| = m \geq n + 2 \wedge |xy| \leq n \implies |x| + |z| \geq 2)$$

$$\Rightarrow |xy^{|x|+|z|} z| \text{ hat die Teiler } 1, (|x| + |z|) \geq 2, (|y| + 1) \geq 2 \text{ und } (|x| + |z|) \cdot (|y| + 1) \geq 4$$

$$\Rightarrow |xy^{|x|+|z|} z| \text{ ist keine Primzahl!}$$

$$\Rightarrow |xy^{|x|+|z|} z| \notin L$$

$$\Rightarrow L \text{ hat nicht die reguläre Pumping-Eigenschaft!}$$

[Zum Text](#)

 $a^n b^n$ nicht regulär

$L := \{a^n b^n \mid n \geq 1\}$

$$\begin{array}{lll}
 x = ab & \forall z \in \Sigma^* \setminus \{\lambda\} : xz \notin L \\
 y = a^2b^2 & \text{---} & : yz \notin L \\
 & x \approx_L y
 \end{array}$$

Betrachte $\forall i, j \in \mathbb{N}$ $x_i = a^i$, dann ist $x_i \not\approx_L x_j$ für $i \neq j$

$\Rightarrow \approx_L$ hat ∞ viele Äquivalenzklassen

$\Rightarrow L$ nicht regulär.

[Zum Text](#)

a^n mit n Quadratzahl nicht regulär

$$L = \{w \in \{a\}^* \mid |w| = n^2, n \in \mathbb{N}\}$$

Nach dem Satz von Myhill-Nerode ist eine Sprache L regulär, wenn die Äquivalenzrelation bezüglich L endlich viele Äquivalenzklassen hat. Um zu zeigen, dass L **nicht regulär** ist, muss also gezeigt werden:

$$|\{a\}^*/\approx_L| = \infty$$

Es kann gezeigt werden: $\forall i, j \in \mathbb{N}, i \neq j : [a^i]_{\approx_L} \neq [a^j]_{\approx_L}$

Beweis mittels Widerspruch

Annahme: $\exists i, j \in \mathbb{N}, i < j : a^i \approx_L a^j$

Ohne Einschränkung der Allgemeinheit gilt:

$$i = n^2, j = m^2 \quad n, m \in \mathbb{N} \quad (i < j \implies n < m)$$

andernfalls betrachte: $a^{i+k} \approx_L a^{j+k}, k := \min\{l \in \mathbb{N} \mid i+k = p^2, p \in \mathbb{N}\}$

Es folgt:

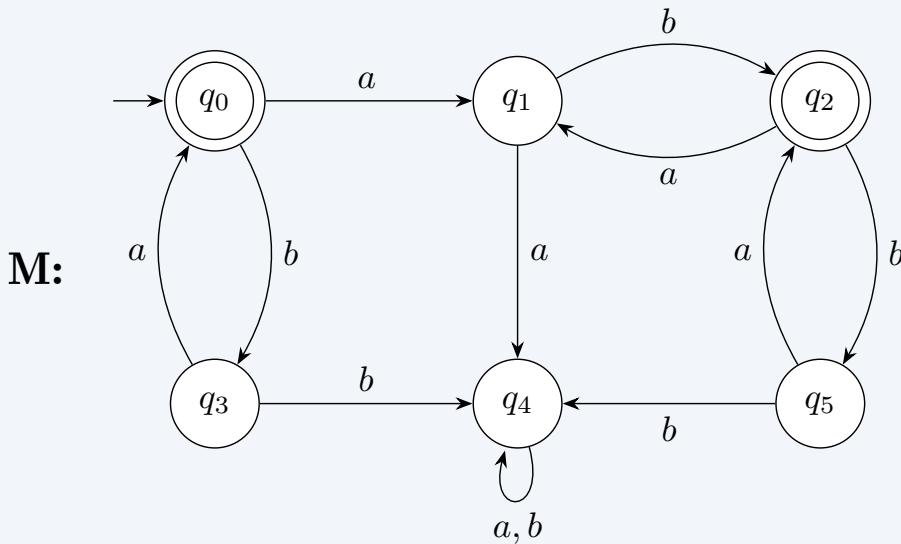
$$\begin{aligned}
 a^i \approx_L a^j &\implies a^{n^2} \approx_L a^{m^2} \\
 a^{n^2} \approx_L a^{m^2} &\implies \forall z \in \{a\}^* : a^{n^2}.z \in L \Leftrightarrow a^{m^2}.z \in L \\
 \rightarrow z = a^{2 \cdot n + 1} : a^{n^2}.a^{2 \cdot n + 1} &= a^{n^2 + 2 \cdot n + 1} = a^{(n+1)^2} \in L \\
 &\left(|a^{(n+1)^2}| = (n+1)^2 \text{ und } (n+1)^2 \text{ ist eine Quadratzahl} \right) \\
 \text{aber: } a^{m^2}.a^{2 \cdot n + 1} &= a^{m^2 + 2 \cdot n + 1} \notin L \quad \text{!} \\
 &\left(\begin{array}{l} m^2 < m^2 + 2 \cdot n + 1 \stackrel{n < m}{<} m^2 + 2 \cdot m + 1 = (m+1)^2 \\ \implies m^2 + 2 \cdot n + 1 \text{ ist keine Quadratzahl!} \end{array} \right) \\
 \Rightarrow \quad \left| \left\{ [a^i]_{\approx_L} \mid i \in \mathbb{N} \right\} \right| &= \infty
 \end{aligned}$$

Es konnte gezeigt werden, dass L unendlich viele Äquivalenzklassen bezüglich \approx_L hat, also ist L **nicht regulär!**

[Zum Text](#)

Beispiel zu 4.15.1 Algorithmus Zustandsminimierung

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



- \equiv_0 : $\underbrace{\{q_1, q_3, q_4, q_5\}}_{K \setminus F}, \underbrace{\{q_0, q_2\}}_F$

- \equiv_1 :

$q \in K \setminus F$	$\delta(q, a)$	$\delta(q, b)$	$q \in F$	$\delta(q, a)$	$\delta(q, b)$
q_1	q_4	q_2	q_0	q_1	q_3
q_3	q_0	q_4	q_2	q_1	q_5
q_4	q_4	q_4			
q_5	q_2	q_4			

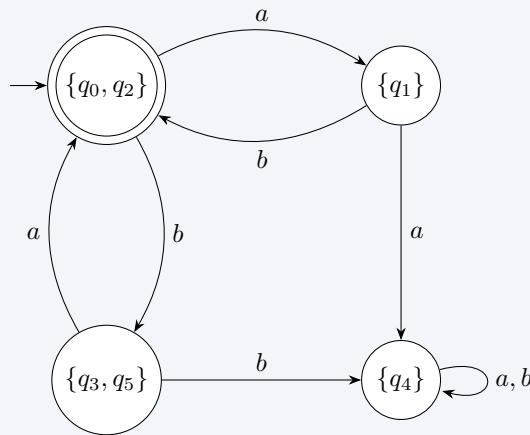
$$\rightarrow \equiv_1 : \{q_0, q_2\}, \{q_1\}, \{q_3, q_5\}, \{q_4\}$$

- $\equiv_2 = \equiv_1 \Rightarrow \equiv_1 = \equiv_M$

\rightarrow die Äquivalenzklassen haben sich nicht mehr verändert, also kann der Algorithmus terminieren, und aus den bestehenden Äquivalenzklassen kann der minimale Zustandsautomat konstruiert werden:

- Jede Äquivalenzklasse von \equiv_M wird eigener Zustand im minimalen Zustandsautomat
- Startzustand ist $\{q_0, q_2\}$, da q_0 in M Startzustand ist
- Endzustand ist $\{q_0, q_2\}$, da $q_0, q_2 \in F$ in M
- Für die Übergänge nimmt man sich einen Repräsentanten der Äquivalenzklasse (egal welcher, da die Zustände innerhalb einer Äquivalenzklasse äquivalent sind) und schaut für alle Buchstaben im Alphabet Σ , in welche Äquivalenzklasse man nach dem Lesen des Buchstabens gelangt.

minimierter Zustandsautomat:

[Zum Text](#)

Beispiel zu 4.16.4 Äquivalenzproblem II

$\Sigma := \{a_1, \dots, a_n\}$, $L := \{w \in \Sigma^* \mid w \text{ enthält } \underline{\text{nicht alle}} \ a_i\}$

Schon gesehen: \exists NDEA für L mit $O(n)$ Zuständen
(siehe Beispiel [hier](#))

Behauptung: Jeder DEA für L hat $\geq 2^n$ Zustände

Beweis

Wir finden $\geq 2^n$ ÄK bzgl. L :

Zu $A \subseteq \Sigma$ ($A \neq \emptyset, A \neq \Sigma$) definiere $L_A := \{w \in \Sigma^* \mid w \text{ enthält alle } a \in A \text{ und keine } b \in \Sigma \setminus A\}$. Σ hat 2^n viele Teilmengen (Kardinalität der Potenzmenge)!

Für zwei solche $A \neq B$ und $x \in L_A$, $y \in L_B$ gilt $x \not\sim_L y$!

O.E.d.A. (Ohne Einschränkung der Allgemeinheit):

Sei $B \setminus A \neq \emptyset$

Sei $z \in L_{\Sigma \setminus B}$, dann folgt $yz \notin L$, aber $xz \in L$ (Buchstaben aus $B \setminus A$ fehlen) □

[Zum Text](#)

Beispiel für inversen Homomorphismus

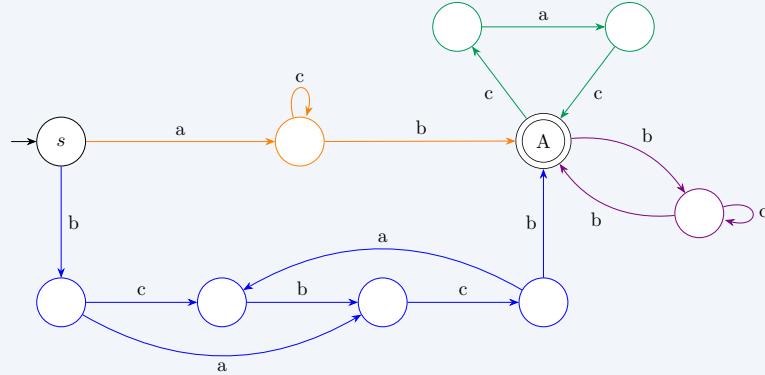
Sei der Homomorphismus $h : \{a, b\}^* \rightarrow \{0, 1\}^*$ mit $h(a) = 01$, $h(b) = 0$

Wenn der ursprüngliche Automat M (mit $L(M) = L \subseteq \{0, 1\}^*$) die Transitionen $(q_1, 0) = q_2$ sowie $(q_2, 1) = q_3$ hat, dann hat der Automat M' für $h^{-1}(L)$ die Transitionen $(q_1, a) = q_3$ sowie $(q_1, b) = q_2$, denn beim Lesen von a macht M' im Prinzip:

Lesen von $h(a) = 01$ im Zustand q_1 , was im Automat M zu Zustand q_2 und anschließend q_3 führt.

[Zum Text](#)

Beispiel zu 4.18 ÄK → REG



$$\alpha = \left(\textcolor{blue}{a} . (\textcolor{red}{c})^* . \textcolor{brown}{b} \cup b . (\textcolor{red}{c}\textcolor{blue}{b}\textcolor{red}{c} \cup \textcolor{red}{a}\textcolor{red}{c}) . (\textcolor{red}{a}\textcolor{red}{b}\textcolor{red}{c})^* . \textcolor{brown}{b} \right) . \left(\textcolor{red}{c}\textcolor{blue}{a}\textcolor{red}{c} \cup b . (\textcolor{red}{c})^* . \textcolor{brown}{b} \right)^*$$

Zum Text

Grammatik zu $a^n b^n$

$$L := \{a^n b^n\}$$

$$\begin{array}{ll} G: & V := \{S\} \\ & \Sigma := \{a, b\} \\ & R := \{S \rightarrow aSb \mid \lambda\} \end{array}$$

$$\Rightarrow L(G) = L$$

Zum Text

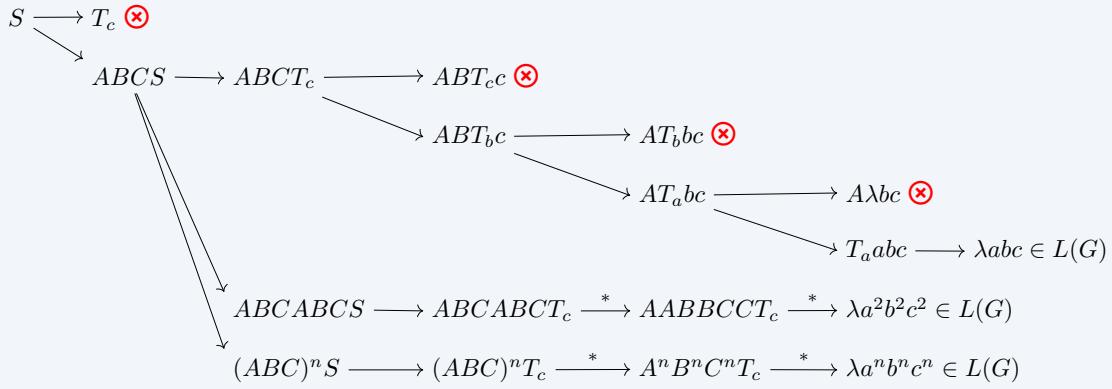
Grammatik zu $a^n b^n c^n$

Definiere $G = (V, \Sigma, S, R)$ mit:

$$\begin{aligned}
 V &:= \{S, A, B, C, T_a, T_b, T_c\} \\
 \Sigma &:= \{a, b, c\} \\
 R &:= \{ \begin{array}{lcl} S & \rightarrow & ABCS \mid T_c, \\ CA & \rightarrow & AC, \\ BA & \rightarrow & AB, \\ CB & \rightarrow & BC, \\ CT_c & \rightarrow & T_c c \mid T_b c, \\ AT_a & \rightarrow & T_a a, \\ BT_b & \rightarrow & T_b b \mid T_a b, \\ T_a & \rightarrow & \lambda \end{array} \}
 \end{aligned}$$

Gesucht: $L(G)$

Mögliche Ableitungen:



$$\Rightarrow L(G) = \{a^n b^n c^n \mid n \geq 1\}$$

[Zum Text](#)

Beispiel zu 5.3 Kellerautomat

$$L := \{a^n b^n \mid n \in \mathbb{N}\} \quad \Sigma := \{a, b\} \quad \Gamma := \{b\} \quad K := \{q_0, q_1, q_2\} \quad s := q_0 \quad F := \{q_2\}$$

$$\begin{aligned} \Delta := \{ & (q_0, a, \lambda) \rightarrow (q_0, b) \\ & (q_0, b, b) \rightarrow (q_1, \lambda) \\ & (q_1, b, b) \rightarrow (q_1, \lambda) \\ & (q_1, \lambda, \lambda) \rightarrow (q_2, \lambda) \} \end{aligned}$$

[Zum Text](#)

Beispiel zu 5.4 Äquivalenz von PDA und kfr. Grammatik

$$L := \{a^n b^n \mid n \in \mathbb{N}\}$$

Ein PDA (mit den verlangten Bedingungen ① und ②) für L gemäß dem Beweis ist:

$$\begin{aligned} K := & \{s_0, s_1\} \\ \Sigma := & \{a, b\} \\ \Gamma := & \{z_0, T\} \\ s := & s_0 \\ \Delta := & \{1. (s_0, \lambda, z_0) \rightarrow (s_0, \lambda) \quad (\lambda \in L(M)) \\ & 2. (s_0, a, z_0) \rightarrow (s_0, T) \\ & 3. (s_0, a, T) \rightarrow (s_0, TT) \\ & 4. (s_0, b, T) \rightarrow (s_1, \lambda) \\ & 5. (s_1, b, T) \rightarrow (s_1, \lambda) \quad \} \end{aligned}$$

Konstruktion von G :

– Startregeln von G (s_0 ist Startzustand) (Typ(1) Regeln)

$$\begin{aligned} S &\rightarrow [s_0, z_0, s_0]_A \\ S &\rightarrow [s_0, z_0, s_1]_B \end{aligned}$$

– Transition 1. $(s_0, \lambda, z_0) \rightarrow (s_0, \lambda)$ in M zu Regel in G (Typ(3) Regel)

$$[s_0, z_0, s_0]_A \rightarrow \lambda$$

– Transition 2. $(\underbrace{s_0}_q, a, \underbrace{z_0}_A) \rightarrow (\underbrace{s_0}_{q_1}, \underbrace{T}_{B_1})$ in M zu Regel in G

Typ(2) Regel, also jede Wahl von Zuständen! $|K| = 2, m = |T| = 1 \implies 2^1 = 2$ Regeln!

q	A	q_2	q_1	B_1	q_2	q_2
$[s_0, z_0, s_0]_A$	\rightarrow	$a[s_0, T, s_0]_C$		$\overline{s_0}$		
$[s_0, z_0, s_1]_B$	\rightarrow	$a[s_0, T, s_1]_D$		s_1		

- Transition 3. $(\underbrace{s_0}_q, a, \underbrace{T}_A) \rightarrow (\underbrace{s_0}_{q_1}, \underbrace{TT}_{B_1 B_2})$ in M zu Regel in G

Typ(2) Regel, also jede Wahl von Zuständen! $|K| = 2, m = |TT| = 2 \implies 2^2 = 4$ Regeln!

q	A	q_3	q_1	B_1	q_2	B_2	q_3	$q_2 \mid q_3$
$[s_0, T, s_0]_C$	\rightarrow	$a[s_0, T, s_0]_C$	$[s_0, T, s_0]_C$	s_0	s_0	s_0	s_0	$s_0 \mid s_0$
$[s_0, T, s_0]_C$	\rightarrow	$a[s_0, T, s_1]_D$	$[s_1, T, s_0]_E$	s_1	s_0	s_1	s_0	$s_0 \mid s_0$
$[s_0, T, s_1]_D$	\rightarrow	$a[s_0, T, s_0]_C$	$[s_0, T, s_1]_D$	s_0	s_1	s_0	s_1	$s_1 \mid s_1$
$[s_0, T, s_1]_D$	\rightarrow	$a[s_0, T, s_1]_D$	$[s_1, T, s_1]_F$	s_1	s_1	s_1	s_1	$s_1 \mid s_1$

- Transition 4. $(s_0, b, T) \rightarrow (s_1, \lambda)$ in M zu Regel in G (Typ(3) Regel)

$$[s_0, T, s_1]_D \rightarrow b$$

- Transition 5. $(s_1, b, T) \rightarrow (s_1, \lambda)$ in M zu Regel in G (Typ(3) Regel)

$$[s_1, T, s_1]_F \rightarrow b$$

Wir schreiben diese Regeln etwas verkürzt wie folgt:

$$\begin{aligned} S &\rightarrow A \mid B \\ A &\rightarrow aC \mid \lambda \\ B &\rightarrow aD \\ C &\rightarrow aCC \mid aDE \\ D &\rightarrow aCD \mid aDF \mid b \\ F &\rightarrow b \end{aligned}$$

E kann nicht abgeleitet werden, daher sind alle Regeln mit E (auf der rechten Seite) überflüssig.

Nach dem Entfernen der Regeln mit E (auf der rechten Seite) stellt man fest, dass C lediglich auf 2 C s abgeleitet werden kann, daher sind auch alle Regeln mit C (auf der rechten Seite) überflüssig.

Schlussendlich wird B immer zu aD , A zu λ und F zu b .

Auf dieser Basis vereinfachen wir die Regeln noch einmal.

$$\begin{aligned} S &\rightarrow aD \mid \lambda \\ D &\rightarrow aDb \mid b \end{aligned}$$

[Zum Text](#)

Beispiel zu 5.5 Chomsky Normalform

Grammatik G :

$$V = \{S, A, B, C\}, \quad \Sigma = \{a, b, c\}$$

$$\begin{array}{lcl} R = \{ S & \rightarrow & aABC \mid a \\ & A & \rightarrow aA \mid a \\ & B & \rightarrow bcB \mid bc \\ & C & \rightarrow cC \mid c \end{array} \quad \boxed{}$$

Regeln von G' (G in CNF:)

$$\begin{array}{lcl} R' = \{ S & \rightarrow & A'T_1 \mid a \\ & A' & \rightarrow a \\ & T_1 & \rightarrow AT_2 \\ & T_2 & \rightarrow BC \\ & A & \rightarrow A'A \mid a \\ & B & \rightarrow B'T_3 \mid B'C' \\ & T_3 & \rightarrow C'B \\ & B' & \rightarrow b \\ & C & \rightarrow C'C \mid c \\ & C' & \rightarrow c \end{array} \quad \boxed{}$$

[Zum Text](#)kfr. PE für $a^n b^n c^n$ $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ hat nicht die kfr. PE.**Beweis mittels Widerspruch:**

Sei $w = a^m b^m c^m \in L$ lang genug. Wäre $w = rstuv$ eine geeignete Zerlegung, dann können s und u nur einen Buchstaben aus $\Sigma = \{a, b, c\}$ enthalten (anderenfalls würde Pumpen die Reihenfolge zerstören). Somit würde Pumpen nur die Anzahl zweier Buchstaben verändern, also aus L hinausführen.

$\Rightarrow L$ ist nicht kfr.

[Zum Text](#)kfr. PE für $L := \{xxx \mid x \in \{a, b\}^*\}$

Es kann gezeigt werden, dass $L := \{xxx \mid x \in \{a, b\}^*\}$ nicht die kontextfreie Pumping-Eigenschaft hat:

- Sei ein beliebiges, aber festes $m \in \mathbb{N}$ gegeben (die Pumping-Konstante)
- Setze $w = \underbrace{b^{(2^m)} a}_{x} \underbrace{b^{(2^m)} a}_{x} \underbrace{b^{(2^m)} a}_{x} \quad (w \in L \text{ und } |w| = 3 \cdot 2^m + 3 \geq 2^m \checkmark)$
- Sei eine beliebige, aber feste Zerlegung $w = rstuv$ mit $r, s, t, u, v \in \Sigma^*$, $|stu| \leq 2^m$ und $|su| \geq 1$ gegeben (Zerlegung erfüllt i)-iii))
- $\exists i \in \mathbb{N}_0 : rs^i tu^i v \notin L :$
 - Fall 1: s oder u enthalten ein a
(Es können nicht beide a enthalten, denn zwischen 2 a 's ist immer ein $b^{(2^m)}$,

womit $|stu| \leq 2^m$ verletzt werden würde)

→ Setze $i = 0$, dann folgt $rs^0tu^0v = rtv \notin L$.

Denn s oder u enthalten ein a , mit $i = 0$ wird dieses abgepumpt, $\#_a(rtv) = 2$ und ein Wort mit 2 a 's kann man nicht in 3 gleiche Teilworte aufteilen!

- Fall 2: s und u enthalten kein a

→ Setze $i = 2$, dann folgt $rs^2tu^2v \notin L$.

Denn s und u enthalten ausschließlich b 's (sonst Fall 1), daher ist rs^2tu^2v , unabhängig von der Zerlegung im Detail, entweder $b^{(2^m+|s|)}ab^{(2^m+|u|)}ab^{(2^m)}a$ oder $b^{(2^m)}ab^{(2^m+|s|)}ab^{(2^m+|u|)}a$, und wegen $|su| \geq 1$ ist entweder $b^{(2^m+|s|)} \neq b^{(2^m)}$ und/oder $b^{(2^m+|u|)} \neq b^{(2^m)}$, somit kann man rs^2tu^2v nicht in drei gleiche Teilworte aufteilen!

$(b^{(2^m+|s|)}ab^{(2^m)}ab^{(2^m+|u|)}a$ ist nicht möglich, da t sonst mindestens $ab^{(2^m)}a$ beinhalten müsste, und dies verletzt unweigerlich $|stu| \leq 2^m$)

[Zum Text](#)

Quadratzahl hat keine kfr. PE

Es kann gezeigt werden, dass $L := \{a^{k^2} \mid k \in \mathbb{N}\}$ nicht die kontextfreie Pumping-Eigenschaft hat:

- Sei ein beliebiges, aber festes $m \in \mathbb{N}$ gegeben (die Pumping-Konstante)
- Setze $w = a^{n^2}$ mit $n \in \mathbb{N}, n \geq 2^m$
- Sei eine beliebige, aber feste Zerlegung $w = rstuv$ mit $r, s, t, u, v \in \Sigma^*$, $|stu| \leq 2^m$ und $|su| \geq 1$ gegeben (Zerlegung erfüllt i)-iii))
- $\exists i \in \mathbb{N}_0 : rs^i tu^i v \notin L :$

Eine beliebige Zerlegung von w hat die Form: $r = a^b$, $s = a^c$, $t = a^d$, $u = a^e$, $v = a^f$ mit $b + c + d + e + f = n^2$

Aus $|su| \geq 1$ folgt $c + e \geq 1$, und aus $|stu| \leq 2^m$ und $2^m \leq n$ folgt $c + d + e \leq 2^m \leq n$.

Wähle $i = 2$ und betrachte $rs^2tu^2v = a^{b+2c+d+2e+f} = a^{n^2+c+e}$ ($b + c + d + e + f = n^2$). Es kann gezeigt werden, dass a^{n^2+c+e} keine Quadratzahl ist, und somit $rs^2tu^2v \notin L$:

$$\begin{aligned} n^2 &< n^2 + c + e \quad (\text{da } c + e \geq 1) \\ &\leq n^2 + c + d + e \quad (|d| \geq 0) \\ &\leq n^2 + n \quad (\text{da } c + d + e \leq n) \\ &< n^2 + 2n + 1 \\ &= (n + 1)^2 \end{aligned}$$

$uv^2wx^2y \notin L_1$ und somit hat L nicht die kfr. PE!

[Zum Text](#)

Berechnung von $N[1,3]$

$$\begin{aligned}
 N^{(2)}[1,3] &\rightarrow 1. \text{ Zeile von } N^{(1)} \odot 3. \text{ Spalte von } N^{(1)} \\
 &= N[1,2] \odot N[2,3] \\
 &= \{A \in V \mid \text{Es gibt Regel } A \xrightarrow[G]{} BC \text{ mit } B \in N[1,2] \text{ und } C \in N[2,3]\} \\
 &= \{A \in V \mid A \xrightarrow[G]{*} w_1 w_2\}
 \end{aligned}$$

[Zum Text](#)

Beispiel zu 5.8 Das Wortproblem für kfr. Sprachen

Sei G eine kfr. Grammatik in CNF mit $V = \{S, A, B, C\}$, $\Sigma = \{a, b\}$ und R :

$$\begin{aligned}
 S &\rightarrow AB \mid BC \\
 A &\rightarrow BA \mid a \\
 B &\rightarrow CC \mid b \\
 C &\rightarrow AB \mid a
 \end{aligned}$$

Wir betrachten das Wort $w = baaba$ mit $n = 5$

Nebendiagonale	Zellen	Hinweise
	$N[1,2] = \{B\}$	
	$N[2,3] = \{A, C\}$	
1.	$N[3,4] = \{A, C\}$	Ablesen welche NTS auf w_1 bis w_5 abbilden
	$N[4,5] = \{B\}$	
	$N[5,6] = \{A, C\}$	
2.	$N[1,3] = \{S, A\}$	$N[1,3] = N[1,2] \odot N[2,3] \rightarrow$ alle NTS, die auf BA oder BC abbilden
	$N[2,4] = \{B\}$	
	$N[3,5] = \{S, C\}$	
	$N[4,6] = \{S, A\}$	
3.	$N[1,4] = \emptyset$	$N[1,4] = N[1,2] \odot N[2,4] \cup N[1,3] \odot N[3,4] \rightarrow$ alle NTS, die auf SA , SC , AA , AC oder BB abbilden
	$N[2,5] = \{B\}$	
	$N[3,6] = \{B\}$	
4.	$N[1,5] = \emptyset$	
	$N[2,6] = \{S, A, C\}$	
5.	$N[1,6] = \{\underline{S}, A, C\}$	

$(\mathcal{N}[1,4] = \emptyset$ bedeutet, es kann keine Ableitung für w mit $S \rightarrow XY$ und $X \xrightarrow{*} w_1w_2w_3$ geben) $\rightarrow S \in \mathcal{N}[1,6]$, d.h. $w \in L(G)$

Eine Ableitung von w findet man ebenfalls leicht.

\rightarrow Schaue rückwärts, wann S in die Menge $\mathcal{N}[1,6]$ gelegt wurde: Dies geschah wegen Regeln

$$S \rightarrow BC \rightarrow bAB \rightarrow baCC \rightarrow \dots \rightarrow baaba$$

[Zum Text](#)

einfache TMs

- a) $f : \Sigma^* \dashrightarrow \Sigma^*$, $\forall w : f(w) = \perp$ ist berechenbar. Die TM hat einfach $F = \emptyset$
- b) $L = \{w \in \{a,b\}^* \mid w \text{ enthält } aba\}$ ist regulär und wird von folgender TM akzeptiert:

$$\begin{aligned} K &= \{q_1, q_2, q_3, q_4\} \\ \Sigma &= \{a, b\} \\ \Gamma &= \Sigma \cup \{\square\} \\ F &= \{q_4\} \\ s &= q_1 \end{aligned}$$

δ :	state	read	move	write	move $\{L, N, R\}$
	K	Γ	K	Γ	
	q_1	a	q_2	*	(egal) R
	q_1	b	q_1	*	R
	q_1	\square	q_1	\square	N
	q_2	a	q_2	*	R
	q_2	b	q_3	*	R
	q_2	\square	q_2	\square	N
	q_3	a	q_4	*	*
	q_3	b	q_1	*	R
	q_3	\square	q_3	\square	N

[Zum Text](#)

TM die $L := \{w \in \{a,b\}^* \mid 2 \text{ teilt } |w|\}$ entscheidet

$$L := \{w \in \{a,b\}^* \mid 2 \text{ teilt } |w|\}$$

Turingmaschine $M := (K, \Sigma, \Gamma, s, F, \delta)$ für L mit $f(M) = \chi_L$:

$$\begin{aligned} K &:= \{q_0, q_1, q_2, q_3\} \\ \Sigma &:= \{a, b, 0, 1\} \\ \Gamma &:= \{a, b, 0, 1, \square\} \\ s &:= q_0 \\ F &:= \{q_3\} \end{aligned}$$

$\delta :$	K	Γ	K	Γ	$\{L, N, R\}$
	q_0	a	q_1	\square	R
	q_0	b	q_1	\square	R
	q_0	\square	q_2	1	R
	q_0	0	q_2	0	R
	q_0	1	q_2	0	R
	q_1	a	q_0	\square	R
	q_1	b	q_0	\square	R
	q_1	\square	q_2	0	R
	q_1	0	q_2	0	R
	q_1	1	q_2	0	R
	q_2	a	q_3	\square	L
	q_2	b	q_3	\square	L
	q_2	\square	q_3	\square	L
	q_2	0	q_3	\square	L
	q_2	1	q_3	\square	L

Kommentar:

Bei der Turingmaschine M entspricht der Zustand q_0 „gerade Anzahl gelesen“, und q_1 wiederum „ungerade Anzahl gelesen“ (M startet in q_0 , weil 0 gerade ist). In q_0 und q_1 wird dann immer ein Symbol nach dem anderen auf dem Band gelesen, nach rechts gegangen und in den jeweils anderen Zustand gewechselt.

- Wenn M 0 oder 1 liest, schreibt M „0 \square “ auf das Band und akzeptiert mit Lesekopf auf der 0, da $0, 1 \notin \{a, b\}$!
- Wenn M das Ende des Worts (Lesen von \square) in q_0 erreicht, dann schreibt M „1 \square “ auf das Band und wechselt in den akzeptierenden Endzustand q_3 (gerade Anzahl insgesamt gelesen).
- Wenn M das Ende des Worts (Lesen von \square) in q_1 erreicht, dann schreibt M stattdessen „0 \square “ auf das Band und wechselt trotzdem in den akzeptierenden Endzustand q_3 (ungerade Anzahl insgesamt gelesen).

Für eine Eingabe mit gerade Länge gibt M also 1 aus, und für eine Eingabe mit ungerade Länge stattdessen 0, also $f_M = \chi_L \checkmark$

Zum Text

Beispiel zu Def(f) und Bild(f)

Geben Sie berechenbare Funktionen f_1, f_2, f_3 und f_4 von Σ^* nach Σ^* an, die folgende Eigenschaften haben:

- $\text{Def}(f_1)$ nicht-entscheidbar und $\text{Bild}(f_1)$ entscheidbar:

$$f(x) = \begin{cases} 1 & , x \in H \\ \perp & , \text{sonst} \end{cases} \rightarrow \begin{array}{l} \text{Def}(f_1) = H \\ \text{Bild}(f_1) = \{1\} \end{array}$$

- $\text{Def}(f_2)$ nicht-entscheidbar und $\text{Bild}(f_2)$ nicht-entscheidbar:

$$f(x) = \begin{cases} x & , x \in H \\ \perp & , \text{sonst} \end{cases} \rightarrow \begin{array}{l} \text{Def}(f_2) = H \\ \text{Bild}(f_2) = H \end{array}$$

- $\text{Def}(f_3)$ entscheidbar und $\text{Bild}(f_3)$ nicht-entscheidbar:

$$f(M, w, t) = \begin{cases} (M, w) & , M \text{ hält auf } w \text{ innerhalb von } t \text{ Schritten} \\ 1 & , \text{sonst} \end{cases}$$

$\rightarrow \text{Def}(f_3) = \{\text{Code aller TMs}\} \times \Sigma^* \times \mathbb{N}$

$\rightarrow \text{Bild}(f_3) = H \cup \{1\}$

- $\text{Def}(f_4)$ entscheidbar und $\text{Bild}(f_4)$ entscheidbar:

$$f(x) = x(id_{\Sigma^*}) \rightarrow \begin{aligned} \text{Def}(f_4) &= \Sigma^* \\ \text{Bild}(f_4) &= \Sigma^* \end{aligned}$$

[Zum Text](#)

Reduktion beim Halteproblem

Beweis der Unentscheidbarkeit von H :

Hier gab es eine *mo*-Reduktion

$$H_1 \leq^{mo} H$$

mit

$$f : w \rightarrow (w, w)$$

[Zum Text](#)

Beispiel zu $a^n b^n$

$$L_1 = \{a^n b^n \mid n \geq 1\}, \quad \Sigma = \{a, b\}$$

L_2 = Halteproblem über $\{a, b\}$

Klar: $L_2 \not\leq^{mo} L_1$ wegen des Lemmas (H ist unentscheidbar).

$L_1 \leq^{mo} L_2$: Sei M eine TM, die L_1 akzeptiert. Zu Eingabe $w \in \{a, b\}^*$ berechnet f das Tupel (C_M, w)

[Zum Text](#)

Reduktion von Subset-Sum auf Lösen eines LGS

L_1 : Gegeben: $n \in \mathbb{N}, s_1, \dots, s_n \in \mathbb{Q}$

Frage: Gibt es eine Menge $S \subseteq \{1, \dots, n\}$ mit $\sum_{i \in S} s_i = \sum_{i \notin S} s_i$

L_2 : Gegeben: $\tilde{n}, \tilde{m} \in \mathbb{N}$, Matrix $A \in \mathbb{Q}^{\tilde{m} \times \tilde{n}}$, $b \in \mathbb{Q}^{\tilde{m}}$

Frage: Gibt es $x \in \{0, 1\}^{\tilde{n}}$ mit $Ax = b$?

$$L_1 \leq^{mo} L_2$$

Konstruktion von f : Zu Eingabe $(n, s_1, \dots, s_n) \in \mathbb{N} \times \mathbb{Q}^n$ muss f ein $(\tilde{n}, \tilde{m}, A, b)$ berechnen mit

$$(n, s_1, \dots, s_n) \in L_1 \Leftrightarrow (\tilde{n}, \tilde{m}, A, b) \in L_2$$

Wir setzen $\tilde{m} := 1, \tilde{n} := n$

$$A := (s_1, \dots, s_n), \quad b := \frac{1}{2} \sum_{i=1}^n s_i$$

Jetzt gilt: Die Subset-Sum Instanz ist lösbar mit einem $S \subseteq \{1, \dots, n\}$ gdw. die lineare Gleichung die Lösung hat:

$$x_i = \begin{cases} 1, & i \in S \\ 0, & \text{sonst} \end{cases}$$

Frage: Gilt auch $L_2 \leq^{mo} L_1$?

[Zum Text](#)

Beispiele für Eigenschaften

1. Die Eigenschaft, genau die Sprache $L := \{0^n 1^n \mid n \geq 1\}$ zu sein
2. Die Eigenschaft, kfr. zu sein
3. Die Eigenschaft, nur Wörter der Länge ≥ 25 zu enthalten
4. Die Eigenschaft, rek. aufzählbar zu sein: $E = \mathcal{RE}$
5. Die leere Eigenschaft: $E = \emptyset$
6. Die Eigenschaft $E = \{\emptyset\}$

[Zum Text](#)

Beispiel: Problem entscheidbar? Satz von Rice anwendbar?

1. Gegeben: Code C_M
Frage: Ist die Anzahl der Schritte, die M bei akzeptierenden Rechnungen ausführt, immer gerade?
Antwort: Keine Anwendung von Rice, unentscheidbar (selbst)
2. Gegeben: Code C_M
Frage: Sind alle Wörter in $L(M)$ von gerader Länge?
Antwort: Unentscheidbar mit Rice
3. Gegeben: Code C_M
Frage: Hält M nur auf endlich vielen Wörtern?
Antwort: Unentscheidbar mit Rice
4. Gegeben: Code C_M
Frage: Schreibt M bei irgendeiner Rechnung ein a aufs Band?
Antwort: Keine Anwendung von Rice, Unentscheidbar
5. Gegeben: Code C_M

Frage: Ist $L(M)$ rek. aufzählbar?

Antwort: Rice nicht anwendbar, da Eigenschaft trivial ist. Problem ist entscheidbar.

[Zum Text](#)

Beispiel zu 6.13 Kontextsensitive Grammatiken

Grammatik G :

$$\begin{aligned} V &= \{S, A, B\}, \\ \Sigma &= \{a, b, c\}, \\ R &= \{ \begin{array}{l} S \rightarrow aAbc \mid abc \\ \quad A \rightarrow aAbB \mid abB \\ \quad Bb \rightarrow bB \\ \quad Bc \rightarrow cc \end{array} \} \end{aligned}$$

$L(G) = \{a^n b^n c^n \mid n \geq 1\}$ ist nich kontextfrei (siehe **Kontextfreies Pumping-Lemma**), aber kontextsensitiv

[Zum Text](#)

Beispiel zu 7.1 Komplexitätsklasse P

Polynomiell beschränkte Laufzeiten sind z.B. $2n^2, cn, 3n^3 + 2n^2 - n, n \cdot \log(n)$
 2^n ist nicht polynomiell beschränkt

[Zum Text](#)

Hamiltonkreis

Sei ein Graph $G = (V, E)$ mit Ecken (Knoten) $V = \{1, \dots, n\}$ und ungerichteten Kanten $E \subseteq V^2$ gegeben.

Ein **Hamiltonkreis HK** in G ist eine Permutation (bijektive Abbildung) $\pi : V \rightarrow V$ mit $(\pi(i), \pi(i+1)) \in E$ sowie $(\pi(n), \pi(1)) \in E$ (Rundreise, wo man alle Knoten genau 1x überschreitet, und im Startkonten endet).

Entscheidungsproblem **HK₁**:

Gegeben: Graph $G = (V, E)$ und Permutation π von V .

Frage: Ist π ein Hamiltonkreis in G ?

Behauptung: **HK₁ ∈ P**

Eingabegröße: Was ist ca. die Eingabegröße einer Instanz (G, π) von **HK₁**?

→ Kodierungslänge für G : $\mathcal{O}(n^2 \cdot \log(n))$

(Ein Graph mit n Knoten kann maximal $\frac{(n-1) \cdot n}{2}$ Kanten haben ($\in \mathcal{O}(n^2)$), jede Knoten benötigt eine Kodierung mit $\log(n)$ vielen Bits, und eine Kante wäre darstellbar durch 2 Knoten + 0 oder 1 Bit für die Frage $\in E$.)

→ Kodierungslänge für π : $\mathcal{O}(n \cdot \log(n))$

(n Knoten mit jeweils $\log(n)$ Bit Kodierung.)

Algorithmus: Prüfe für jedes $i = 0, 1, \dots, n - 1$ ob $(\pi_{(i)}, \pi_{(i+1)}) \in E$ sowie ob $(\pi_{(n)}, \pi_{(1)}) \in E$. Falls dies für alle i gilt, dann akzeptiere (G, π) , sonst nicht.

\rightsquigarrow Laufzeit insgesamt polynomiell, beschränkt durch $\mathcal{O}(n^3 \cdot \log(n))$
(n viele Kanten für die man $|G, \pi| \in \mathcal{O}(n^2 \cdot \log(n))$ nachschlägt)

HK:

Gegeben: Graph $G = (V, E)$.

Frage: Hat G einen Hamiltonkreis?

Algorithmus: **HK** ist entscheidbar:

Berechne alle $\frac{(n-1)!}{2}$ in Frage kommenden Permutationen (es gibt $n!$ Permutationen, aber Wahl des Startknoten egal $((n-1)!)$ und „Laufrichtung“ auch egal $(1/2)$) und prüfe für jede die entsprechende **HK1**-Instanz

\rightsquigarrow exponentielle Laufzeit wegen $\mathcal{O}(n!)$.

Gilt $\text{HK} \in P$?

Bemerkung

Beachte: Wenn man eine Permutation vorliegen hat, kann man „schnell“ verifizieren, ob π einen HK darstellt.

[Zum Text](#)

Beispiel zu 7.2 Erfüllbarkeitsproblem

$$\phi(x_1, x_2, x_3, x_4) \equiv (\overline{x_1} \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3} \vee \overline{x_4}) \wedge x_4$$

$$\rightarrow \phi(1, 0, 0, 1) = 0 \wedge 1 \wedge 1 \wedge 1 = 0$$

[Zum Text](#)

Beispiel zu 7.2.2 k-SAT

- $1-SAT \in P$: Prüfe ob es Literale x_i sowie $\overline{x_i}$ gibt $\begin{cases} \text{Ja} & \rightarrow \text{unerfüllbar} \\ \text{Nein} & \rightarrow \text{erfüllbar} \end{cases}$
- $2-SAT \in P$: komplizierter, aber lösbar in Polynomzeit
- $3-SAT \in ??$

[Zum Text](#)

$HK \in NP$ und $k-SAT \in NP$

$HK \in NP$: Eine NDTM M erzeugt („räte“) zunächst mittels nicht-deterministischer Operationen Rechenpfade, die immer zwei Abzweigungen haben (0 oder 1). Der so entstandene Rechenpfad repräsentiert einen Vektor der Länge $\lceil n \cdot \log(n) \rceil$ (n Knoten mit Kodierung $\log(n)$), welcher eine Permutation der n Ecken kodieren soll. Danach löst M deterministisch eine entsprechende HK_1 -Instanz. M erfüllt $i) - iv)$

$k-SAT \in NP$: Eine NDTM M erzeugt nicht-det. eine Belegung aus $\{0,1\}^n$ und wendet det. auf diese den SAT_1 -Algorithmus an

[Zum Text](#)

Polynomialzeit Reduktion von Subset Sum auf Bin-Packing

1) Subset-Sum $\in NP$

Eingabe:

$n \in \mathbb{N}, C_1, \dots, C_n \in \mathbb{N}$

Frage:

Gibt es $S \subseteq \{1, \dots, n\}$ mit $\sum_{i \in S} C_i = \sum_{i \notin S} C_i$

Verifikationsalgorithmus:

„räte“ nicht-deterministisch eine Teilmenge S , berechnet die beiden Summen und vergleicht.

Laufzeit:

Eingabegröße $\rightarrow \mathcal{O}\left(n \cdot \max_{i \in \{1, \dots, n\}} \{\lceil \log(C_i) \rceil\}\right)$ (die Eingabegröße hängt nicht nur von der Anzahl (n), sondern auch dem numerischen Wert ab (C_i)).
Addition geht effizient.

2) Bin-Packing $\in NP$

Eingabe:

$n \in \mathbb{N}, B \in \mathbb{N}$ (Maximalgewicht), $k \in \mathbb{N}$ (Anzahl Kartons), $C_1, \dots, C_n \in \mathbb{N}$ (Gewichte)

Frage:

Gibt es eine **Partition** S_1, \dots, S_k von $\{1, \dots, n\}$ mit $\forall k \left(\sum_{i \in S_k} C_i \right) \leq B$

Verifikationsalgorithmus:

„räte“ nicht-deterministisch die k vielen Teilmengen S_1, \dots, S_k , prüft die Eigenschaft der Partition und ob die jeweiligen Summen $\leq B$ bleiben.

Laufzeit:

Eingabegröße $\rightarrow \mathcal{O}(n \cdot \log(n) \cdot \lceil \log(B) \rceil)$ (O.E. kann man $C_i \leq B$ (sonst unerfüllbar) und $k \leq n$ (mehr Kartons als Gewichte ist unsinnig), woraus folgt $\max_{i \in \{1, \dots, n\}} \{\lceil \log(C_i) \rceil, \lceil \log(B) \rceil\} = \lceil \log(B) \rceil$ und $k \in \mathcal{O}(n)$).
Addition geht effizient.

Polynomialzeit Reduktion:

\rightarrow Eingabe Subset-Sum: n, C_1, \dots, C_n

\rightarrow Konstruktion einer Bin-Packung Instanz $\tilde{n}, B, k, \tilde{C}_1, \dots, \tilde{C}_n$:

- $\tilde{n} := n$
- $k := 2$
- $B := \frac{1}{2} \cdot \sum_{i=1}^n C_i$
- $\tilde{C}_i := C_i \ \forall i$

[Zum Text](#)

Beispiel 1

$$(A \in P, B \neq \emptyset, B \neq \Sigma^*) \implies A \preceq_p^{mo} B$$

$$B \neq \emptyset \implies \exists w_1 \in B$$

$$\begin{aligned}
 B \neq \Sigma^* &\implies \exists w_2 \notin B \\
 A \leq_p^{mo} B : \forall x \in \Sigma^* : x \in A &\Leftrightarrow f(x) \in B \\
 \rightarrow f(x) = \begin{cases} w_1 & x \in A \\ w_2 & x \notin A \end{cases}
 \end{aligned}$$

(f ist in Polynomialzeit berechenbar, denn wegen $A \in P$ ist das Entscheiden $x \in A$ in Polynomzeit möglich, und anschließend wird davon abhängig ein Wort zurückgegeben, was in konstanter Zeit möglich ist.)

[Zum Text](#)

Beispiel 2

$$(A \in NPC \wedge NP \neq co-NP) \implies \underbrace{\{1\} \cdot A \cup \{2\} \cdot \overline{A}}_L \notin NP \cup co-NP$$

$$(co-NP = \{A \mid \overline{A} \in NP\})$$

$$A \in NPC \wedge NP \neq co-NP \implies A \notin co-NP$$

(Aus $A \in NPC$ und $A \in co-NP$ folgt $NP = co-NP$ (siehe Beweis am Ende), was einen Widerspruch zu $NP \neq co-NP$ darstellt)

$$A \notin co-NP \implies \overline{A} \notin NP$$

$L \notin NP$:

$$\overline{A} \notin NP \implies L \notin NP \quad (\overline{A} \leq_p^{mo} L \text{ mit } f(x) = 2 \cdot x)$$

$L \notin co-NP$:

$$A \notin co-NP \implies L \notin co-NP \quad (A \leq_p^{mo} L \text{ mit } f(x) = 1 \cdot x)$$

$$L \notin NP \wedge L \notin co-NP \implies L \notin NP \cup co-NP$$

Beweis für $A \in NPC \wedge A \in co-NP \implies NP = co-NP$

[Zum Text](#)

Beispiel 3

$$NP \subseteq REC$$

Ein Problem L liegt in NP, wenn es eine NDTM M gibt, die in polynomieller Zeit entscheidet, ob eine Eingabe in der Sprache liegt. Da eine NDTM immer hält, kann man alle Rechenpfade von M nacheinander mit einer deterministischen TM simulieren, womit die charakteristische Funktion von L durch eine deterministische TM berechnet werden kann.

[Zum Text](#)

Beispiel 1

Zeigen von $\mathcal{O}(\log_2) = \mathcal{O}(\log_{10})$

$$\lim_{n \rightarrow \infty} \frac{\log_2(n)}{\log_{10}(n)} = \lim_{n \rightarrow \infty} \log_2(10) < \infty \checkmark$$

Oder:

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{\log_2(n)}{\log_{10}(n)} &= \lim_{n \rightarrow \infty} \frac{\frac{\ln(n)}{\log_e(2)}}{\frac{\ln(n)}{\log_e(10)}} \\ &= \frac{\log_e(10)}{\log_e(2)} \\ &= \log_2(10) < \infty \checkmark \end{aligned}$$

[Zum Text](#)

Beispiel 2

Zeigen von $\mathcal{O}(\log(n + \log(n))) = \mathcal{O}(\log(n))$

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{\log(n + \log(n))}{\log(n)} &= \lim_{n \rightarrow \infty} \frac{\log\left(n \cdot \left(1 + \frac{\log(n)}{n}\right)\right)}{\log(n)} \\ &= \lim_{n \rightarrow \infty} \frac{\log(n) + \log\left(1 + \frac{\log(n)}{n}\right)}{\log(n)} \\ &= \lim_{n \rightarrow \infty} \frac{\log(n)}{\log(n)} + \frac{\log\left(1 + \frac{\log(n)}{n}\right)}{\log(n)} \\ &= 1 + 0 \\ &= 1 < \infty \checkmark \end{aligned}$$

Oder:

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{\log(n + \log(n))}{\log(n)} &\stackrel{L'Hospital}{=} \lim_{n \rightarrow \infty} \frac{\frac{1}{n+log(n)} \cdot \left(1 + \frac{1}{n}\right)}{\frac{1}{n}} \\ &= \lim_{n \rightarrow \infty} \frac{n \cdot \left(1 + \frac{1}{n}\right)}{n + \log(n)} \\ &= \lim_{n \rightarrow \infty} \frac{n \cdot \left(1 + \frac{1}{n}\right)}{n \cdot \left(1 + \frac{\log(n)}{n}\right)} \\ &= 1 < \infty \checkmark \end{aligned}$$

[Zum Text](#)

Beispiel 3

Ist die charakteristische Funktion χ_A der Sprache A auf Eingaben der Länge n in Zeit $\mathcal{O}(\log(n!))$ berechenbar, so liegt A in P.

$$\underbrace{\mathcal{O}(\log(n!))}_{\text{Stirling-Formel}} = \mathcal{O}\left(\log\left(\left(\frac{n}{e}\right)^n\right)\right) = \mathcal{O}(\log((n^n))) = \mathcal{O}(n \cdot \log(n)) < \mathcal{O}(n^2) \implies A \in P$$

10.2 Beweise

Beweis zu Σ^* ist abzählbar unendlich

Beweis mittels Widerspruch:

Annahme: Σ^* ist endlich

Dann enthält Σ^* ein Wort w maximaler Länge. Sei a ein Zeichen aus dem Alphabet Σ . Dann ist wa ein Wort über Σ , d.h. $wa \in \Sigma^*$, aber andererseits $wa \notin \Sigma^*$, da Σ^* nur Wörter der Länge $\leq |w|$ enthält und $|wa| > |w|$ gilt. Dies ist ein Widerspruch; also ist die Annahme falsch.

Um zu zeigen, dass Σ^* abzählbar unendlich viele Wörter enthält, benötigen wir eine bijektive Abbildung zwischen Σ^* und \mathbb{N} . Die bijektive Abbildung zwischen Σ^* und \mathbb{N} ergibt sich, indem jedes Wort von Σ^* auf die Position abgebildet wird, in der es in der **längen-lexikographischen** Ordnung (Ordnung erst nach Länge, und bei gleicher Länge lexikographisch) auftritt.

□

[Zum Text](#)

Beweis zu Menge aller Sprachen ist überabzählbar unendlich

Die Menge aller Wörter Σ^* ist abzählbar unendlich (siehe vorheriger Beweis), und eine Sprache L ist eine Teilmenge von Σ^* . Die Menge aller Sprachen ist also die Menge aller Teilmengen von Σ^* bzw. $\mathcal{P}(\Sigma^*)$ (Potenzmenge), und die Potenzmenge von einer Menge ist immer mächtiger als die Menge selbst (**Satz von Cantor (Potenzmenge)**). Also ist die Menge aller Sprachen überabzählbar unendlich.

□

[Zum Text](#)

Beweis zu Menge der regulären Sprachen ist abzählbar unendlich

Die Menge der regulären Sprachen ist abzählbar unendlich, denn damit damit eine Sprache L regulär ist, muss es einen regulären Ausdruck x mit $\mathcal{L}(x) = L$, und die Menge der regulären Sprachen ist abzählbar unendlich:

- Möglichkeit 1:

Ein regulärer Ausdruck ist ein Wort über dem endlichen Alphabet $\Sigma \cup \{\cup, ., *\}$ und die Menge aller Wörter über einem endlichen Alphabet ist abzählbar unendlich.

- Möglichkeit 2:

Die Menge der regulären Ausdrücke ist gleichmächtig zu \mathbb{N} , denn man kann die regulären Ausdrücke **längen-lexikografisch** anordnen, und dann jeden Ausdruck auf sein Platz in der Ordnung abbilden, was einer Bijektion auf \mathbb{N} entspricht.

□

[Zum Text](#)

Beweis

$L = (\{1\}.H_1 \cup \{0\}.\overline{H_1})$ ist nicht rekursiv aufzählbar, und nicht co-rekursiv aufzählbar:

- $L \notin RE$: $\overline{H_1} \leq_{mo} L$ mit $f(w) = 0.w$:

- $w \in \overline{H_1} \implies f(w) = 0.w \in L$
- $w \notin \overline{H_1} \implies f(w) = 0.w \notin L$ (müsste $1.w$ sein)
- $L \notin co-RE: H_1 \leq_{mo} L$ mit $f(w) = 1.w$:
 - $w \in H_1 \implies f(w) = 1.w \in L$
 - $w \notin H_1 \implies f(w) = 1.w \notin L$ (müsste $0.w$ sein)

□

Zum Text**Beweis**

$L = H_1 \times \overline{H_1}$ ist nicht rekursiv aufzählbar, und nicht co-rekursiv aufzählbar:

Sei $x \in H_1$, $y \in \overline{H_1}$ fest.

- $L \notin RE: \overline{H_1} \leq_{mo} L$ mit $f(w) = (x, w)$:
 - $w \in \overline{H_1} \implies f(w) = (x, w) \in L$
 - $w \notin \overline{H_1} \implies f(w) = (x, w) \notin L$ (kein valides Paar, da $w \notin \overline{H_1}$)
- $L \notin co-RE: H_1 \leq_{mo} L$ mit $f(w) = (w, y)$:
 - $w \in H_1 \implies f(w) = (w, y) \in L$
 - $w \notin H_1 \implies f(w) = (w, y) \notin L$ (kein valides Paar, da $w \notin H_1$)

□

Zum Text**Beweis für \emptyset und Σ^* sind nicht NP-vollständig**

$\emptyset \notin NPC$ Angenommen $\emptyset \in NPC$. Dann folgt: $\forall A \in NP: A \preceq_p^{\text{mo}} \emptyset$

Die Reduktion \preceq_p^{mo} wiederum bedeutet, dass es eine in Polynomzeit berechenbare Funktion f gibt mit:

$$\forall x \in \Sigma^*: x \in A \Leftrightarrow f(x) \in \emptyset$$

\emptyset enthält keine Wörter, also $f(x) \notin \emptyset$ für alle x . Aus $\emptyset \in NPC$ würde also folgen, dass alle Sprachen in NP leer sind, was offensichtlich falsch ist. \emptyset kann somit nicht NP-vollständig sein!

$\Sigma^* \notin NPC$ Angenommen $\Sigma^* \in NPC$. Dann folgt: $\forall A \in NP: A \preceq_p^{\text{mo}} \Sigma^*$

Die Reduktion \preceq_p^{mo} wiederum bedeutet, dass es eine in Polynomzeit berechenbare Funktion f gibt mit:

$$\forall x \in \Sigma^*: x \in A \Leftrightarrow f(x) \in \Sigma^*$$

Σ^* enthält alle Wörter, also $f(x) \in \Sigma^*$ für alle x . Aus $\Sigma^* \in NPC$ würde also folgen, dass alle Sprachen in NP keine Nein-Instanzen besitzen, was offensichtlich falsch ist. Σ^* kann somit nicht NP-vollständig sein!

□

Zum Text**Beweis für $A \in NPC \wedge A \in co-NP \implies NP = co-NP$**

„ $NP \subseteq co-NP$ “ Aus $A \in NPC$ folgt $A \in NP$ und $\forall L \in NP : L \preceq_p^{mo} A$

$$L \preceq_p^{mo} A \implies \bar{L} \preceq_p^{mo} \bar{A}$$

$$\bar{L} \preceq_p^{mo} \bar{A} \wedge A \in co-NP (\bar{A} \in NP) \implies \bar{L} \in NP$$

$$\forall L \in NP : \bar{L} \in NP \implies L \in co-NP \implies NP \subseteq co-NP$$

„ $NP \supseteq co-NP$ “ $\forall L \in co-NP : \underbrace{\bar{L} \in NP \implies \bar{L} \in co-NP}_{NP \subseteq co-NP \text{ oben bereits gezeigt}} \implies L \in NP$

$$\implies co-NP \subseteq NP \implies NP = co-NP$$

□

[Zum Text](#)

10.3 „Wahr oder Falsch“-Fragen

W | F 1

Sei $L \subseteq \{0,1\}^*$ co-endlich (d.h. \bar{L} ist endlich). Dann \exists DEA M mit $L(M) = L$?

[Zur Lösung](#) [Zum Text](#)

W | F 2

Die Menge der DEAs und die Menge der formalen Sprachen ist gleichmächtig?

[Zur Lösung](#) [Zum Text](#)

W | F 3

Sei $\Sigma = \{0,1\}$, A die Menge der durch DEAs akzeptierten Sprachen, B die Menge der DEAs (beides über Σ^*). Zu jedem $L \in A$ gibt es unendlich viele $M \in B$ mit $L = L(M)$?

[Zur Lösung](#) [Zum Text](#)

W | F 4

Sei $\Sigma = \{0,1\}$, A die Menge der durch DEAs akzeptierten Sprachen, B die Menge der DEAs (beides über Σ^*). Es gibt eine surjektive Abbildung $A \rightarrow B$?

[Zur Lösung](#) [Zum Text](#)

W | F 5

Zu jeder regulären Sprache L gibt es einen DEA M mit genau einem Endzustand und $L = L(M)$?

[Zur Lösung](#) [Zum Text](#)

W | F 6

Sei M ein DEA, $R(i,j,k)$ wie in **Satz von Kleene** definiert. Dann gilt $\forall i, j, k_1, k_2 : k_1 \neq k_2 \implies R(i,j,k_1) \neq R(i,j,k_2)$?

[Zur Lösung](#) [Zum Text](#)

W | F 7

$\forall n \in \mathbb{N} \exists \text{ÄR auf } \Sigma^*, \text{ sodass } R \text{ genau } n \text{ ÄK hat?}$

[Zur Lösung](#) [Zum Text](#)

W | F 8

Sei L regulär, dann gilt: $\exists n \in \mathbb{N} \forall w \in L$ mit $|w| \geq n \exists x, y_1, y_2, z \in \Sigma^*$ mit:

1. $w = x.y_1.y_2.z$
 2. $|y_1| \geq 1, |y_2| \geq 1$
 3. $|xy_1y_2| \leq n$
 4. $\forall i \in \mathbb{N}_0 : xy_1^i y_2^i z \in L$
- ?

[Zur Lösung](#) [Zum Text](#)

W | F 9

Ein $L \subseteq \Sigma^*$ ist genau dann regulär, wenn jede ÄK bezüglich L endlich ist?

[Zur Lösung](#) [Zum Text](#)

W | F 10

Es gibt nicht-reguläre Sprachen L so, dass L endliche ÄK bezüglich L hat?

[Zur Lösung](#) [Zum Text](#)

W | F 11

Sei L nicht regulär, L' regulär. Dann ist $L \setminus L'$ nicht regulär?

[Zur Lösung](#) [Zum Text](#)

W | F 12

Sei L nicht regulär, $L' \subseteq L$ regulär. Dann ist $L \setminus L'$ nicht regulär?

[Zur Lösung](#) [Zum Text](#)

W | F 13

Sei $h : \Sigma^* \rightarrow \Sigma^*$. $\forall a \in \Sigma$ gilt $h^{-1}(\{a\}) \neq \emptyset$?

[Zur Lösung](#) [Zum Text](#)

W | F 14

Sei Hom die Menge aller Homomorphismen von $\Sigma^* \rightarrow \Sigma^*$. Sei L regulär, dann $\exists h \in Hom$ mit $h(\Sigma^*) = L$?

[Zur Lösung](#) [Zum Text](#)

W | F 15

Sei Hom die Menge aller Homomorphismen von $\Sigma^* \rightarrow \Sigma^*$.

$\forall h \in Hom$ gilt: $(\exists g \in Hom \forall w \in \Sigma^* g \circ h(w) = w) \implies h$ ist injektiv?

[Zur Lösung](#) [Zum Text](#)

W | F 16

Sei Σ ein endliches Alphabet. Die Menge $\{(G, L(G)) \mid G \text{ ist kfr. Grammatik über } \Sigma\}$ ist abzählbar?

[Zur Lösung](#) [Zum Text](#)

W | F 17

Sei G kfr. Grammatik. Dann gibt es einen Algorithmus, der entscheidet ob $\lambda \in G$?

[Zur Lösung](#) [Zum Text](#)

W | F 18

Jede kfr. Sprache wird von einem PDA mit ≤ 2 Zuständen erkannt?

[Zur Lösung](#) [Zum Text](#)

W | F 19

Sei M PDA. Konstruiere gemäß VL äquivalente kfr. Grammatik und aus dieser wiederum einen äquivalenten PDA M' . Dann sind M und M' gleich?

[Zur Lösung](#) [Zum Text](#)

W | F 20

Sei $L \neq \emptyset$ kfr. Dann gibt es disjunkte Zerlegung $L = L_1 \cup L_2$ mit L_1 kfr. und L_2 regulär ($L_2 \neq \emptyset$)?

[Zur Lösung](#) [Zum Text](#)

W | F 21

Neues Modell eines PDA → man darf an beiden Enden des Kellers arbeiten. PDA^{neu} kann mehr als ein PDA?

[Zur Lösung](#) [Zum Text](#)

W | F 22

Sei $N := \{f \mid f : \mathbb{N} \rightarrow \mathbb{N} \text{ total}\}$, $E := \{f \mid f : \{a,b\}^* \rightarrow \{a,b\}^* \text{ total}\}$. Dann gibt es eine Bijektion von N nach E ?

[Zur Lösung](#) [Zum Text](#)

W | F 23

Sei M TM auf Σ^* , dann ist $g : \Sigma^* \dashrightarrow \Sigma^*$, $g(w) := \begin{cases} 1 & , w \in L(M) \\ \perp & , \text{ sonst} \end{cases}$ (semi-charakteristische Funktion von $L(M)$) berechenbar?

[Zur Lösung](#) [Zum Text](#)

W | F 24

Jedes $f : \mathbb{N} \rightarrow \mathbb{N}$ total ist berechenbar?

[Zur Lösung](#) [Zum Text](#)

W | F 25

\mathbb{Q} ist rekursiv aufzählbar?

[Zur Lösung](#) [Zum Text](#)

W | F 26

$\forall L \subseteq \Sigma^* \exists f : \Sigma^* \dashrightarrow \Sigma^* \text{ mit } f(\Sigma^*) = \overline{L}$?

[Zur Lösung](#) [Zum Text](#)

W | F 27

 $L_1 \leq^{mo} L_2 \Leftrightarrow L_2 \leq^{mo} L_1?$ [Zur Lösung](#) [Zum Text](#)

W | F 28

Sei f eine many-one Reduktion von L_1 auf L_2 . Was stimmt? f ist nie/ injektiv/
 immer surjektiv[Zur Lösung](#) [Zum Text](#)

W | F 29

Gelte $L_1 \leq^{mo} L_2$ und sei L_2 entscheidbar. Was folgt?[Zur Lösung](#) [Zum Text](#)

W | F 30

Gegeben TM M_1, M_2 . Ist das Problem „existiert ein Wort, das beide TMs akzeptieren“ entscheidbar?[Zur Lösung](#) [Zum Text](#)

W | F 31

Seien Σ, Γ fest. Gegeben eine TM M_1 mit ≤ 10 Zuständen, und eine TM M_2 mit ≤ 20 Zuständen. Ist das Problem „gilt $L(M_1) = L(M_2)$ “ entscheidbar?[Zur Lösung](#) [Zum Text](#)

W | F 32

Seien L_1 und L_3 entscheidbar, L_2 unentscheidbar. Falls $L_1 \leq_{mo} L_2$ und $L_2 \leq_{mo} L_3$, dann $L_2 \leq_{mo} L_1$?[Zur Lösung](#) [Zum Text](#)

W | F 33

Seien $f : \Sigma^* \rightarrow \Sigma^*$ total sowie berechenbar, und $L \subseteq \Sigma^*$. Dann ist f eine many-one Reduktion von L auf $f(L)$?[Zur Lösung](#) [Zum Text](#)

W | F 34

Seien p Polynom und M NDTM, die für jedes $x \in \Sigma^*$ bei der Rechnung auf $x \leq p(|x|)$ Zellen benutzt (auch wenn sie nicht hält). Dann ist $L(M)$ entscheidbar?[Zur Lösung](#) [Zum Text](#)

W | F 35

Sei $L \subseteq \Sigma^*$, dann gibt es eine Zerlegung $L = \bigcup_{i=1}^{\infty} L_i$ mit $L_i \in P \forall i \in \mathbb{N}$?[Zur Lösung](#) [Zum Text](#)

W | F 36

Sei $L \subseteq \Sigma^*$ regulär, $A \subseteq \Sigma^*$ eine ÄK bezüglich \approx_L . Dann gilt $A \in P$?

[Zur Lösung](#) [Zum Text](#)

W | F 37

Die aussagenlogische Formel $x \Leftrightarrow y \vee z$ ist äquivalent zu einer 3-SAT-Formel in x, y, z ?

[Zur Lösung](#) [Zum Text](#)

W | F 38

Jede boolesche Funktion $f : \{0, 1\}^n \rightarrow \{0, 1\}$ kann als SAT-Formel Φ in x_1, \dots, x_n geschrieben werden (d.h. $f(x) = 1 \Leftrightarrow \Phi(x) = 1$)?

[Zur Lösung](#) [Zum Text](#)

W | F 39

Seien $L_1, L_2 \in P$, dann gilt $L_1 \leq_p^{mo} L_2$?

[Zur Lösung](#) [Zum Text](#)

W | F 40

Folgendes Problem liegt in NP?

Geg.: $n, m \in \mathbb{N}$, $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$

Frage: Gibt es ein $x \in \mathbb{Q}^n$ mit $Ax = b$?

[Zur Lösung](#) [Zum Text](#)

W | F 41

Es gibt $f : \mathbb{N} \rightarrow \mathbb{N}$ so, dass f nicht durch ein Polynom beschränkt ist und $2^n \notin \mathcal{O}(f)$?

[Zur Lösung](#) [Zum Text](#)

W | F 42

Seien A, B, C rekursiv aufzählbar, dann ist $A \cap C \setminus B \cap C$ rekursiv aufzählbar?

[Zur Lösung](#) [Zum Text](#)

W | F 43

$\exists L \subseteq \Sigma^*$ so, dass die # ÄK zu \approx_L überabzählbar ist.

[Zur Lösung](#) [Zum Text](#)

W | F 44

Wahrheitstafel für $(A \Rightarrow B) \Rightarrow C$

[Zur Lösung](#) [Zum Text](#)

W | F 45

$\forall i \in \mathbb{N}$ sei $L_i \subseteq \Sigma^i$, $L_i \in NP$. Definiere $L = \bigcup_{i \in \mathbb{N}} L_i$, dann ist $L \in NP$?

[Zur Lösung](#) [Zum Text](#)

W | F 46

Seien A, B so, dass $A \cup B$ regulär ist. Dann gilt $A \leq_{poly}^{mo}$ SAT?

[Zur Lösung](#) [Zum Text](#)

W | F 47

Sei Σ ein endliches Alphabet und $L = \Sigma$. Ist L eine reguläre Sprache?

[Zur Lösung](#) [Zum Text](#)

W | F 48

Die Menge der Reellen Zahlen ist regulär?

[Zur Lösung](#) [Zum Text](#)

W | F 49

Sei $L \subseteq \Sigma^*$, M DEA mit $L = L(M)$ und $a \in \Sigma$. Dann wird auch $L.\{a\}$ von einem DEA akzeptiert.

[Zur Lösung](#) [Zum Text](#)

W | F 50

Folgende Abbildung ist injektiv, surjektiv, bijektiv: $f : \text{Menge der DEAs} \rightarrow \mathcal{P}(\Sigma^*)$ mit $M \mapsto L(M)$

[Zur Lösung](#) [Zum Text](#)

W | F 51

Folgende Abbildung ist injektiv, surjektiv, bijektiv
 $f : \text{Menge der regulären Sprachen} \rightarrow \text{Menge der regulären Ausdrücke}$
 $L \mapsto x$ mit $\mathcal{L}(x) = L$

[Zur Lösung](#) [Zum Text](#)

W | F 52

L endlich $\Rightarrow L$ erfüllt reg. PE

[Zur Lösung](#) [Zum Text](#)

W | F 53

$x \approx y \Rightarrow \forall z \in \Sigma^* : zx \in L \Leftrightarrow yz \in L$ ist ÄR

[Zur Lösung](#) [Zum Text](#)

W | F 54

Jede endliche Sprache hat endlich viele ÄK bzgl. \approx_L

[Zur Lösung](#) [Zum Text](#)

W | F 55

Zu jedem $L \subseteq \Sigma^*$ sind alle ÄK bzgl. \approx_L entweder endlich oder alle unendlich.

[Zur Lösung](#) [Zum Text](#)

W | F 56

$f : \Sigma^* \rightarrow \Sigma^*$ bijektiv. Wenn $L \subseteq \Sigma^*$ regulär ist, dann ist auch $f(L)$ regulär.

[Zur Lösung](#) [Zum Text](#)

W | F 57

Sei $L \subseteq \Sigma^*$ regulär. Dann ist folgende Aufgabe algorithmisch lösbar?

Eingabe: $x, y \in \Sigma^*$

Frage: Gilt $x \approx_L y$?

[Zur Lösung](#) [Zum Text](#)

W | F 58

Sei $\Sigma = \{a, b, c\}$, $L := \{a, b\}^* \cup \{a^k b^k c^n \mid k, n \geq 1\}$. Dann ist L nicht regulär.

[Zur Lösung](#) [Zum Text](#)

W | F 59

Sei $h : \Sigma^* \rightarrow \Sigma^*$ homomorph, $L \subseteq \Sigma^*$ nicht regulär. Dann ist $h(L)$ auch nicht regulär.

[Zur Lösung](#) [Zum Text](#)

W | F 60

Sei $L = \{w \in \{0, 1\}^* \mid \#_0(w) \text{ ist gerade}\}$. Dann gibt es ein $h : \{0, 1\}^* \rightarrow \{0, 1\}^*$ mit $h(L) = \{0, 1\}^+$.

[Zur Lösung](#) [Zum Text](#)

W | F 61

Sei $L_1, L_2 \subseteq \Sigma^*$ kontextfrei und $h : \Sigma^* \rightarrow \Sigma^*$ homomorph. Dann ist $h(L_1) \cup h(L_2)$ kontextfrei.

[Zur Lösung](#) [Zum Text](#)

W | F 62

Sei $h : \Sigma^* \rightarrow \Sigma^*$ homomorph. Dann gibt es $\tilde{h} : \Sigma^* \rightarrow \Sigma^*$ homomorph mit $\tilde{h} \circ h = \text{id}$
 $(\text{id} : \Sigma^* \rightarrow \Sigma^*, \text{id}(x) = x \quad \forall x \in \Sigma^*)$

[Zur Lösung](#) [Zum Text](#)

W | F 63

Sei $L \subseteq \Sigma^*$ regulär. Dann gibt es ein Kellerautomat M mit $L = L(M)$ und die Überführungsrelation Δ von M ist eine Funktion.

[Zur Lösung](#) [Zum Text](#)

W | F 64

Sei G rechts-lineare Grammatik für a^*b^* , M_{kfr} die Menge aller kontextfreien Grammatiken, M_{rl} die Menge der rechtslinearen Grammatiken. Dann gibt es ein $f : M_{kfr} \rightarrow M_{rl}$ mit $f(M_{kfr}) = M_{rl} \setminus \{G\}$ und f injektiv.

[Zur Lösung](#) [Zum Text](#)

W | F 65

Seien $L = L(M)$ für PDS M und $\tilde{L} \subseteq L$. Dann gibt es kfr. G mit $L(G) = L \setminus \tilde{L}$.

Zur Lösung Zum Text**W | F 66**

Zu jedem regulären L gibt es eine Grammatik G , die nicht kontextfrei ist, mit $L = L(G)$.

Zur Lösung Zum Text**W | F 67**

Seien L_1 kontextfrei, L_2 regulär. Dann ist $L_1 \cap L_2$ kontextfrei.

Zur Lösung Zum Text**W | F 68**

Sei L regulär. Dann erfüllt L die kfr. PE.

Zur Lösung Zum Text**W | F 69**

Sei $h : \{a,b\}^* \rightarrow \{a,b\}$ homomorph mit $h(b) = a$ und $h(a) = aa$. Sei $L := \{baba^2ba^3 \dots ba^{n-1}ba^n b \mid n \geq 1\}$. Dann ist $h(L)$ kfr.

Zur Lösung Zum Text**W | F 70**

Zu jeder unendlichen rekursiv aufzählbaren Sprache L gibt es unendlich viele verschiedene Funktionen f , die L aufzählen?

Zur Lösung Zum Text

10.4 „Wahr oder Falsch“-Lösungen

Lösung zu W | F 1

→ Wahr!

DEA für \bar{L} (endlich) erstellen, indem man alle Wörter fest im Automat verdrahtet, und danach akzeptierende und nicht-akzeptierende Zustände tauschen!

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 2

→ Falsch!

Die Menge der formalen Sprachen ist überabzählbar unendlich (siehe **Reguläre Sprachen**), die Menge der DEAs ist aber abzählbar unendlich, denn es gibt eine bijektive Abbildung auf \mathbb{N} mittels **längen-lexikografischer Ordnung**. Die Menge der formalen Sprachen ist also mächtiger als die Menge der DEAs!

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 3

→ Wahr!

Man kann den DEA, der L akzeptiert, um endlich viele Zustände erweitern, die nicht erreichbar sind, womit die akzeptierte Sprache gleich bleibt!

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 4

→ Wahr!

Die Menge der DEAs (B) ist aber abzählbar unendlich, denn es gibt eine bijektive Abbildung auf \mathbb{N} mittels **längen-lexikografischer Ordnung**. Die Menge der regulären Sprachen (REG) ist gleichmächtig zu der Menge der von DEA akzeptierten Sprachen (A) (**Satz von Kleene**) und die Menge der regulären Sprachen ist abzählbar unendlich (**Reguläre Sprachen**). $|A| = |REG| \wedge |REG| = |B| \implies |A| = |B|$, und gleichmächtig bedeutet, es gibt eine bijektive Abbildung, also gibt es auch eine surjektive Abbildung!

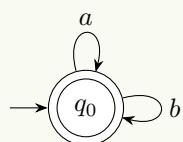
[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 5

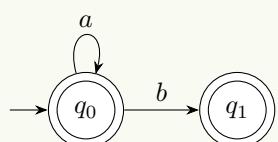
→ Falsch!

Gegenbeispiel: $L = \mathcal{L}(a^* \cup b)$

Wegen a^* ist $\lambda \in L$, daher muss der Startzustand auch akzeptierender Endzustand sein:



↯ (ein Endzustand, aber $L(M) = \mathcal{L}(a^* \cup b^*) \neq L$)



↯ ($L(M) = L$, aber mehr als ein Endzustand)

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 6

→ Falsch!

Gegenbeispiel:

Nicht erreichbare Zustände:



$2 \neq 3$ aber $R(1, 2, 2) = R(1, 2, 3)$

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 7

→ Wahr!

Eine Partitionierung mit n Teilmengen, und 2 Wörter sind äquivalent, wenn sie in der selben Partition liegen.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 8

→ Wahr!

$|w| \geq n$ mit $n = 2p$ (p ist die ursprüngliche Pumping-Konstante) und $y_1 = y_2 = y$!

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 9

→ Falsch!

Gegenbeispiel: $L = \Sigma^* \rightarrow 1 \text{ ÄK}$, welche unendlich ist!

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 10

→ Wahr!

Beispiel: $L = \{a^n b^n \mid n \in \mathbb{N}\} \cup \{a\}$

$[a] = \{a\}$ (→ endlich), aber L ist nicht regulär, denn die regulären Sprachen sind Schnitt-stabil (Abgeschlossenheit bezüglich \cap), aber $L \cap \{a^*.b^*.b\} = \{a^n b^n \mid n \in \mathbb{N}\}$, was nicht regulär ist!

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 11

→ Falsch!

Gegenbeispiel: $L = \mathcal{L}(a^n b^n)$, $L' = \mathcal{L}(a^* b^*) \rightarrow L \setminus L' = \emptyset$, was regulär ist!

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 12

→ Wahr!

Beweis mittels Widerspruch:

Annahme: $L \setminus L'$ ist regulär

$$L = \underbrace{L'}_{\text{regulär}} \cup \underbrace{L \setminus L'}_{\text{regulär}}$$

⇒ L regulär ↯ (Abgeschlossenheit der regulären Sprachen unter Vereinigung)

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 13

→ Falsch!

Gegenbeispiel: $h(a) = b, h(b) = b \rightarrow h^{-1}(\{a\}) = \emptyset$

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 14

→ Falsch!

Gegenbeispiel: $L = \mathcal{L}(a)$

$\exists w \ h(w) = a \Rightarrow h(w.w) = h(w).h(w) = a.a \notin L \not\models$

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 15

→ Wahr!

Wenn h nicht injektiv wäre, dann würde folgen:

$$\begin{aligned} & \exists w_1, w_2 \in \Sigma^*, w_1 \neq w_2 : h(w_1) = h(w_2) = w \\ \Rightarrow & g \circ h(w_1) = g(w) = \tilde{w} = g(w) = g \circ h(w_2) \\ \Rightarrow & w_1 = \tilde{w} = w_2 \not\models \end{aligned}$$

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 16

→ Wahr!

Die Menge der kontextfreien Grammatik ist abzählbar unendlich, denn es gibt eine Bijektion auf \mathbb{N} mittels der Abbildung von G auf ihre Position in der **längenlexikografischen Ordnung**:

Anzahl NTS endlich → Anzahl Regeln endlich → Höchstlänge der rechten Seite

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 17

→ Wahr!

Suche $S \rightarrow A_1 \rightarrow A_2 \rightarrow \dots \rightarrow W \rightarrow \lambda$

(der Algorithmus muss sich merken können, welche NTS bereits besucht wurden)

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 18

→ Wahr!

Top-Down Verfahren siehe obigen Beweis

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 19

→ Falsch!

Wenn $M > 2$ viele Zustände hat, dann hat M' trotzdessen maximal 2, also $M \neq M'$![Zur Frage](#) [Zum Text](#)

Lösung zu W | F 20

→ Wahr!

1 Wort w aus der Sprache L herausnehmen, dieses bildet L_2 (regulär ✓), und für L_1 baut man einen Kellerautomat, der simultan den Kellerautomat für L und einen DEA, der überprüft ob w vorliegt (Produkt-Kellerautomat, kfr. ✓)[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 21

→ Wahr!

PDA^{neu} kann $a^n b^n c^n$ erkennen, indem er a 's oben und b 's unten zählt!

→ Kellerautomat mit 2 Kellern ist am mächtigsten (gleiches Level wie Turingmaschine)

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 22

→ Wahr!

Die beiden Definitions- und Wertebereiche können bijektiv aufeinander zugeordnet werden (beide abzählbar unendlich)

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 23

→ Wahr!

Neue TM \tilde{M} simuliert M und wenn M hält, schreibt \tilde{M} 1. Sollte M unendlich laufen, dann \tilde{M} auch und $f_{\tilde{M}}(w) = \perp = g(w)$ [Zur Frage](#) [Zum Text](#)

Lösung zu W | F 24

→ Falsch!

Es gibt abzählbar unendlich viele TMs (**längen-lexikografische Ordnung**), also auch nur abzählbar unendlich viele berechenbare Funktionen. Jedoch ist \mathbb{N} abzählbar unendlich, die Mächtigkeit der Menge aller Abbildungen (totale Funktion) von \mathbb{N} nach \mathbb{N} ist somit $|\mathbb{N}|^{|\mathbb{N}|} = \aleph_0^{\aleph_0} = 2^{\aleph_0} > \aleph_0$ überabzählbar unendlich. Demnach muss es nicht berechenbare Funktionen geben! $(\aleph_0^{\aleph_0} = 2^{\aleph_0}$ ist Kardinal-Arithmetik)[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 25

→ Wahr!

Eine rationale Zahl kann als Bruch, bestehend aus Nenner $p \in \mathbb{Z}$ und Zähler $q \in \mathbb{Z} \setminus \{0\}$, dargestellt werden. Diese Paare (p, q) können systematisch angeordnet werden (siehe **Cantor's erstes Diagonalargument**), danach wird (p, q) auf seine Position in der Ordnung abgebildet, und man erhält eine berechenbare, surjektive (alle Brüche werden getroffen) Funktion $f(n) = (p, q)$ mit n die Position in der Ordnung. Eine TM kann dann \mathbb{Q} kann einfach rekursiv aufzählen, indem sie nacheinander $f(0), f(1), f(2), \dots$ berechnet.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 26

→ Wahr!

Jede Sprache kann als Bild von einer partiellen Funktion (Abbildung) dargestellt werden:

$$\begin{aligned} w &\mapsto w & \forall w \in \overline{L} \\ w &\mapsto \perp & \forall w \notin \overline{L} \end{aligned}$$

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 27

→ Falsch!

$A \in REC$ (entscheidbar) und $B \notin REC$ (nicht entscheidbar)

→ $A \leq^{mo} B$ aber $B \not\leq^{mo} A$!

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 28

→ nicht immer injektiv sowie surjektiv!

z.B. $H_1 \leq^{mo} H$ mit $f(w) = (w, w)$ ist injektiv, aber nicht surjektiv.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 29

→ L_1 ist auch entscheidbar! (**siehe Lemma zu Reduktion**)

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 30

→ Falsch!

Das Problem bedeutet: $\exists x \in \Sigma^* : x \in (L(M_1) \cap L(M_2))$, was nicht entscheidbar ist.

Widerspruchsbeweis

Annahme: das Problem ist entscheidbar

Daraus würde folgen, dass auch der Sonderfall $L(M_2) = \{a\}$ entscheidbar ist, wobei für diesen Sonderfall die Problemstellung zu „ist a in der von M_1 akzeptierten Sprache“ wird. Dies stellt aber eine Eigenschaft dar, und nach dem **Satz von Rice** kann man nicht entscheiden, ob die von einer TM akzeptierte Sprache eine bestimmte Eigenschaft hat. Dies ist ein Widerspruch, also ist die Annahme falsch und das Problem ist unentscheidbar!

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 31

→ Wahr!

Mit einer festen Anzahl an Zuständen (und festem Σ, Γ) ist die Anzahl der möglichen TMs endlich, somit gibt es endlich viele TMs M_1 und M_2 . Das Problem hat also endlich viele Eingaben und endliche Probleme sind immer entscheidbar!

Eine TM, die das Problem entscheidet, kann einfach alle richtigen Kombination von TMs M_1 und M_2 intern speichern (fest verdrahtet) und dann auf Eingabe M_1, M_2 die passende Antwort geben (unabhängig von einem Algorithmus, der $L(M_1) = L(M_2)$ bestimmen könnte).

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 32

→ Wahr!

Die Aussage stellt folgende Implikation dar: $L_1 \leq_{mo} L_2 \wedge L_2 \leq_{mo} L_3 \implies L_2 \leq_{mo} L_1$. $L_2 \leq_{mo} L_3$ und L_3 entscheidbar würde folgen, dass L_2 auch entscheidbar ist, was einen Widerspruch darstellt. Weil $L_2 \leq_{mo} L_3$ falsch ist, und in der Voraussetzung ein Und (\wedge), ist die Voraussetzung immer falsch, und die Implikation somit immer wahr. Die Aussage gilt also immer!

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 33

→ Falsch!

Gegenbeispiel: f ist nicht injektiv, also z.B. $\forall x \in \Sigma^* f(x) = a$ (f ist konstant) und $L \neq \Sigma^*$.

Dann ist $f(L) = \{a\}$, und L kann nicht mit f auf $f(L) = \{a\}$ reduziert werden:

$$\begin{aligned} L \neq \Sigma^* &\implies \exists x \notin L \\ &\rightarrow x \notin L \Leftrightarrow f(x) = a \in f(L) = \{a\} \not\in \end{aligned}$$

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 34

→ Wahr!

Die Anzahl der, für eine Rechnung verfügbaren, Zellen ist durch die Länge der Eingabe beschränkt, somit ist die Anzahl der möglichen Konfigurationen für die Eingabe endlich. Alle diese endlichen vielen Simulationen können dann durch eine TM simuliert werden, wobei überprüft wird, ob eine davon doppelt auftritt (∞ -Zyklus).

(Die tatsächliche Funktion spielt keine Rolle, also ob Polynom, Exponentialfunktion, etc., die Anzahl der Zellen muss lediglich abhängig von der Eingabe beschränkt sein.)

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 35

→ Wahr!

Ordne die Wörter in L längen-lexikographisch, und dann enthält L_i lediglich das i -te

Wort in dieser Ordnung. Da L_i nur ein Wort enthält, gibt es eine TM M_i , die dieses Wort fest verdrahtet hat und daher nur 1x über die Eingabe geht, also $L_i \in P$.

(Ob L entscheidbar ist, spielt keine Rolle, da die Zerlegung nur existieren, aber nicht berechenbar sein muss. Außerdem ist die Frage $\in P$ nur für unendliche Sprachen interessant.)

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 36

→ Wahr!

L ist regulär, also kann man einen minimalen Zustandsautomaten M mit $L(M) = L$ konstruieren, wobei die Zustände von M den einzelnen Äquivalenzklassen entsprechen. Anschließend kann man eine Rechnung von M auf einem Repräsentant $a \in A$ der ÄK durchführen, und den endgültigen Zustand, wo die Rechnung akzeptiert/verwirft, zu einem akzeptierenden Zustand von M machen (alle bisherigen akzeptierenden Endzustände verlieren ihren Status). Der resultierende DEA entscheidet dann $x \in A$ in Polynomzeit.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 37

→ Wahr!

$$\begin{aligned} x \Leftrightarrow y \vee z &\equiv (x \Rightarrow (y \vee z)) \wedge ((y \vee z) \Rightarrow x) \\ &\equiv (\overline{x} \vee y \vee z) \wedge ((\overline{(y \vee z)} \vee x) \quad (A \Rightarrow B \equiv \overline{A} \vee B) \\ &\equiv (\overline{x} \vee y \vee z) \wedge ((\overline{y} \wedge \overline{z}) \vee x) \\ &\equiv (\overline{x} \vee y \vee z) \wedge (\overline{y} \vee x) \wedge (\overline{z} \vee x) \rightarrow \text{KNF bzw. 3SAT} \end{aligned}$$

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 38

→ Wahr!

Für alle (endlich viele) Eingaben $x \in \{0,1\}^n$ kann man $f(x)$ berechnen, und alles zu einer Tabelle zusammentragen. Anschließend kann man die zugehörige KNF aus den Zeilen konstruieren, die auf 0 ausgewertet wurden. Dazu konstruiert man aus jeder dieser Zeilen eine eigene Klausel, wobei man für $x_i = 0$ x_i und $x_i = 1$ $\overline{x_i}$ in der Klausel einträgt („quasi negierte Version“):

A	B	C	Ergebnis	Klausel
0	0	0	0	$A \vee B \vee C$
0	0	1	0	$A \vee B \vee \neg C$
0	1	0	1	$\neg A \wedge B \wedge \neg C$
0	1	1	1	$\neg A \wedge B \wedge C$
1	0	0	0	$\neg A \vee B \vee C$
1	0	1	1	$A \wedge \neg B \wedge C$
1	1	0	0	$\neg A \vee \neg B \vee C$
1	1	1	1	$A \wedge B \wedge C$

$$\text{DNF: } (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge B \wedge C) \vee (A \wedge \neg B \wedge C) \vee (A \wedge B \wedge C)$$

$$\text{KNF: } (A \vee B \vee C) \wedge (A \vee B \vee \neg C) \wedge (\neg A \vee B \vee C) \wedge (\neg A \vee \neg B \vee C)$$

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 39

→ Falsch!

Gegenbeispiel: $L_1 \neq \Sigma^*$, $L_2 = \Sigma^*$

Es kann keine Reduktion von L_1 auf L_2 geben, da L_2 alle Wörter enthält, somit müsste $f(x) \in L_2$ für alle $x \in \Sigma^*$ gelten, und nach der Reduktion $x \Leftrightarrow f(x)$ auch $x \in L_1$ für alle $x \in \Sigma^*$, was mit $L_1 \neq \Sigma^*$ im Widerspruch steht.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 40

→ Wahr!

Das Problem liegt in NP wegen folgenden Verifikationsalgorithmus:

Die NDTM M „räte“ ein $x \in \mathbb{Q}^n$ und prüft mittels Matrixmultiplikation (effizient berechenbar), ob $Ax = b$ gilt.

Das Raten von x ist möglich, obwohl \mathbb{Q} unendlich viele Werte besitzt, da sowohl A als auch b eine feste, endliche Kodierung besitzen, die auch von deren numerischen Werten abhängt. Die Länge der Bit-Kodierung einer möglichen Lösung x ist somit polynomiell beschränkt in der Eingabegröße. Dadurch gibt es höchsten exponentiell viele mögliche Werte für x , womit der richtige nicht-deterministisch geraten werden kann.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 41

→ Wahr!

Frage zielt darauf ab: Gibt es beim asymptotischen Wachstumsverhalten eine Stufe zwischen polynomiell und exponentiell?

→ Sub-exponentiell, z.B. $f(n) = 2^{\sqrt{n}}$

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 42

→ Falsch!

Gegenbeispiel: $A = C = \Sigma^*$, $B = H$ (Halteproblem)

$A \cap C \setminus B \cap C = \Sigma^* \setminus H = \overline{H}$ ist nicht rekursiv aufzählbar!

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 43

→ Falsch!

Die Menge der Äquivalenzklassen ist eine Partition von L , also die Äquivalenzklassen sind paarweise disjunkte, nicht-leere Teilmengen von L , und vereinigt ergeben sie L . Damit kann jedes $x \in L$ nur in einer ÄK sein, und die maximale Anzahl an Äquivalenzklassen erhält man, indem in jeder Äquivalenzklasse lediglich ein Wort ist.

Da Σ^* abzählbar unendlich ist, kann die Teilmenge L auch nur abzählbar unendlich sein, also gibt es maximal abzählbar unendlich viele Wörter, und die Anzahl an Äquivalenzklassen ist höchstens abzählbar unendlich. Sie kann also nicht überabzählbar sein.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 44

A	B	$A \Rightarrow B$	C	$(A \Rightarrow B) \Rightarrow C$
0	0	1	0	0
0	1	1	0	0
1	0	0	0	1
1	1	1	0	0
<hr/>				
0	0	1	1	1
0	1	1	1	1
1	0	0	1	1
1	1	1	1	1

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 45

→ Falsch!

Σ^i ist endlich, also auch L_i , und damit gilt natürlich $L_i \in P$ und $L_i \in NP$. Da L_i beliebig gewählt werden kann, kann durch die Vereinigung aller L_i auch eine beliebige Sprache erzeugt werden. Die Frage bedeutet also vielmehr „alle Sprachen liegen in NP“, was natürlich mit $H \notin REC$ nicht der Fall ist!

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 46

→ Falsch!

Gegenbeispiel: $A = H$, $B = \Sigma^*$

$A \cup B = \Sigma^*$ ist regulär, aber $H \not\leq_{poly}^{mo} SAT$!

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 47

→ Wahr!

Alle Elemente $x \in \Sigma$ sind reguläre Grundausdrücke. Die Vereinigung all dieser x und anschließende Hülle-Bildung (Kleene-Stern) bildet einen regulären Ausdruck für L .

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 48

→ Falsch!

Reelle Zahlen sind überabzählbar unendlich. Reguläre Sprachen sind abzählbar unendlich.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 49

→ Wahr!

Sei α ein regulärer Ausdruck mit $L(\alpha) = L(M) = L$.

α ist also der reguläre Ausdruck, welcher Wörter der Sprache L generiert. Wir bilden einen neuen Ausdruck α' , mit $\alpha' = \alpha.a$

Wir haben somit den regulären Ausdruck für die Sprache $L.\{a\}$ konstruiert.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 50

→ Falsch!

Injektiv: $\forall x_1, x_2 \in A : (f(x_1) = f(x_2) \Rightarrow x_1 = x_2)$

Die Funktion ist nicht injektiv, da es für eine reguläre Sprache L , unendlich viele DEAs gibt, welche L akzeptieren (nicht erreichbare Zustände hinzufügen).

Surjektiv: $\forall y \in B \exists x \in A : f(x) = y$

Die Funktion ist nicht surjektiv, da nur reguläre Sprachen von DEAs akzeptiert werden können. Da es überabzählbar unendlich viele Sprachen, aber nur abzählbar unendlich viele reguläre Sprachen gibt, muss es Sprachen geben die nicht regulär sind, und folglich nicht von DEAs akzeptiert werden können. Also können nicht alle Sprachen im Bildbereich von einem DEA des Definitionsbereiches abgebildet werden.

Bijektiv: injektiv und surjektiv

Die Funktion ist weder injektiv noch surjektiv und somit nicht bijektiv.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 51

→ Falsch!

Die Aussage ist schon falsch formuliert. f ist keine Abbildung.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 52

→ Wahr!

Beweis:

1. Jede endliche Sprache ist regulär, und jede Sprache erfüllt die reg. PE.
2. Wähle n größer als das längste Wort, somit können wir zum Pumpen kein Wort

aus L wählen, und die Sprache erfüllt die reg. PE.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 53

→ Falsch!

Es müsste sein: $x \approx_L y \Rightarrow \forall z \in \Sigma^* xz \in L \Leftrightarrow yz \in L$

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 54

→ Wahr!

Endliche Sprachen sind regulär. Dafür kann man einen DEA konstruieren, welcher endlich viele Zustände hat. Der minimale Automat, hat dann genauso viele Zustände wie L Äquivalenzklassen bzgl. \approx_L

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 55

→ Falsch!

Sei $L = \{a, b\}$. Dann sind die Äquivalenzklassen bzgl. \approx_L $K_1 = [a] = \{a, b\}$, $K_2 = [\lambda] = \{\lambda\}$ und $K_3 = \Sigma^* \setminus (\{a, b\} \cup \{\lambda\})$. Dabei sind K_1 und K_2 endlich und K_3 unendlich.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 56

→ Falsch!

Seien $\Sigma = \{a, b\}$, $L = \{a^n \mid n \in \mathbb{N}\}$ und $f(a^n) = a^n b^n$.

Dann ist L regulär und $f(L) = \{a^n b^n \mid n \in \mathbb{N}\}$ offensichtlich nicht regulär.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 57

→ Wahr!

Wenn L regulär ist, gibt es einen endlichen Automaten M , der L akzeptiert. Minimiert man diesen, so sind die Äquivalenzklassen \approx_L und \sim_M die Gleichen.

Nun lassen wir M auf x und y rechnen, und prüfen ob er auf beiden Wörtern im gleichen Zustand hält. Ist es der gleiche Zustand, gilt $x \approx_L y$.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 58

→ Wahr!

Bilde $L' = L \cap \{a^i b^j c^k \mid i, j, k \geq 1\} = \{a^k b^k c^n \mid k, n \geq 1\}$.

Bilde dann Homomorphismus $h : \Sigma^* \rightarrow \Sigma^*$ mit $h(a) = a$, $h(b) = b$, $h(c) = \lambda$. Dann ist $h(L) = \{a^k b^k \mid k \geq 1\}$ nicht regulär.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 59

→ Falsch!

Wähle für alle $a \in \Sigma$ die Abbildung: $h(a) = \lambda$. Dann gilt $h(L) = \{\lambda\}$. Das ist offen-

sichtlich regulär.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 60

→ Wahr!

$h(0) = 0, h(1) = 1, h(\lambda) = 0$

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 61

→ Wahr!

Kontextfreie Sprachen sind unter Homomorphismus und Vereinigung abgeschlossen. Aufgrund der Homomorphie sind $h(L_1)$ und $h(L_2)$ kontextfrei, und aufgrund der Abgeschlossenheit unter Vereinigung auch $h(L_1) \cup h(L_2)$.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 62

→ Falsch!

Sei $\Sigma = \{a, b\}$ und $h(a) = b, h(b) = b$. Durch die Doppelbelegung kommt man nie wieder auf das Urbild.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 63

→ Wahr!

Ist L regulär, so kann ich einen DEA M konstruieren, sodass $L(M) = L$ gilt. Dieser besitzt eine Überführungsfunktion. M kann ich ebenso als Kellerautomat bilden, bei welchem nie etwas auf den Stack gelegt wird. So hätte dieser Automat auch eine Überführungsfunktion.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 64

→ Wahr!

Beide sind abzählbar unendlich, daher muss es eine bijektive Abbildung geben, und diese ist ja auch injektiv.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 65

→ Wahr!

Sei $L = \Sigma^*$ und $\tilde{L} = \overline{\{a^n b^n c^n \mid n \geq 1\}} \subseteq L$. Dann ist $L \setminus \tilde{L} = \{a^n b^n c^n \mid n \geq 1\}$. Die resultierende Sprache ist nicht kontextfrei und somit gibt es auch keine kfr. Grammatik dafür.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 66

→ Wahr!

Bilde einfach die kontextfreie Grammatik die L generiert. Und erstelle einen Zwischen-

schritt mit

$$\begin{aligned} S &\rightarrow S_1 S_2 \\ S_1 S_2 &\rightarrow S' \end{aligned}$$

Dann nutze S' als neues Startsymbol, welches die „normalen“ Regeln der kontextfreien Grammatik einleitet.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 67

→ Wahr!

Man kann einen Produktautomaten aus dem PDA M_1 für L_1 ($L(M_1) = L_1$) und dem DEA M_2 für L_2 ($L(M_2) = L_2$) bauen. Die erkannte Sprache ist wieder kontextfrei.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 68

→ Wahr!

Die regulären Sprachen sind genau die rechtslinearen Grammatiken, welche wiederum kontextfrei sind. Und jede kontextfreie Sprache erfüllt die kontextfreie Pumping-Eigenschaft (kfr. Pumping-Lemma).

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 69

→ Wahr!

$h(L) = \{a^{k^2} \mid k \in \mathbb{N} \setminus \{0, 1\}\}$ ist nicht kontextfrei, da sie nicht die kfr. PE hat (siehe [hier](#)).

Man hat $1+2+3+\dots+n = \frac{n(n+1)}{2}$ viele a in $w \in L$, und auf jede a -Gruppe folgt ein $b+1$ am Anfang macht $n+1$ viele b 's. Mit $h(a) = aa$ folgt also $n(n+1)$ viele a in $h(w)$ und mit $h(b) = a$ noch $n+1$ weitere. Macht insgesamt $n(n+1)+n+1 = n^2+2n+1 = (n+1)^2$ viele a 's.

[Zur Frage](#) [Zum Text](#)

Lösung zu W | F 70

→ Wahr!

L ist rekursiv aufzählbar, also muss es eine berechenbare Funktion $f : \mathbb{N} \rightarrow \Sigma^*$ geben mit:

$$\text{Bild}(f) = L \quad , f \text{ ist surjektiv}$$

(die Aufzähler-TM durchläuft alle $i \in \mathbb{N}$ parallel diagonalisiert, und gibt jeweils das Resultat von $f(i)$ aus)

Aus dieser Funktion f können dann unendlich viele Funktionen g_j ($j \in \mathbb{N}$) gebildet werden:

$$g_j(i) = \begin{cases} f(j) & , i = 1 \\ f(1) & , i = j \\ f(i) & , \text{sonst} \end{cases}$$

Da L unendlich ist, und f surjektiv sein muss, bildet f auf abzählbar unendlich viele verschiedene Wörter ab, also hat f auch abzählbar unendlich viele verschiedene $a, b \in \mathbb{N}$ mit $f(a) \neq f(b)$. Somit lassen sich auch abzählbar unendlich viele verschiedene Funktionen g_j konstruieren, die L aufzählen.

[Zur Frage](#) [Zum Text](#)