

Formale Sprachen und Automaten

Jan Hladik



1. Einführung

1.1 Organisation

1.2 Motivation

1.3 Grundlagen formaler Sprachen

2. Reguläre Sprachen und endliche Automaten

3. Chomsky-Grammatiken und kontextfreie Sprachen

4. Turing-Maschinen

5. Entscheidbarkeit

6. Berechenbarkeit

7. Komplexität

Inhalt

1. Einführung

1.1 Organisation

1.2 Motivation

1.3 Grundlagen formaler Sprachen

2. Reguläre Sprachen und endliche Automaten

3. Chomsky-Grammatiken und kontextfreie Sprachen

4. Turing-Maschinen

5. Entscheidbarkeit

6. Berechenbarkeit

7. Komplexität

1993–2001 Diplom Informatik, RWTH Aachen

- Diplomarbeit: Tableau-Algorithmus
- Nebenfach: Philosophie

2001–2007 Promotion, TU Dresden

- Dissertation: Tableaus vs. Automaten
- Leitung von Übungsgruppen

2008–2014 Industrieerfahrung, SAP Research Dresden

- Öffentlich geförderte Forschungsprojekte
- Betreuung von Studenten und Doktoranden

seit 2014 Professor, DHBW Stuttgart

- Logik
- Formale Sprachen und Automaten
- Semantic Web

Forschung Semantische Technologien

- Beschreibungslogik
- Logische Schlussfolgerungsverfahren
- Semantic Web



- Präsentation, Übungsaufgaben
 - <http://wwwlehre.dhbw-stuttgart.de/~hladik/FSA>
- Klausur
 - Dauer: 90 min
 - Hilfsmittel: Formelsammlung 5 Blatt DIN A4 (beidseitig beschrieben)
- Literatur
 - Dirk W. Hoffmann:
[Theoretische Informatik](#)
 - Ulrich Hettstädt:
[Einführung in die theoretische Informatik](#)
 - Uwe Schöning:
[Theoretische Informatik - kurz gefasst](#)
 - John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman:
[Introduction to Automata Theory, Languages, and Computation](#)

1. Einführung

1.1 Organisation

1.2 Motivation

1.3 Grundlagen formaler Sprachen

2. Reguläre Sprachen und endliche Automaten

2.1 Reguläre Ausdrücke

2.2 Endliche Automaten

2.2.1 Deterministische endliche Automaten

2.2.2 Nicht-deterministische endliche Automaten

2.2.3 Endliche Automaten und reguläre Ausdrücke

2.2.4 Minimierung

2.3 Nicht-reguläre Sprachen und das Pumping-Lemma

2.4 Eigenschaften regulärer Sprachen

3. Chomsky-Grammatiken und kontextfreie Sprachen

3.1 Chomsky-Grammatiken

3.2 Die Chomsky-Hierarchie

3.3 Rechtslineare Grammatiken

3.4 Kontextfreie Grammatiken

3.5 Entscheidungsprobleme

3.6 Kellerautomaten

3.7 Eigenschaften kontextfreier Sprachen

4. Turing-Maschinen

4.1 Motivation

4.2 Aufbau und Funktionsweise

4.3 Mehrband-Turingmaschinen

4.4 Unbeschränkte Grammatiken

4.5 Linear beschränkte Automaten

5. Entscheidbarkeit

5.1 Unentscheidbarkeit des speziellen Wortproblems

5.2 Reduktionsbeweise

5.3 Das PKP und weitere unentscheidbare Probleme

5.4 Semi-Entscheidbarkeit

5.5 Die universelle Turing-Maschine

5.6 Abschlusseigenschaften

6. Berechenbarkeit

6.1 Turing-Berechenbarkeit

6.2 WHILE-Programme

6.3 Die Church-Turing-These

7. Komplexität

7.1 Komplexitätsklassen

7.2 NP-Vollständigkeit

Inhalt

1. Einführung

1.1 Organisation

1.2 Motivation

1.3 Grundlagen formaler Sprachen

2. Reguläre Sprachen und endliche Automaten

3. Chomsky-Grammatiken und kontextfreie Sprachen

4. Turing-Maschinen

5. Entscheidbarkeit

6. Berechenbarkeit

7. Komplexität

Alphabet: endliche Menge Σ von Symbolen (Zeichen)

- $\{a, b, c\}$

Wort: endliche Folge w von Zeichen (String)

- $ab \neq ba$

Sprache: (möglicherweise unendliche) Menge L von Wörtern

- $\{ab, ba\} = \{ba, ab\}$

Formal: L eindeutig definiert

- im Gegensatz zu Grenzfällen in natürlichen Sprachen

Beispiel 1.1

- Namen in einem Telefonbuch
- Telefonnummern in einem Telefonbuch
- zulässige C-Bezeichner
- zulässige C-Programme
- zulässige HTML5-Dokumente
- leere Menge
- ASCII-Strings
- Unicode-Strings
- gültige mathematische Sätze

Vier Sprachklassen unterschiedlicher Komplexität und Ausdrucksstärke

regulär begrenzte Ausdrucksstärke, aber einfach zu handhaben

kontext-frei ausdrucksstärker, aber noch handhabbar

kontext-sensitiv noch ausdrucksstärker, algorithmisch schwierig zu handhaben

rekursiv aufzählbar allgemeinste Klasse (dieser Vorlesung); unentscheidbar

- Abfolge verschiedener Muster
- Alternativen
- Wiederholung von Mustern
- Serielle Strukturen
- **Zählen** nur bis zu einer fixen Grenze

Beispiel 1.2 (reguläre Sprachen)

- „Wörter, die mit einem Buchstaben anfangen, gefolgt von bis zu 7 Buchstaben oder Ziffern“
- zulässige C-Bezeichner
- Telefonnummern

Jan says that we
let
the children
help
Hans
paint
the house

- Verschachtelte Strukturen ([nested](#) dependencies)
- „Zu jedem <token> gibt es ein </token>“
- Schließende Token in umgekehrter Reihenfolge der öffnenden
- Unbegrenztes [Zählen](#)

Beispiel 1.3 (kontextfreie Sprachen)

- korrekt geklammerte arithmetische Ausdrücke
- HTML / XML
- (meiste Eigenschaften von) zulässigen C-Programmen
- meiste natürliche Sprachen (Englisch, Deutsch)

Jan sagt, dass wir
die Kinder
dem Hans
das Haus
anstreichen
helfen
ließen

- Kreuz-Abhangigkeiten (cross-serial dependencies)
- „Jede Variable muss vor ihrer ersten Benutzung deklariert werden“
(Beliebige Reihenfolge; beliebiger Code zwischen Deklaration und Verwendung)
- Schlieende Token in gleicher Reihenfolge wie offnende

Beispiel 1.4 (kontextsensitive Sprachen)

- (verbleibende Eigenschaften von) zulssigen C-Programmen
- verbleibende natrliche Sprachen (Schweizerdeutsch)

Jan sat das mer
d'chind
em Hans
es huus
lnd
helfe
aastriche

Jan sagt, dass wir
die Kinder
dem Hans
das Haus
lieen
helfen
anstreichen

- praktisch brauchbare, „vernünftige“ Sprachen
- von einem Algorithmus sukzessive erzeugbar
 - Wort₀, Wort₁, Wort₂, ...
- auch: semi-entscheidbare Sprachen
 - Frage: Gehört Wort w zu Sprache L ?
 - Wenn ja, findet Computer dies in endlicher Zeit heraus
 - Wenn nein, kann die Berechnung unendlich lange dauern
- nicht alle Sprachen sind rekursiv aufzählbar!

Beispiel 1.5 (rekursiv aufzählbare Sprachen)

- alle gültigen Sätze der Prädikatenlogik erster Stufe
- alle Programme, die auf einer gegebenen Eingabe anhalten

Automaten

- abstraktes formales Maschinenmodell
- charakterisiert durch Zustände, Symbole, Übergänge und externen Speicher

Ein Automat

- akzeptiert Wörter
- erkennt eine Sprache

Satz 1.6 (Korrespondenz von Sprachklassen und Maschinenmodellen)

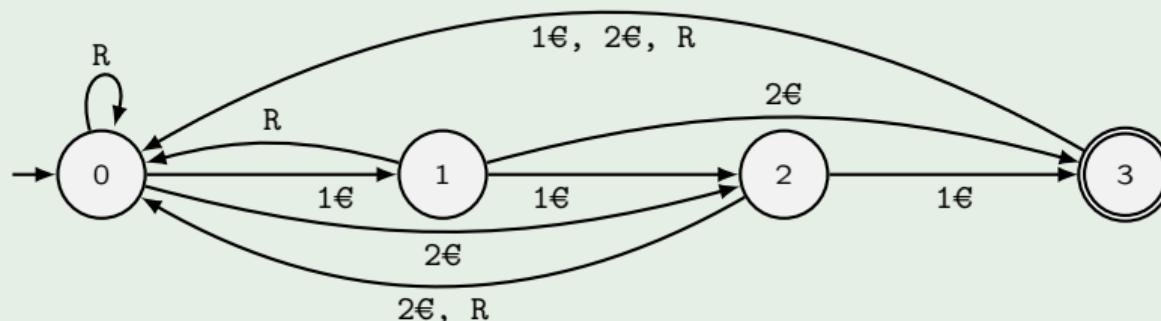
Für jede genannte Sprachklasse K gibt es ein Maschinenmodell M_K , so dass

- für jede Sprache $L \in K$ ein Automat $\mathcal{A}(L)$ existiert, der zum Modell M_K gehört und genau die Sprache L erkennt;
- für jeden Automat \mathcal{A} , der zum Modell M_K gehört, die von \mathcal{A} erkannte Sprache $\mathcal{L}(\mathcal{A}mc)$ zur Klasse K gehört.

regulär	\rightsquigarrow	endlicher Automat
kontextfrei	\rightsquigarrow	Kellerautomat
kontextsensitiv	\rightsquigarrow	Linear beschränkte Turingmaschine
rekursiv aufzählbar	\rightsquigarrow	(unbeschränkte) Turingmaschine

Beispiel 1.7

- Eintritt kostet 3 EUR
- Einwurf für 1€- und 2€-Münzen
- Geldrückgabe bei Überzahlung oder durch Rückgabeknopf R

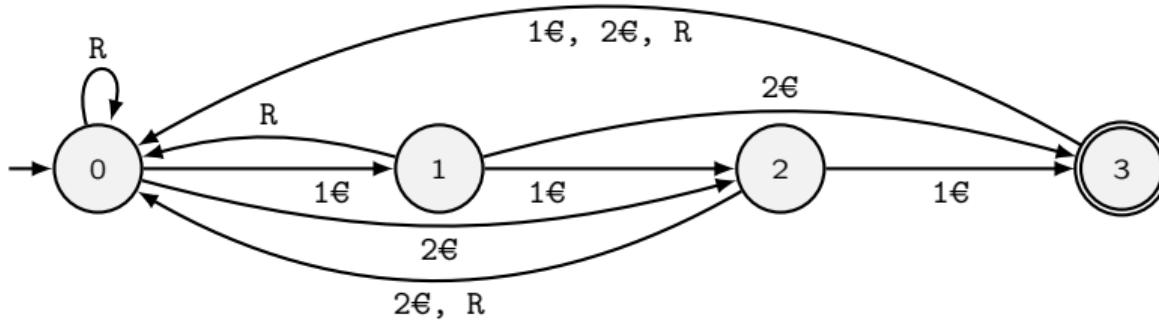


Zustände Mögliche Situationen; **Gedächtnis** des Automaten

Startzustand Zustand zu Beginn: 0€

Endzustand Eingabe wird akzeptiert: 3€

Übergänge von Ausgangszustand zu Zielzustand mit **Symbol** (1€, 2€, R)



■ Endliche Zustandsmenge

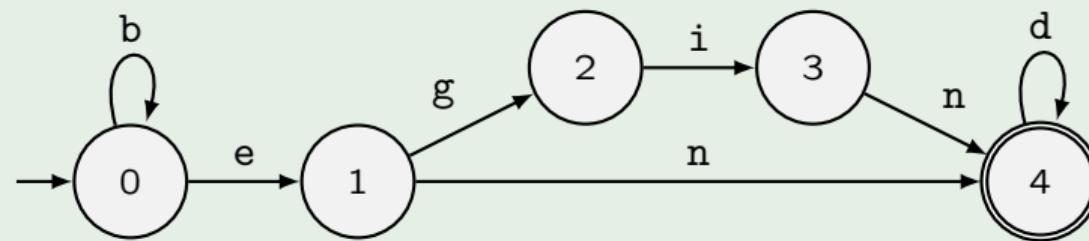
- Menge möglicher Situationen muss **endlich** und **im Voraus bekannt** sein
- Nicht geeignet z.B. zur Modellierung eines Kontos

■ Keine Ausgabe

- Nur „akzeptiert“ oder „nicht akzeptiert“
- Geldrückgabe oder Wechselgeld können nicht modelliert werden

Übung 1.8

Welche der Wörter begin, end, bind, bend, beg akzeptiert der folgende Automat?



Erzeugen Wörter über einem Alphabet

Terminalsymbole Dürfen im erzeugten Wort auftauchen (Alphabet)

Nichtterminalsymbole Dürfen **nicht** im erzeugten Wort auftauchen (temporär)

Produktionsregeln $l \rightarrow r$ bedeutet: l kann irgendwo im Wort durch r ersetzt werden

Beispiel 1.9 (Grammatik für arithmetische Ausdrücke über $\{x, y\}$)

$$\begin{aligned}\Sigma &= \{x, y, +, \cdot, (,)\} \\ N &= \{E\} \\ P &= \{E \rightarrow x, E \rightarrow y, \\ &\quad E \rightarrow (E) \\ &\quad E \rightarrow E + E \\ &\quad E \rightarrow E \cdot E\}\end{aligned}$$

Eine Ableitung: $E \Rightarrow E \cdot E \Rightarrow (E) \cdot E \Rightarrow (E + E) \cdot E \Rightarrow (x + E) \cdot E \Rightarrow (x + y) \cdot E \Rightarrow (x + y) \cdot y$

Übung 1.10

Gegeben seien

- das Nichtterminalsymbol S ,
- die Terminalsymbole b, d, e, g, i, n ,
- die Produktionsregeln $S \rightarrow \text{begin}, \text{in} \rightarrow n, eg \rightarrow \text{egg}, \text{beg} \rightarrow e, ggg \rightarrow b, \text{in} \rightarrow \text{ind}$.

Können Sie ausgehend vom Symbol S die Wörter `bend` und `end` erzeugen?

- Wenn ja, wieviele Ableitungsschritte benötigen Sie?
- Wenn nein, warum nicht?

- Wie kann man für eine Sprache L , die durch einen Automaten \mathcal{A}_L gegeben ist, eine Grammatik G_L finden (und umgekehrt)?
- Was ist der einfachste Automat für L ?
 - „einfach“ im Sinne von „aus der schwächsten Klasse“
 - „einfach“ im Sinne von „mit den wenigsten Zuständen“
- Wie können wir formale Sprachen in der Praxis einsetzen?
 - Erkennen zulässiger oder reservierter Wörter
 - Test auf zulässige Struktur
 - Suche nach Mustern

Abschlusseigenschaften Wenn L_1 und L_2 in einer Klasse sind, gilt das auch für

- die Vereinigung von L_1 und L_2 ,
- den Durchschnitt von L_1 und L_2 ,
- die Konkatenation von L_1 und L_2 ,
- das Komplement von L_1 ?

Entscheidungsprobleme Für ein Wort w und Sprachen L_1 und L_2 :

- Gilt $w \in L_1$?
- Ist L_1 endlich?
- Ist L_1 leer?
- Gilt $L_1 = L_2$?

Beispiel 1.11

Scanner: Suche nach Mustern (reguläre Sprachen)

- Matching regulärer Ausdrücke in großen Texten
- Schlüsselwörter in Quellcode

Parser: Analyse der Dateistruktur (kontextfreie Sprachen)

- Compiler
- HTML und Web-Browser
- Spracherkennung und KI

1. Einführung

1.1 Organisation

1.2 Motivation

1.3 Grundlagen formaler Sprachen

2. Reguläre Sprachen und endliche Automaten

3. Chomsky-Grammatiken und kontextfreie Sprachen

4. Turing-Maschinen

5. Entscheidbarkeit

6. Berechenbarkeit

7. Komplexität

Erinnerung: \mathbb{N} bezeichnet die Menge der natürlichen Zahlen $\{0, 1, 2, \dots\}$.

Definition 1.12 (Alphabet)

Ein **Alphabet** Σ ist eine endliche, nicht-leere Menge von Symbolen (Zeichen, Buchstaben).

$$\Sigma = \{c_1, \dots, c_n\}, \quad n \in \mathbb{N}^{>0}$$

Im Folgenden verwenden wir häufig die Alphabete $\Sigma_{ab} = \{a, b\}$ und $\Sigma_{abc} = \{a, b, c\}$.

Beispiel 1.13

- $\Sigma_{bin} = \{0, 1\}$ kann natürliche Zahlen im Binärsystem darstellen.
- Die deutsche Sprache basiert auf $\Sigma_{de} = \{a, \dots, z, ä, ö, ü, ß, A, \dots, Ü\}$.
- $\Sigma_{ASCII} = \{0, \dots, 127\}$ bezeichnet die Menge der ASCII-Zeichen und codiert Buchstaben, Ziffern, Sonderzeichen und Kontrollsymbole.

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	!	"	#	\$	%	&	'	()	*	+	,	-	.	/	
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	-
6	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Definition 1.14 (Wort)

- Ein **Wort** über dem Alphabet Σ ist eine endliche Folge von Symbolen aus Σ :

$$w = c_1 \dots c_n \quad \text{mit} \quad c_1, \dots, c_n \in \Sigma, n \in \mathbb{N}.$$

- Das **leere Wort** ε enthält keine Symbole.
- Σ^* bezeichnet die Menge aller Wörter über dem Alphabet Σ .

Beispiel 1.15

- Wörter aus Σ_{bin}^* : $w_1 = 01100$ und $w_2 = 11001$
- Ein Wort aus Σ_{de}^* : $w = \text{Beispiel}$

In Programmiersprachen werden Wörter **Strings** genannt.

Definition 1.16 (Länge, einzelne Zeichen eines Wortes)

Sei w ein Wort.

$|w|$ ist die Länge von w , d.h. die Anzahl der Zeichen in w .

$|w|_c$ ist die Anzahl der Vorkommen des Zeichens c in w .

$w[i]$ ist das Zeichen an der Stelle i in w (mit $i \in \{1, 2, \dots, |w|\}$).

$w[i, j]$ ist das Teilwort von w von der Stelle i bis zur Stelle j
(mit $\{i, j\} \subseteq \{1, 2, \dots, |w|\}$).

Beispiel 1.17

- $|\text{Beispiel}| = 8$ und $|\varepsilon| = 0$
- $|\text{Beispiel}|_e = 2$ und $|\text{Beispiel}|_k = 0$
- $\text{Beispiel}[3] = i$
- $\text{Beispiel}[4, 8] = \text{spiel}$

Definition 1.18 (Konkatenation von Wörtern)

Für Wörter w_1 und w_2 ist die Konkatenation $w_1 \cdot w_2$ definiert als w_1 gefolgt von w_2 .

$w_1 \cdot w_2$ wird oft einfach w_1w_2 geschrieben.

Beispiel 1.19

Mit $w_1 = \text{ab}$ und $w_2 = \text{ba}$ gilt:

$$w_1w_2 = \text{abba} \quad \text{and} \quad w_2w_1 = \text{baab}$$

Definition 1.20 (Potenz)

Für $n \in \mathbb{N}$ bezeichnet die n -te Potenz w^n eines Wortes w die n -fache Konkatenation von w :

$$\begin{aligned} w^0 &= \varepsilon \\ w^{n+1} &= w^n \cdot w \end{aligned}$$

Beispiel 1.21

Mit $w = ab$ gilt:

$$\begin{aligned} w^0 &= \varepsilon \\ w^1 &= ab \\ w^3 &= ababab \end{aligned}$$

Übung 1.22

Gegeben seien die folgenden Wörter über Σ_{abc} :

- $u = abc,$
- $v = aa,$
- $w = cb.$

Was bezeichnen die folgenden Ausdrücke?

- 1 $u^2 \cdot w$
- 2 $v \cdot \varepsilon \cdot w \cdot u^0$
- 3 $|u^3|_a$
- 4 $v \cdot a^2 \cdot (v[2])$
- 5 $(v \cdot a^2 \cdot v)[2]$
- 6 $(u \cdot w)[3, 4]$
- 7 $|w^0|$
- 8 $|w^0 \cdot w|$

Definition 1.23 (Formale Sprache)

Eine formale Sprache L über einem Alphabet Σ ist eine Menge von Wörtern aus Σ^* : $L \subseteq \Sigma^*$.

Beispiel 1.24

- $L_1 = \{w \cdot \text{heit} \mid w \in \Sigma_{\text{de}}^*\}$
ist die Menge aller Wörter über Σ_{de} , die auf heit enden;
- $L_2 = \{w \in \Sigma_{\text{ASCII}}^* \mid |w|_a \geq 1\}$
ist die Menge aller Wörter über dem ASCII-Alphabet, die mindestens ein a enthalten;
- $L_3 = \{w \in \Sigma_{\text{bin}}^* \mid |w| \geq 2 \wedge w[|w| - 1] = 1\}$
ist die Menge aller Binärwörter, an deren vorletzter Stelle 1 steht.

Definition 1.25 (Produkt formaler Sprachen)

Das **Produkt** zweier formaler Sprachen L_1, L_2 , über einem Alphabet Σ ist

$$L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1, w_2 \in L_2\}.$$

Beispiel 1.26

Gegeben seien die Sprachen

$$L_1 = \{\text{ab}, \text{bc}\} \quad \text{and} \quad L_2 = \{\text{ac}, \text{cb}\}.$$

über dem Alphabet Σ_{de} . Ihr Produkt ist

$$L_1 \cdot L_2 = \{\text{abac}, \text{abcb}, \text{bcac}, \text{bccb}\}.$$

Definition 1.27 (Potenz einer Sprache)

Für ein Alphabet Σ , eine formale Sprache $L \subseteq \Sigma^*$ und eine natürliche Zahl $n \in \mathbb{N}$ ist die ***n*-te Potenz** von L wie folgt rekursiv definiert:

$$\begin{aligned}L^0 &= \{\varepsilon\} \\L^{n+1} &= L^n \cdot L\end{aligned}$$

Beispiel 1.28

Sei $L = \{ab, ba\}$ über dem Alphabet Σ_{de} . Dann gilt:

$$\begin{aligned}L^0 &= \{\varepsilon\} \\L^1 &= \{\varepsilon\} \cdot \{ab, ba\} = \{ab, ba\} \\L^2 &= \{ab, ba\} \cdot \{ab, ba\} = \{abab, abba, baab, baba\}\end{aligned}$$

Definition 1.29 (Kleene-Stern)

Für ein Alphabet Σ und eine formale Sprache $L \subseteq \Sigma^*$ ist der Operator **Kleene-Stern** wie folgt definiert:

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



Stephen Cole Kleene
(1909–1994)

Beispiel 1.30

Sei $L = \{ab, ba\}$ über dem Alphabet Σ_{de} . Dann gilt:

$$L^* = \{\varepsilon, ab, ba, abab, abba, baab, baba, ababab, \dots\}.$$

Beispiel 1.31 (Binärwörter)

- Die Sprache $L_{\mathbb{N}} = \{1w \mid w \in \Sigma_{\text{bin}}^*\} \cup \{0\}$
 - enthält genau die Wörter über Σ_{bin} , die mit 1 anfangen, sowie das Wort 0;
 - enthält (außer dem Wort 0) keine Wörter mit führenden Nullen ($10 \in L_{\mathbb{N}}$, aber $010 \notin L_{\mathbb{N}}$);
 - enthält ein eindeutiges Binärwort für jede natürliche Zahl.
- Die Funktion $n : L_{\mathbb{N}} \rightarrow \mathbb{N}$ bildet jedes Wort aus $L_{\mathbb{N}}$ auf ihren Zahlenwert ab:
 - $n(0) = 0$,
 - $n(1) = 1$,
 - $n(w0) = 2 \cdot n(w)$ für $|w| > 0$,
 - $n(w1) = 2 \cdot n(w) + 1$ für $|w| > 0$.

Achtung

- $n(w)$ ist eine natürliche Zahl,
- w ist ein String, der diese Zahl codiert!

Übung 1.32

Gegeben seien

- das Alphabet Σ_{bin} ,
- die Funktion n aus Beispiel 1.31,
- die Sprache $L = \{1\}$.

Geben Sie formale Beschreibungen für:

- 1 die Sprache $A = L^* \setminus \{\varepsilon\}$
- 2 die Menge $B = \{n(w) \mid w \in A\}$
- 3 die Sprache $C = \{w \in \Sigma_{\text{bin}}^* \mid n(w) - 1 \in B\}$
- 4 die Sprache $D = \{w \in \Sigma_{\text{bin}}^* \mid n(w) + 1 \in B\}$

- Endliches Alphabet
- Endlich lange Wörter
- Endliche oder unendliche Sprachen
- Operationen auf Wörtern und Sprachen
 - $L_1 \cdot L_2$ Konkatenation
 - L^n Potenz
 - L^* Kleene-Stern
- Vier Sprach-Klassen unterschiedlicher Ausdrucksstärke
 - Zugehörige Maschinenmodelle
 - Zugehörige Grammatiken

1. Einführung

2. Reguläre Sprachen und endliche Automaten

2.1 Reguläre Ausdrücke

2.2 Endliche Automaten

2.3 Nicht-reguläre Sprachen und das Pumping-Lemma

2.4 Eigenschaften regulärer Sprachen

3. Chomsky-Grammatiken und kontextfreie Sprachen

4. Turing-Maschinen

5. Entscheidbarkeit

6. Berechenbarkeit

7. Komplexität

Inhalt

1. Einführung

2. Reguläre Sprachen und endliche Automaten

2.1 Reguläre Ausdrücke

2.2 Endliche Automaten

2.3 Nicht-reguläre Sprachen und das Pumping-Lemma

2.4 Eigenschaften regulärer Sprachen

3. Chomsky-Grammatiken und kontextfreie Sprachen

4. Turing-Maschinen

5. Entscheidbarkeit

6. Berechenbarkeit

7. Komplexität

Beispiel 2.1 (Anwendungen regulärer Ausdrücke)

- Beschreibung von Token für Compiler
- Beschreibung von Suchbegriffen in einer Datenbank
- Suche nach Mustern in Gen-Daten
- Extraktion von URLs oder Mail-Adressen aus Webseiten

Beispiel 2.2 (Reguläre Ausdrücke für Sprachen aus Übung 1.32)

$L_{\mathbb{N}}$ $0 + 1 \cdot (0 + 1)^*$ - das Wort „0“ und Wörter, die mit „1“ anfangen

A $1 \cdot 1^*$ - Wörter, die aus mindestens einer „1“ bestehen

C $1 \cdot 0 \cdot 0^*$ - Wörter, die mit „1“ anfangen, gefolgt vom mindestens einer „0“

D $1^* \cdot 0$ - Wörter, die mit einer (möglicherweise leeren) Folge von „1“ anfangen und mit „0“ enden

Ein regulärer Ausdruck über Σ ...

- ... ist ein Wort über dem Alphabet $\Sigma \cup \{\emptyset, \varepsilon, +, \cdot, ^*, (,)\}$
 - Wir setzen implizit voraus, dass $\{\emptyset, \varepsilon, +, \cdot, ^*, (,)\} \cap \Sigma = \{\}$ gilt
 - \emptyset bezeichnet einen regulären Ausdruck (Syntax)
 - $\{\}$ bezeichnet die leere Menge (Semantik)
- ... beschreibt eine formale Sprache über Σ

Achtung

- Ein regulärer Ausdruck über Σ beschreibt eine formale Sprache.
- Die Menge aller regulären Ausdrücke über Σ ist eine formale Sprache.

Definition 2.3 (Reguläre Ausdrücke)

Auf den folgenden Seiten werden definiert:

- Die Menge R_Σ der regulären Ausdrücke über einem Alphabet Σ .
- Die Funktion \mathcal{L} , die jedem regulären Ausdruck r eine formale Sprache $\mathcal{L}(r) \subseteq \Sigma^*$ zuordnet.

Definition 2.3 (Reguläre Ausdrücke)

Sei Σ ein Alphabet.

- 1 Der reguläre Ausdruck \emptyset bezeichnet die [leere Sprache](#).
 $\emptyset \in R_\Sigma$ und $\mathcal{L}(\emptyset) = \{\}$
- 2 Der reguläre Ausdruck ε bezeichnet die Sprache, die nur aus dem [leeren Wort](#) besteht.
 $\varepsilon \in R_\Sigma$ und $\mathcal{L}(\varepsilon) = \{\varepsilon\}$
- 3 Jedes Symbol $c \in \Sigma$ ist ein regulärer Ausdruck.
 $c \in R_\Sigma$ und $\mathcal{L}(c) = \{c\}$

Definition 2.3 (Reguläre Ausdrücke (Fortsetzung))

Sei $\{r, s\} \subseteq R_\Sigma$.

- 4 Der Operator $+$ bezeichnet die **Vereinigung** der Sprachen von r und s .
 $(r + s) \in R_\Sigma$ und $\mathcal{L}((r + s)) = \mathcal{L}(r) \cup \mathcal{L}(s)$
- 5 Der Operator \cdot bezeichnet das **Produkt** der Sprachen von r und s .
 $(r \cdot s) \in R_\Sigma$ und $\mathcal{L}((r \cdot s)) = \mathcal{L}(r) \cdot \mathcal{L}(s)$
- 6 Der **Kleene-Stern** von r bezeichnet den Kleene-Stern der Sprache von r .
 $(r^*) \in R_\Sigma$ und $\mathcal{L}((r^*)) = (\mathcal{L}(r))^*$

Definition 2.4 (Äquivalenz)

Zwei reguläre Ausdrücke r_1 und r_2 sind äquivalent ($r_1 \equiv r_2$), wenn sie dieselbe Sprache beschreiben ($\mathcal{L}(r_1) = \mathcal{L}(r_2)$).

Notation

- Der Vorrang der Operatoren ist:
* \gg . \gg +
- Folgende Symbole können zur Vereinfachung weggelassen werden:
 - die äußersten Klammern um einen Ausdruck
 - Klammern, die den Vorrang nicht ändern
 - der Produkt-Operator .
- Manchmal wird „|“ als Vereinigungs-Operator verwendet.

Beispiel 2.5 (Vereinfachte Notation regulärer Ausdrücke)

$$\begin{aligned}(a + (b \cdot (c^*))) &\equiv a + b \cdot c^* \\ ((a \cdot c) + ((b \cdot c)^*)) &\equiv ac + (bc)^*\end{aligned}$$

Vorsicht!

$$((a \cdot c) + ((b \cdot c)^*)) \neq ac + (bc)^*$$

Beispiel 2.6

- Sprache aller Wörter über Σ_{abc} , die aus genau zwei Symbolen bestehen:

$$\begin{aligned} r_1 &= (a + b + c)(a + b + c) \\ \mathcal{L}(r_1) &= \{w \in \Sigma_{abc}^* \mid |w| = 2\} \end{aligned}$$

- Sprache aller Wörter über Σ_{abc} , die aus mindestens einem Symbol bestehen:

$$\begin{aligned} r_2 &= (a + b + c)(a + b + c)^* \\ \mathcal{L}(r_2) &= \{w \in \Sigma_{abc}^* \mid |w| \geq 1\} \end{aligned}$$

- Sprache aller Wörter über Σ_{abc} , die eine gerade Länge (einschließlich 0) haben:

$$\begin{aligned} r_3 &= ((a + b + c)(a + b + c))^* \\ \mathcal{L}(r_3) &= \{w \in \Sigma_{abc}^* \mid \exists n \in \mathbb{N} : |w| = 2 \cdot n\} \end{aligned}$$

Übung 2.7

- 1 Geben Sie einen regulären Ausdruck r_1 für alle Wörter $w \in \Sigma_{abc}^*$ an, die genau ein a oder genau ein b enthalten.
- 2 Beschreiben Sie $\mathcal{L}(r_1)$ formal als Menge.
- 3 Geben Sie einen regulären Ausdruck r_3 für alle Wörter $w \in \Sigma_{abc}^*$ an, die mindestens ein a und mindestens ein b enthalten.
- 4 Geben Sie einen regulären Ausdruck r_4 für alle Wörter $w \in \Sigma_{ab}^*$ an, deren drittletztes Symbol ein b ist.
- 5 Beschreiben Sie anschaulich die Sprache $\mathcal{L}(r_5)$.

$$r_5 = a(ba)^* + (ba)^* + b(ab)^* + (ab)^*$$

- 6 Beschreiben Sie anschaulich die Sprache $\mathcal{L}(r_6)$.

$$r_6 = (b + \varepsilon)(aa^*b)^*a^*$$

- 7 Geben Sie einen regulären Ausdruck r_7 für alle Wörter $w \in \Sigma_{ab}^*$ an, die nicht das Teilwort bba enthalten.

Satz 2.8

Es gelten die folgenden Äquivalenzen:

1	$r_1 + r_2 \equiv r_2 + r_1$	Kommutativität von +
2	$(r_1 + r_2) + r_3 \equiv r_1 + (r_2 + r_3)$	Assoziativität von +
3	$(r_1 r_2) r_3 \equiv r_1 (r_2 r_3)$	Assoziativität von ·
4	$\emptyset r \equiv r \emptyset \equiv \emptyset$	Absorbierendes Element für ·
5	$\varepsilon r \equiv r \varepsilon \equiv r$	Neutrales Element für ·
6	$\emptyset + r \equiv r$	Neutrales Element für +
7	$(r_1 + r_2) r_3 \equiv r_1 r_3 + r_2 r_3$	Distributivität links
8	$r_1(r_2 + r_3) \equiv r_1 r_2 + r_1 r_3$	Distributivität rechts
9	$r + r \equiv r$	Idempotenz von +
10	$(r^*)^* \equiv r^*$	Idempotenz von *
11	$\emptyset^* \equiv \varepsilon$	
12	$\varepsilon^* \equiv \varepsilon$	
13	$\varepsilon + r^* r \equiv r^*$	
14	$(\varepsilon + r)^* \equiv r^*$	
15	$r^* r \equiv rr^*$	

Beweis der Kommutativität (Äquivalenz 1, $r_1 + r_2 \equiv r_2 + r_1$).

$$\mathcal{L}(r_1 + r_2) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2) = \mathcal{L}(r_2) \cup \mathcal{L}(r_1) = \mathcal{L}(r_2 + r_1)$$

□

Beweis der Absorption (Äquivalenz 4, $\emptyset r \equiv \emptyset$).

$$\begin{array}{ll} \mathcal{L}(\emptyset r) & \text{Definition Konkatenation} \\ & \equiv \\ & \text{Definition Leere Sprache} \\ & \equiv \\ & \text{Definition Produkt} \\ & \equiv \\ & \text{Definition Konkatenation} \\ & \equiv \\ & \text{Definition Leere Sprache} \\ & \equiv \end{array} \begin{array}{l} \mathcal{L}(\emptyset) \cdot \mathcal{L}(r) \\ \{\} \cdot \mathcal{L}(r) \\ \{w_1 w_2 \mid w_1 \in \{\}, w_2 \in \mathcal{L}(r)\} \\ \{\} \\ \mathcal{L}(\emptyset) \end{array}$$

□

Übung 2.9

- 1 Zeigen Sie die folgende Äquivalenz nur mit algebraischen Äquivalenzen aus Satz 2.8:

$$r^* \equiv \varepsilon + r^*$$

- 2 Vereinfachen Sie den regulären Ausdruck s so weit wie möglich mit Hilfe algebraischer Äquivalenzen:

$$s = a(\varepsilon + a + b)^* + (\varepsilon + b)(b + a)^* + \varepsilon$$

Übersicht: Zulässige Operatoren

	Vereinigung	Konkatenation	Potenz	Kleene-Stern
Wörter	$\textcolor{red}{X}$	$w_1 \cdot w_2$	w^n	$\textcolor{red}{X}$
Sprachen	$L_1 \cup L_2$	$L_1 \cdot L_2$	L^n	L^*
Reguläre Ausdrücke	$r_1 + r_2$	$r_1 \cdot r_2$	$\textcolor{red}{X}$	r^*

Beispiel 2.10 (Zulässige und unzulässige Konstruktionen)

1 a^2b^3

Wort aabbbb

2 $\{a^n b^m \mid n \in \mathbb{N}^{\geq 1} \wedge m \in \mathbb{N}^{\leq 5}\}$

Sprache aller Wörter, die zuerst mindestens ein a enthalten und anschließend höchstens 5 b

3 $\{a, b\}^3 \cup \{a\}^*$

Sprache aller Wörter, die aus genau 3 a oder b bestehen, oder nur aus a

4 $(a + b)(a + b)(a + b) + a^*$

Regulärer Ausdruck für 3

1 $\{a^n b + b^n a \mid n \in \mathbb{N}\}$

Falsch: + in Menge

Korrekt: $\{a^n b \mid n \in \mathbb{N}\} \cup \{b^n a \mid n \in \mathbb{N}\}$

2 $\{a^n \cdot b^* \mid n \in \mathbb{N}^{\geq 2}\}$

Falsch: * für Wort

Korrekt: $\{a^n \cdot b^m \mid n \in \mathbb{N}^{\geq 2} \wedge m \in \mathbb{N}\}$

3 $a^* + ab^2$

Falsch: Potenz in regulärem Ausdruck

Korrekt: $a^* + abb$

- Basis: $\emptyset, \varepsilon, c \in \Sigma$
- Operatoren: $+, \cdot, ^*$
- Algebraische Operationen und Äquivalenz
 - keine Entscheidungsverfahren für $r \equiv s$ oder $w \in \mathcal{L}(r)$
- Anwendung: Suche nach Mustern
 - Schlüsselwörter
 - Token im Compilerbau (Scanner)

1. Einführung

2. Reguläre Sprachen und endliche Automaten

2.1 Reguläre Ausdrücke

2.2 Endliche Automaten

2.3 Nicht-reguläre Sprachen und das Pumping-Lemma

2.4 Eigenschaften regulärer Sprachen

3. Chomsky-Grammatiken und kontextfreie Sprachen

4. Turing-Maschinen

5. Entscheidbarkeit

6. Berechenbarkeit

7. Komplexität

- sehr einfaches Berechnungsmodell
- erkennt reguläre Sprachen
- äquivalent zu regulären Ausdrücken
 - Transformation eines EA in einen RA
 - Transformation eines RA in einen EA
- Varianten:
 - deterministisch (DEA)
 - nicht-deterministisch (NEA)
- in Anwendungen leicht zu implementieren

Inhalt

1. Einführung

2. Reguläre Sprachen und endliche Automaten

2.1 Reguläre Ausdrücke

2.2 Endliche Automaten

2.2.1 Deterministische endliche Automaten

2.2.2 Nicht-deterministische endliche Automaten

2.2.3 Endliche Automaten und reguläre Ausdrücke

2.2.4 Minimierung

2.3 Nicht-reguläre Sprachen und das Pumping-Lemma

2.4 Eigenschaften regulärer Sprachen

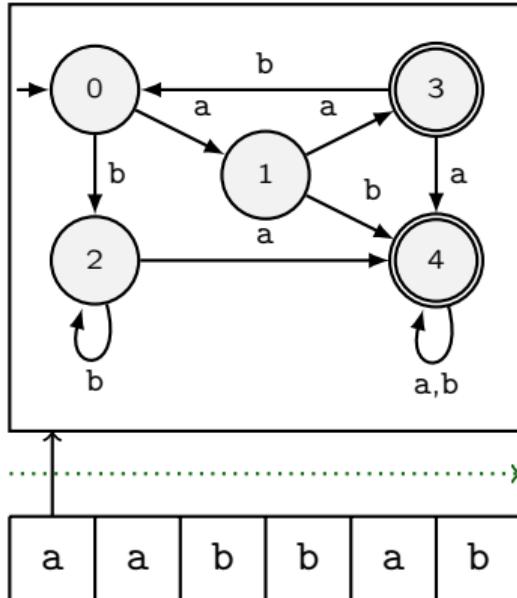
3. Chomsky-Grammatiken und kontextfreie Sprachen

4. Turing-Maschinen

5. Entscheidbarkeit

6. Berechenbarkeit

7. Komplexität



Deterministischer Endlicher Automat (DEA)

- hat endlich viele **Zustände**
- beginnt im **Startzustand**
- liest **Eingabe** von Links nach rechts
 - ändert aktuellen Zustand abhängig vom gelesenen Zeichen
 - entsprechend der **Übergangsfunktion**
 - kein Zurückspulen!
 - kein Schreiben!
- akzeptiert Eingabe, wenn
 - nach dem Lesen der gesamten Eingabe
 - ein **Endzustand** erreicht ist

Definition 2.11 (Deterministischer Endlicher Automat)

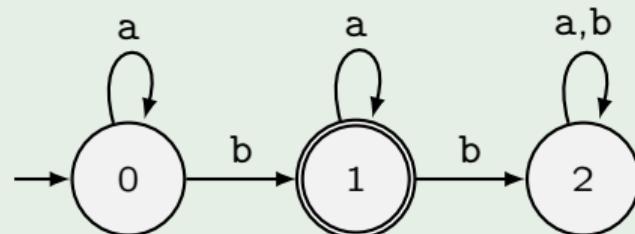
Ein deterministischer endlicher Automat (DEA) ist ein 5-Tupel $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ mit den folgenden Komponenten:

- Q ist eine endliche Menge von Zuständen.
- Σ ist ein endliches Alphabet.
- $\delta : Q \times \Sigma \rightarrow Q$ ist die Übergangsfunktion.
- $q_0 \in Q$ ist der Startzustand.
- $F \subseteq Q$ ist die Menge der Endzustände.

Anmerkungen:

- Die Funktion δ ist **total**: Es gibt von jedem Zustand mit jedem Symbol einen Übergang.
- Ein Automat mit **partieller** Übergangsfunktion ist kein DEA, ...
- ... aber man kann ihn „reparieren“, indem man einen weiteren Zustand hinzufügt.

Beispiel 2.12 (Automat \mathcal{A}_b für $\mathcal{L}(a^*ba^*)$)



$\mathcal{A}_b = (Q, \Sigma, \delta, q_0, F)$ mit

- $Q = \{0, 1, 2\}$
- $\Sigma = \Sigma_{ab}$
- $\delta(0, a) = 0; \delta(0, b) = 1, \delta(1, a) = 1; \delta(1, b) = \delta(2, a) = \delta(2, b) = 2$
- $q_0 = 0$
- $F = \{1\}$

Zustand 2: „Mülleimerzustand“ (junk state), d.h. kein Wort wird mehr akzeptiert

Definition 2.13 (Konfiguration, Lauf, erkannte Sprache eines DEA)

Sei $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ ein DEA.

Eine **Konfiguration** von \mathcal{A} ist ein Paar (q, w) mit $q \in Q$ und $w \in \Sigma^*$.

Ein **Lauf** von \mathcal{A} auf einem Wort $w = c_1 \cdot c_2 \cdots c_n$ ist eine Folge von Konfigurationen:

$$((q_0, c_1 \cdot c_2 \cdots c_n), (q_1, c_2 \cdots c_n), \dots, (q_n, \varepsilon))$$

so dass für alle $1 \leq i \leq n$ gilt: $\delta(q_{i-1}, c_i) = q_i$.

Ein Lauf heißt **akzeptierend**, wenn $q_n \in F$ gilt; sonst heißt er **verwerfend**.

Die von \mathcal{A} erkannte Sprache ist:

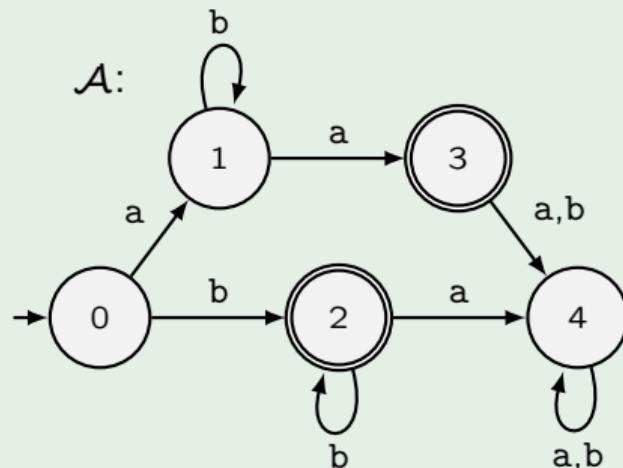
$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \text{der Lauf von } \mathcal{A} \text{ auf } w \text{ ist akzeptierend}\}$$

Wörter werden akzeptiert.

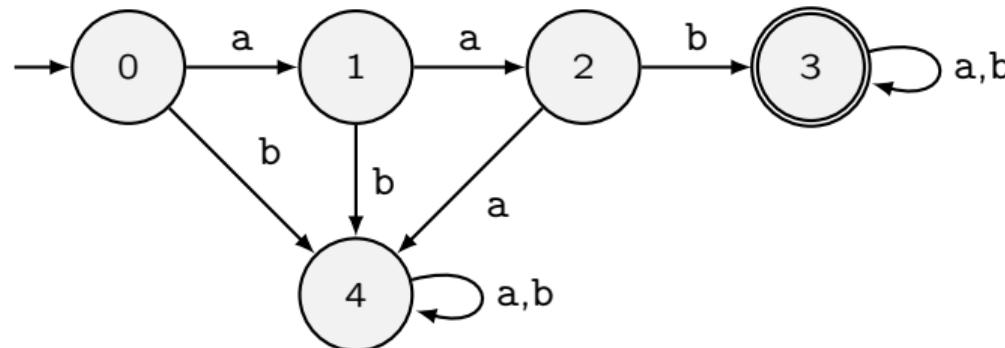
Sprachen werden erkannt.

Übung 2.14

- 1 Gegeben sei der DEA \mathcal{A} .
 - a Geben Sie eine formale Definition von \mathcal{A} (nicht $\mathcal{L}(\mathcal{A})$).
 - b Geben Sie einen regulären Ausdruck für $\mathcal{L}(\mathcal{A})$ an.
- 2 Gegeben seien die Sprachen L_1, L_2, L_3 über Σ_{ab} mit den folgenden Bedingungen:
 - 1 L_1 enthält genau die Wörter, die mit ab beginnen.
 - 2 L_2 enthält genau die Wörter, die irgendwo ab enthalten.
 - 3 L_3 enthält genau die Wörter, die mit ab enden.Geben Sie für L_1, L_2, L_3 jeweils an:
 - a einen regulären Ausdruck,
 - b einen DEA in grafischer Darstellung,
 - c eine formale Definition der Sprache.



Tabellendarstellung eines DEA



$$\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$$

- $Q = \{0, 1, 2, 3, 4\}$
- $\Sigma = \{a, b\}$
- $\delta(0, a) = 1, \delta(0, b) = 4, \delta(1, a) = 2, \delta(1, b) = 4,$
 $\delta(2, a) = 4, \delta(2, b) = 3, \delta(3, a) = 3, \delta(3, b) = 3,$
 $\delta(4, a) = 4, \delta(4, b) = 4$
- $q_0 = 0$
- $F = \{3\}$

\mathcal{A}	a	b
→ 0	1	4
1	2	4
2	4	3
*	3	3
4	4	4

Übung 2.15

Gegeben seien die Sprachen

- 1 $L_1 = \{ubw \mid u \in \Sigma_{abc}^*, w \in \Sigma_{abc}\}$
- 2 $L_2 = \{ubw \mid u \in \Sigma_{abc}, w \in \Sigma_{abc}^*\}$

Geben Sie DEAs für L_1 und L_2 an und notieren Sie jeweils die Zeit, die sie benötigen.

- Komponenten:
 - endliche Zustandsmenge, Startzustand, Endzustände
 - endliches Alphabet
 - Übergangs-Funktion
 - genau ein Übergang aus jedem Zustand mit jedem Buchstaben
 - ggf. Mülleimerzustand
- kein Schreiben
- kein Zurückspulen
- Darstellung: Tupel, Graph, Tabelle
- Lauf: Konfigurationsfolge
- Akzeptanzbedingung: Nach Lesen der gesamten Eingabe ist Endzustand erreicht

Inhalt

1. Einführung

2. Reguläre Sprachen und endliche Automaten

2.1 Reguläre Ausdrücke

2.2 Endliche Automaten

2.2.1 Deterministische endliche Automaten

2.2.2 Nicht-deterministische endliche Automaten

2.2.3 Endliche Automaten und reguläre Ausdrücke

2.2.4 Minimierung

2.3 Nicht-reguläre Sprachen und das Pumping-Lemma

2.4 Eigenschaften regulärer Sprachen

3. Chomsky-Grammatiken und kontextfreie Sprachen

4. Turing-Maschinen

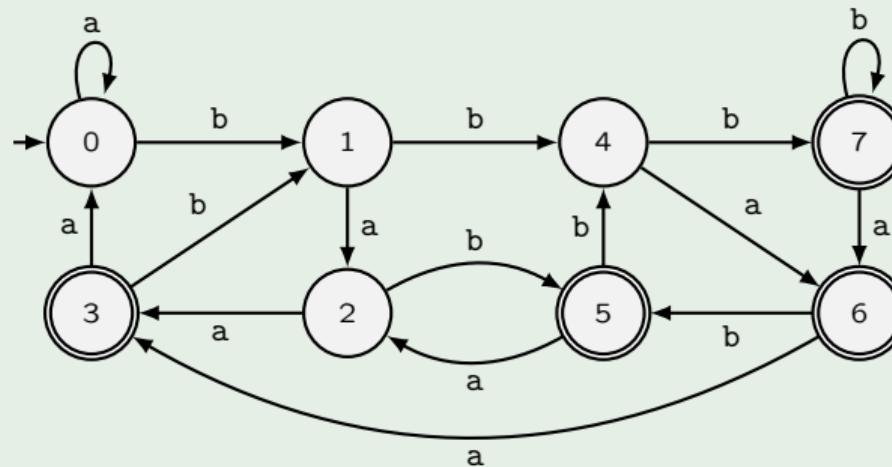
5. Entscheidbarkeit

6. Berechenbarkeit

7. Komplexität

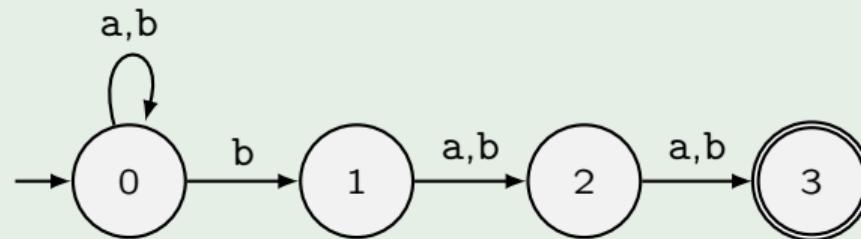
- Übergangs-Funktion δ
- für jede Konfiguration genau eine Folgekonfiguration
- führt schon für einfache Sprachen zu komplexen Automaten

Beispiel 2.16 (DEA \mathcal{A}_d für $(a + b)^*b(a + b)(a + b)$)

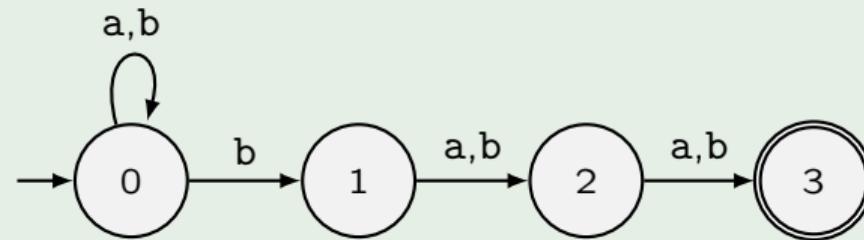


- Ziel: Weniger notwendige Zustände \rightsquigarrow übersichtlichere Automaten
- Ansatz: Aus einer Konfiguration kann es
 - einen,
 - mehrere,
 - oder gar keinen Übergang geben.
- Intuitiv: Automat **rät** Übergang
 - und rät immer „richtig“ (akzeptiert, wenn möglich)
- statt Übergangs-Funktion δ : Übergangs-**Relation** Δ
- **Nicht-deterministischer Endlicher Automat (NEA)**

Beispiel 2.17 (NEA \mathcal{A}_n für $(a + b)^*b(a + b)(a + b)$)



Beispiel 2.18 (Übergänge von \mathcal{A}_n)

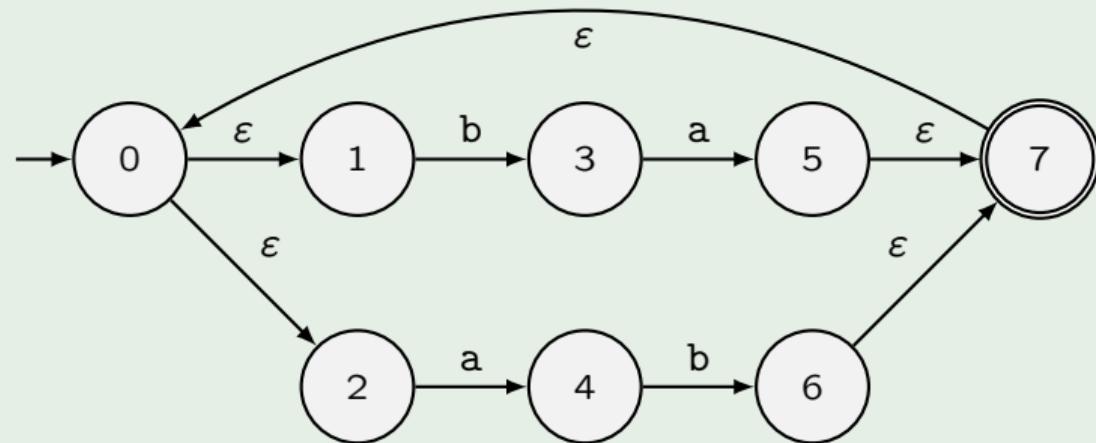


- Im Zustand 0 mit Eingabe **b** rät \mathcal{A}_n den Folgezustand.
- Es gibt 3 mögliche Läufe von \mathcal{A}_n auf dem Wort **abab**:
 - 1 ((0, abab), (0, bab), (0, ab), (0, b), (0, ϵ)) (verwerfend)
 - 2 ((0, abab), (0, bab), (0, ab), (0, b), (1, ϵ)) (verwerfend)
 - 3 ((0, abab), (0, bab), (1, ab), (2, b), (3, ϵ)) (akzeptierend)
- \mathcal{A}_n akzeptiert **abab**, weil ein akzeptierender Lauf existiert

ε -Übergänge

- NEA kann Zustand ändern, ohne Eingabesymbol zu lesen: Übergang $(0, \varepsilon, 1) \in \Delta$.
- Δ ist eine Relation über $Q \times (\Sigma \cup \{\varepsilon\}) \times Q$

Beispiel 2.19 (NEA mit ε -Übergängen)



Definition 2.20 (NEA, Lauf)

Ein NEA ist ein 5-Tupel $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ mit den folgenden Komponenten:

- Q, Σ, q_0 und F sind wie für einen DEA definiert.
- Δ ist eine Relation über $Q \times (\Sigma \cup \{\varepsilon\}) \times Q$.

Ein Lauf von \mathcal{A} auf einem Wort $w_0 = c_1 \cdot c_2 \cdots c_n \in \Sigma^*$ ist eine Folge von Konfigurationen

$$l = ((q_0, w_0), (q_1, w_1), \dots, (q_m, \varepsilon))$$

so dass für alle Paare von Konfigurationen $(q_i, w_i), (q_{i+1}, w_{i+1})$ in l gilt:

- Es gibt einen Übergang $(q_i, c, q_{i+1}) \in \Delta$ mit
- $w_i = c \cdot w_{i+1}$.

Ein Lauf l ist akzeptierend, wenn q_m ein Endzustand ist.

Wegen der Möglichkeit von ε -Übergängen

- ist die Definition eines Laufs komplexer als für DEAs,
- kann ein Lauf auf w mehr als $|w| + 1$ Konfigurationen haben.

Definition 2.21 (von einem NEA erkannte Sprache)

Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NEA.

Die von \mathcal{A} erkannte Sprache ist:

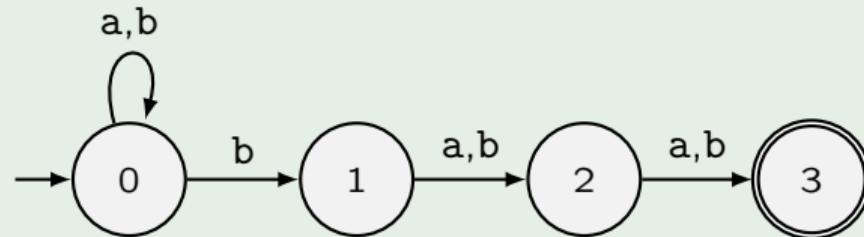
$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \text{es gibt einen akzeptierenden Lauf von } \mathcal{A} \text{ auf } w\}$$

Wichtig

- Nur die Existenz eines akzeptierenden Laufs wird gefordert.
- Existenz verwerfender Läufe ist irrelevant.

Beispiel: NEA formal

Beispiel 2.22 (Formale Definition von \mathcal{A}_n)



$\mathcal{A}_n = (Q, \Sigma, \Delta, q_0, F)$ mit

$$Q = \{0, 1, 2, 3\}$$

$$\Sigma = \Sigma_{ab}$$

$$\Delta = \{(0, a, 0), (0, b, 0), (0, b, 1), (1, a, 2), (1, b, 2), (2, a, 3), (2, b, 3)\}$$

$$q_0 = 0$$

$$F = \{3\}$$

\mathcal{A}_n	a	b	ϵ
→ 0	{0}	{0, 1}	{}
1	{2}	{2}	{}
2	{3}	{3}	{}
*	{}	{}	{}
3			

Übung 2.23

Erzeugen Sie einen NEA \mathcal{A} über dem Alphabet Σ_{ab} , der genau die Wörter akzeptiert, die das Teilwort **aba** enthalten.

Geben Sie hierzu an:

- 1 einen regulären Ausdruck für $\mathcal{L}(\mathcal{A})$,
- 2 eine grafische Darstellung von \mathcal{A} ,
- 3 eine formale Definition von \mathcal{A} (Tabelle oder Tupel).

Satz 2.24 (Rabin, Scott)

NEA und DEA beschreiben dieselbe Sprachklasse.

- 1 Für jeden DEA \mathcal{A} gibt es einen NEA $\text{ndet}(\mathcal{A})$ mit $\mathcal{L}(\text{ndet}(\mathcal{A})) = \mathcal{L}(\mathcal{A})$.
- 2 Für jeden NEA \mathcal{A} gibt es einen DEA $\text{det}(\mathcal{A})$ mit $\mathcal{L}(\text{det}(\mathcal{A})) = \mathcal{L}(\mathcal{A})$.

1 ist trivial:

- Jede Funktion kann als Relation geschrieben werden.
- \rightsquigarrow Jeder DEA kann als NEA geschrieben werden.



Michael Rabin
(*1931)



Dana Scott
(*1932)

Konstruktiver Beweis

- durch Angabe eines Algorithmus,
- der aus einem gegebenen NEA \mathcal{A}
- einen DEA $\text{det}(\mathcal{A})$ für $\mathcal{L}(\mathcal{A})$ erzeugt.

Ansatz

- Zustand von $\text{det}(\mathcal{A})$ repräsentiert Menge M von Zuständen von \mathcal{A}
- In $\text{det}(\mathcal{A})$ ist $\delta(M, c)$ die Menge aller Zustände, zu denen es in \mathcal{A}
 - von **einem** der Zustände in M
 - einen Übergang mit dem Zeichen c gibt.
- Endzustände von $\text{det}(\mathcal{A})$: Mengen, die **mindestens einen** Endzustand aus \mathcal{A} enthalten

Drei Hilfsfunktionen

ec ε -Abschluss: mit ε -Übergängen erreichbare Zustände

δ' mögliche Folgezustände eines **einzelnen** Zustands von \mathcal{A}

$\hat{\delta}$ mögliche Folgezustände einer Zustands-Menge

Definition 2.25 (Potenzmenge)

Die Potenzmenge 2^S einer Menge S ist die Menge aller Teilmengen von S .

Beispiel 2.26

Sei $S = \{0, 1, 2\}$. Dann gilt: $2^S = \{\emptyset, \{0\}, \{1\}, \{2\}, \{0, 1\}, \{0, 2\}, \{1, 2\}, \{0, 1, 2\}\}$

Definition 2.27 (ε -Abschluss)

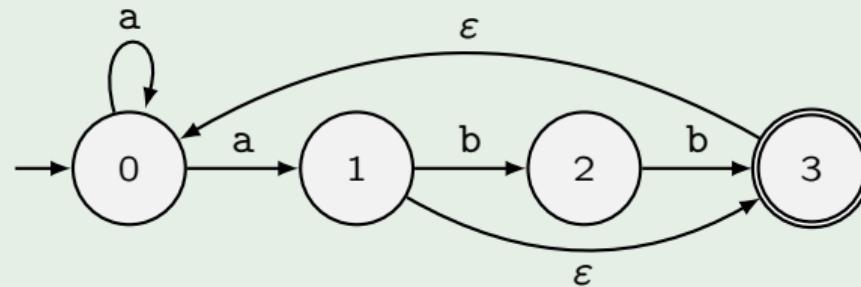
Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NEA.

Die Funktion $ec : Q \rightarrow 2^Q$ bildet jeden Zustand q auf die kleinste Menge ab, für die gilt:

- $q \in ec(q)$
- gilt $p \in ec(q)$ und $(p, \varepsilon, r) \in \Delta$, dann auch $r \in ec(q)$.

Beispiel: ε -Abschluss

Beispiel 2.28



- $ec(0) = \{0\}$
- $ec(1) = \{1, 3, 0\}$
- $ec(2) = \{2\}$
- $ec(3) = \{3, 0\}$

Schritt 2: Folgezustände eines einzelnen Zustands

Die Funktion δ'

- bildet ein Paar (q, c) ab
- auf die Menge aller Zustände, die der NEA von q aus erreichen kann,
 - indem er zuerst c liest
 - und danach beliebig viele ε -Übergänge durchführt.

Definition 2.29 (Funktion δ')

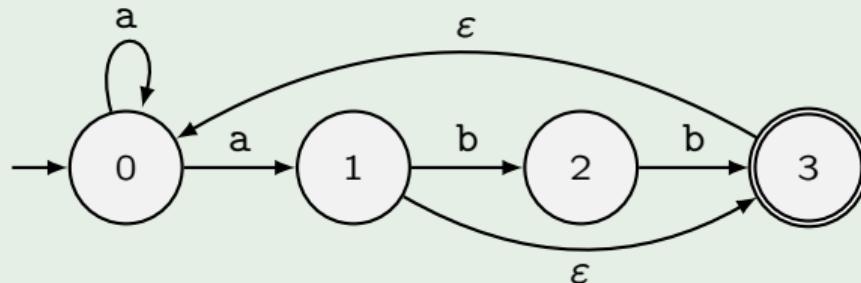
Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NEA.

Die Funktion $\delta' : Q \times \Sigma \rightarrow 2^Q$ ist wie folgt definiert:

$$\delta'(q, c) = \bigcup_{r \in Q : (q, c, r) \in \Delta} ec(r)$$

Beispiel: Funktion δ'

Beispiel 2.30



$$\delta'(q, c) = \bigcup_{r \in Q : (q, c, r) \in \Delta} ec(r)$$

- $\delta'(0, a) = \{0, 1, 3\}$
- $\delta'(0, b) = \{\}$
- $\delta'(1, a) = \{\}$
- $\delta'(1, b) = \{2\}$
- $\delta'(2, a) = \{\}$
- $\delta'(2, b) = \{0, 3\}$
- $\delta'(3, a) = \{\}$
- $\delta'(3, b) = \{\}$

Schritt 3: Folgezustände einer Zustandsmenge

Die Funktion $\hat{\delta}$

- bildet ein Paar (M, c) mit einer Zustandsmenge M ab
- auf die Menge aller Zustände, die der NEA von **einem** Zustand in M aus erreichen kann,
 - indem er **zuerst** c liest
 - und **danach** beliebig viele ε -Übergänge durchführt.

Definition 2.31 (Funktion $\hat{\delta}$)

Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NEA.

Die Funktion $\hat{\delta} : 2^Q \times \Sigma \rightarrow 2^Q$ ist wie folgt definiert:

$$\hat{\delta}(M, c) = \bigcup_{q \in M} \delta'(q, c).$$

Definition 2.32 (Potenzautomat)

Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NEA.

Der zu \mathcal{A} äquivalente DEA ist

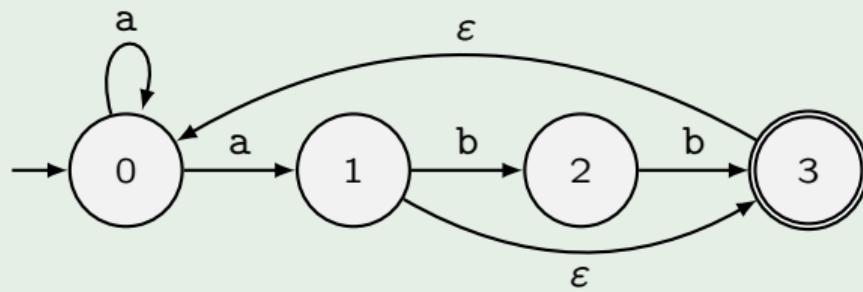
$$\text{det}(\mathcal{A}) = (2^Q, \Sigma, \hat{\delta}, \text{ec}(q_0), \hat{F})$$

mit $\hat{F} = \{M \in 2^Q \mid M \cap F \neq \emptyset\}$.

- \hat{F} : Alle Teilmengen von Q , die **mindestens** einen Endzustand von \mathcal{A} enthalten.
- Effiziente Berechnung von \hat{Q} :
 - erzeuge nicht gesamte Potenzmenge
 - sondern nur die von $\text{ec}(q_0)$ aus **erreichbaren** Zustände

Beispiel: Funktion $\hat{\delta}$

Beispiel 2.33



$$\hat{\delta}(M, c) = \bigcup_{q \in M} \delta'(q, c)$$

- $\delta'(0, a) = \{0, 1, 3\}$
- $\delta'(1, b) = \{2\}$
- $\delta'(2, b) = \{0, 3\}$
- $\text{ec}(0) = \{0\} =: S_0$

- $\hat{\delta}(S_0, a) = \{0, 1, 3\} =: S_1$
- $\hat{\delta}(S_0, b) = \{\} =: S_j$
- $\hat{\delta}(S_1, a) = \{0, 1, 3\} = S_1$
- $\hat{\delta}(S_1, b) = \{2\} =: S_2$
- $\hat{\delta}(S_2, a) = \{\} = S_j$
- $\hat{\delta}(S_2, b) = \{0, 3\} =: S_3$

- $\hat{\delta}(S_3, a) = \{0, 1, 3\} = S_1$
- $\hat{\delta}(S_3, b) = \{\} = S_j$
- $\hat{\delta}(S_j, a) = \{\} = S_j$
- $\hat{\delta}(S_j, b) = \{\} = S_j$

Beispiel 2.34

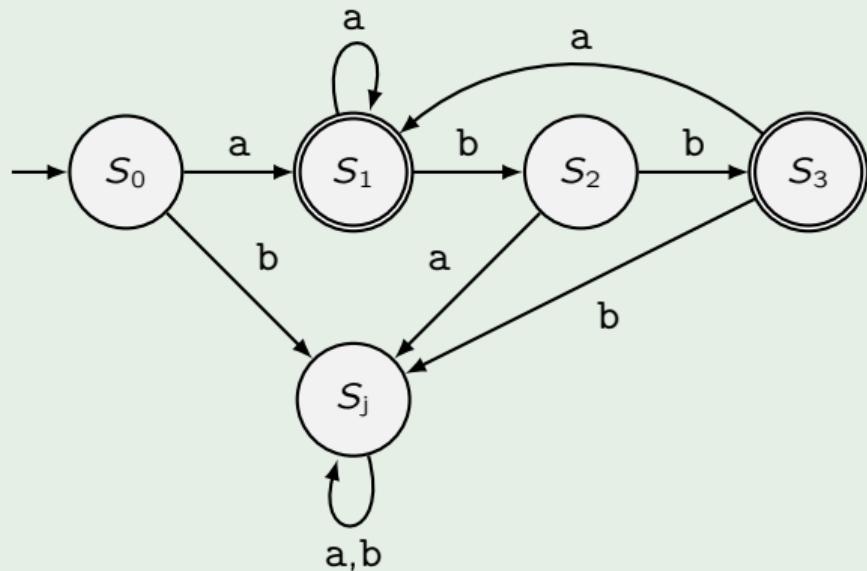
■ Tupel: $(\hat{Q}, \Sigma, \hat{\delta}, S_0, \hat{F})$

- $\hat{Q} = \{S_0, S_1, S_2, S_3, S_j\}$
- $\hat{\delta}(S_0, a) = S_1, \hat{\delta}(S_0, b) = S_j,$
 $\hat{\delta}(S_1, a) = S_1, \hat{\delta}(S_1, b) = S_2,$
 $\hat{\delta}(S_2, a) = S_j, \hat{\delta}(S_2, b) = S_3,$
 $\hat{\delta}(S_3, a) = S_1, \hat{\delta}(S_3, b) = S_j,$
 $\hat{\delta}(S_j, a) = S_j, \hat{\delta}(S_j, b) = S_j,$
- $\hat{F} = \{S_1, S_3\}$

■ Tabelle:

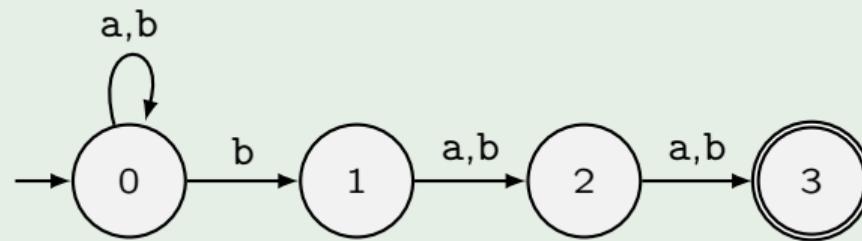
	a	b
\rightarrow	S_0	S_1 S_j
*	S_1	S_1 S_2
	S_2	S_j S_3
*	S_3	S_1 S_j
S_j	S_j	S_j

■ Graph:



Übung 2.35

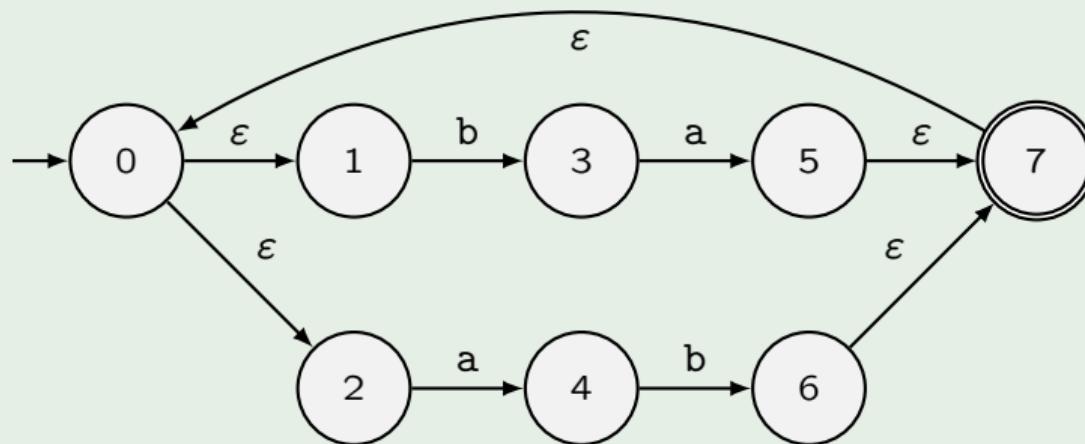
Gegeben sei der NEA \mathcal{A}_n :



- 1 Berechnen Sie $\det(\mathcal{A}_n)$.
- 2 Stellen Sie $\det(\mathcal{A}_n)$ grafisch dar.

Übung 2.36

Gegeben sei der NEA \mathcal{A} :



- 1 Berechnen Sie $\det(\mathcal{A})$.
- 2 Stellen Sie $\det(\mathcal{A})$ grafisch dar.

- Übergangs-Relation
 - kein Folgezustand,
 - ein Folgezustand,
 - mehrere Folgezustände möglich
 - zusätzlich: ϵ -Übergänge
- Akzeptanzbedingung: Existenz eines akzeptierenden Laufs
 - intuitiv: NEA rät immer richtig
- oft leichter verständlich und kompakter als DEA
- kann in DEA transformiert werden
 - Potenzmengenkonstruktion
 - exponentielle Komplexität

Inhalt

1. Einführung

2. Reguläre Sprachen und endliche Automaten

2.1 Reguläre Ausdrücke

2.2 Endliche Automaten

2.2.1 Deterministische endliche Automaten

2.2.2 Nicht-deterministische endliche Automaten

2.2.3 Endliche Automaten und reguläre Ausdrücke

2.2.4 Minimierung

2.3 Nicht-reguläre Sprachen und das Pumping-Lemma

2.4 Eigenschaften regulärer Sprachen

3. Chomsky-Grammatiken und kontextfreie Sprachen

4. Turing-Maschinen

5. Entscheidbarkeit

6. Berechenbarkeit

7. Komplexität

- Reguläre Ausdrücke **beschreiben** reguläre Sprachen
 - per Definition
 - Für jede reguläre Sprache L gibt es einen regulären Ausdruck r mit $\mathcal{L}(r) = L$
 - Für jeden regulären Ausdruck r ist $\mathcal{L}(r)$ eine reguläre Sprache
- Endliche Automaten **erkennen** reguläre Sprachen
 - Für jede reguläre Sprache L gibt es einen endlichen Automaten \mathcal{A} mit $\mathcal{L}(\mathcal{A}) = L$
 - Für jeden endlichen Automaten \mathcal{A} ist $\mathcal{L}(\mathcal{A})$ eine reguläre Sprache
 - bisher nicht bewiesen
- Konstruktiver Beweis der Äquivalenz von RA und EA
 - Transformation NEA \rightarrow DEA ✓
 - Jetzt: Transformation **RA \rightarrow NEA**
 - Später: Transformation DEA \rightarrow RA

Wiederholung: Reguläre Ausdrücke

- Elementare reguläre Ausdrücke über Σ :

- 1 \emptyset mit $\mathcal{L}(\emptyset) = \{\}$
- 2 ϵ mit $\mathcal{L}(\epsilon) = \{\epsilon\}$
- 3 $c \in \Sigma$ mit $\mathcal{L}(c) = \{c\}$

- Gegeben: reguläre Ausdrücke r_1 und r_2 über Σ .

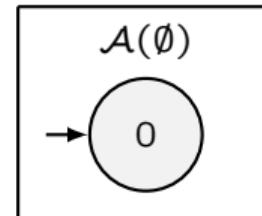
Komplexe reguläre Ausdrücke über Σ :

- 4 $r_1 + r_2$ mit $\mathcal{L}(r_1 + r_2) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2)$
- 5 $r_1 \cdot r_2$ mit $\mathcal{L}(r_1 \cdot r_2) = \mathcal{L}(r_1) \cdot \mathcal{L}(r_2)$
- 6 r_1^* mit $\mathcal{L}(r_1^*) = (\mathcal{L}(r_1))^*$

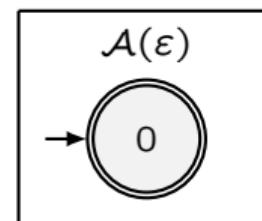
Ziel: Transformation von RA r zu NEA $\mathcal{A}(r)$ mit $\mathcal{L}(\mathcal{A}(r)) = \mathcal{L}(r)$

- Bottom-up
- Erzeuge NEA für elementare RA ($\emptyset, \epsilon, c \in \Sigma$)
- Kombiniere NEA für Teilausdrücke, um NEA für komplexe Ausdrücke ($+, \cdot, ^*$) zu erhalten

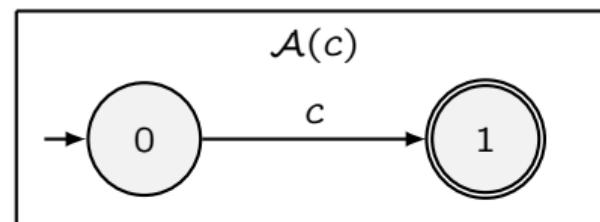
1 $\mathcal{A}(\emptyset) = (\{0\}, \Sigma, \{\}, 0, \{\})$



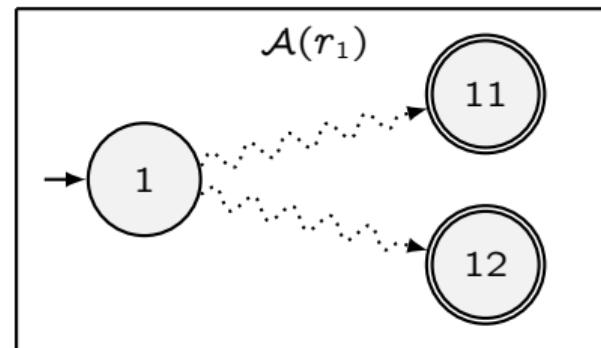
2 $\mathcal{A}(\varepsilon) = (\{0\}, \Sigma, \{\}, 0, \{0\})$



3 $\mathcal{A}(c) = (\{0, 1\}, \Sigma, \{(0, c, 1)\}, 0, \{1\})$ für alle $c \in \Sigma$

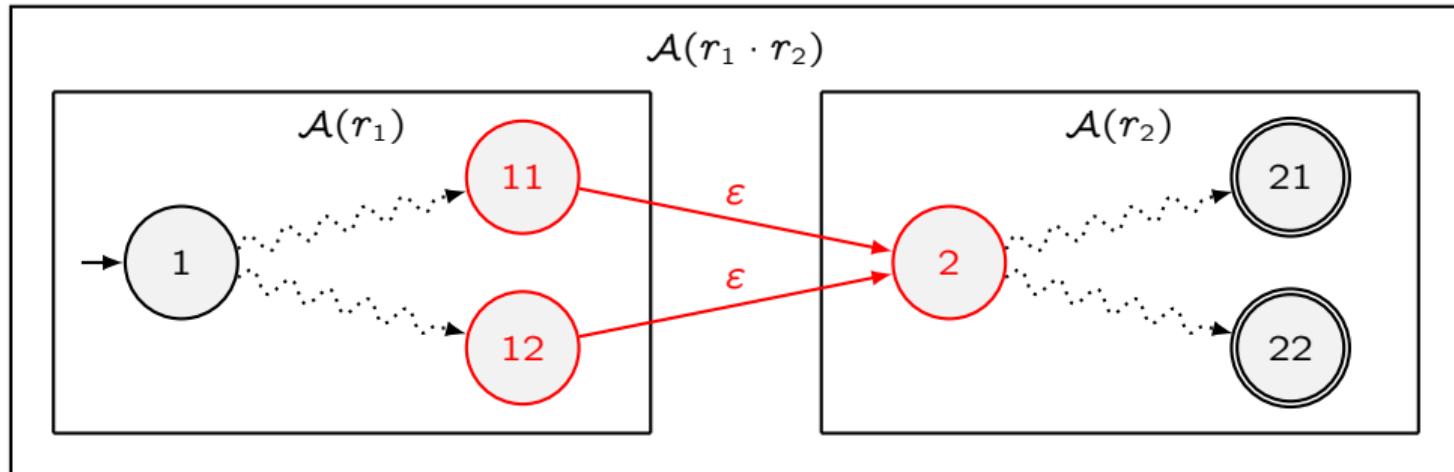


- Gegeben: Zwei RA r_1, r_2 und zugehörige NEA $\mathcal{A}(r_1), \mathcal{A}(r_2)$
- Gesucht: NEA für komplexen Ausdruck mit r_1 und r_2
- O.B.d.A. gelte
 - $\mathcal{A}(r_1) = (Q_1, \Sigma, \Delta_1, 1, \{11, 12, \dots\})$
 - $\mathcal{A}(r_2) = (Q_2, \Sigma, \Delta_2, 2, \{21, 22, \dots\})$
 - $Q_1 \cap Q_2 = \{\}$
 - $0 \notin Q_1 \cup Q_2$
- $\mathcal{A}(r_1)$ wird dargestellt als „Black Box“ mit „Anschlüssen“
 - dem (ursprünglichen) Startzustand 1 links
 - den (ursprünglichen) Endzuständen 11, 12, ... rechts (wenn vorhanden)
 - alle Übergänge und weiteren Zustände sind unbekannt und werden gepunktet dargestellt



- $\mathcal{A}(r_1) = (Q_1, \Sigma, \Delta_1, 1, \{11, 12, \dots\})$

- $\mathcal{A}(r_2) = (Q_2, \Sigma, \Delta_2, 2, \{21, 22, \dots\})$

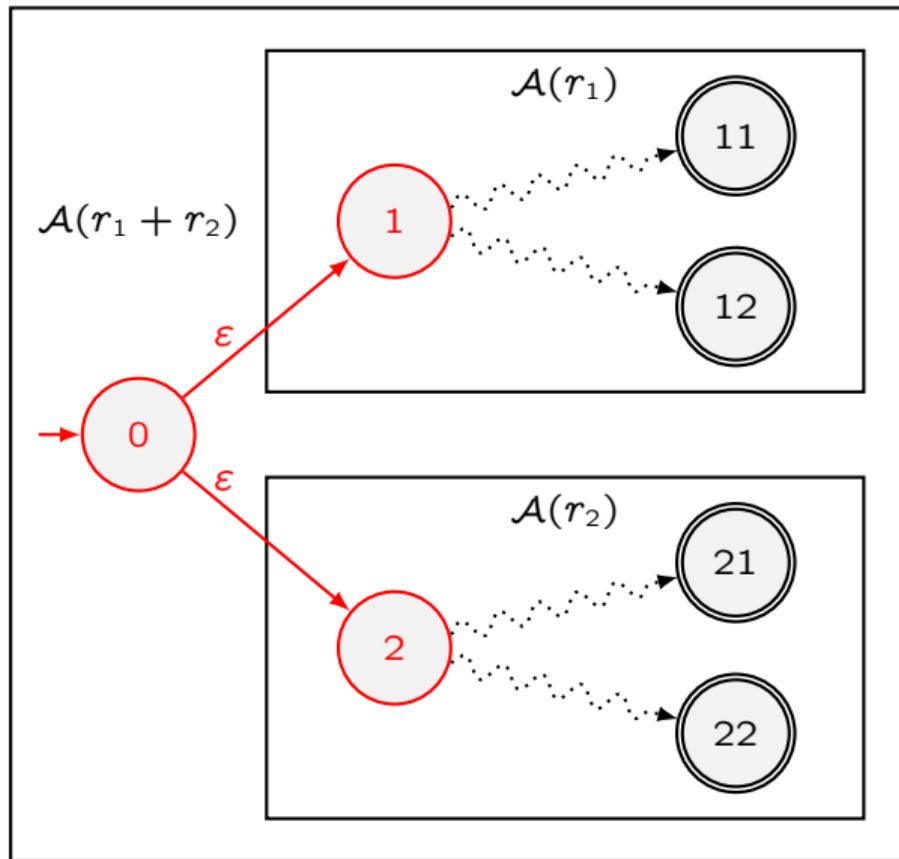


- Startzustand 1 bleibt
 - Endzustände 11 und 12 sind **keine** Endzustände mehr
 - Startzustand 2 ist **kein** Startzustand mehr
 - ε -Übergänge von 11 und 12 zu 2
 - Endzustände 21 und 22 bleiben
- 4 $\mathcal{A}(r_1 \cdot r_2) = (Q_1 \cup Q_2, \Sigma, \Delta_1 \cup \Delta_2 \cup \{(11, \varepsilon, 2), (12, \varepsilon, 2)\}, 1, \{21, 22\})$

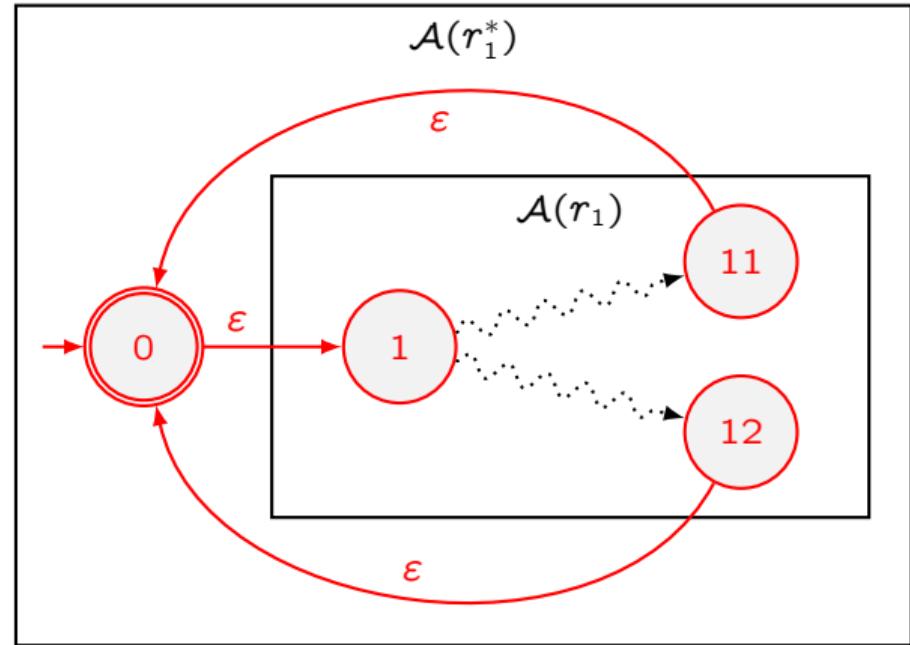
- $\mathcal{A}(r_1) = (Q_1, \Sigma, \Delta_1, 1, \{11, 12\})$
- $\mathcal{A}(r_2) = (Q_2, \Sigma, \Delta_2, 2, \{21, 22\})$

- Startzustände 1 und 2 sind keine Startzustände mehr
- Neuer Startzustand 0
- ε -Übergänge von 0 zu 1 und 2
- Endzustände 11, 12, 21 und 22 bleiben

- 5 $\mathcal{A}(r_1 + r_2) = (Q, \Sigma, \Delta, 0, F)$
 - $Q = Q_1 \cup Q_2 \cup \{0\}$
 - $\Delta = \Delta_1 \cup \Delta_2 \cup \{(0, \varepsilon, 1), (0, \varepsilon, 2)\}$
 - $F = \{11, 12, 21, 22\}$



- $\mathcal{A}(r_1) = (Q_1, \Sigma, \Delta_1, 1, \{11, 12\})$
 - Startzustand 1 ist
kein Startzustand mehr
 - Neuer Start- und Endzustand 0
 - ε -Übergang von 0 zu 1
 - Endzustände 11 und 12 sind
keine Endzustände mehr
 - ε -Übergänge von 11 und 12 zu 0
- 6 $\mathcal{A}(r_1^*) = (Q, \Sigma, \Delta, 0, \{0\})$
- $Q = Q_1 \cup \{0\}$
 - $\Delta = \Delta_1 \cup \{(0, \varepsilon, 1), (11, \varepsilon, 0), (12, \varepsilon, 0)\}$



Satz 2.37 (Kleene)

Sei r ein regulärer Ausdruck.

Dann gilt $\mathcal{L}(r) = \mathcal{L}(\mathcal{A}(r))$.

Beweis.

Induktion über den Aufbau von r .

(Umfangreich aber einfach.)



Korollar 2.38

Für jede Sprache L , die durch einen regulären Ausdruck beschrieben werden kann, gibt es einen NEA \mathcal{A} mit $\mathcal{L}(\mathcal{A}) = L$.

Übung 2.39

Verwenden Sie **exakt** den gezeigten Algorithmus, um für den RA r den NEA $\mathcal{A}(r)$ zu erzeugen.

$$r = ((a + b) + \varepsilon)a^*b$$

Tip: Prüfsumme zum Testen der korrekten Anzahl der Zustände

Leere Sprache 1 Zustand

Leeres Wort 1 Zustand

Einzelner Buchstabe 2 Zustände

Konkatenation keine zusätzlichen Zustände

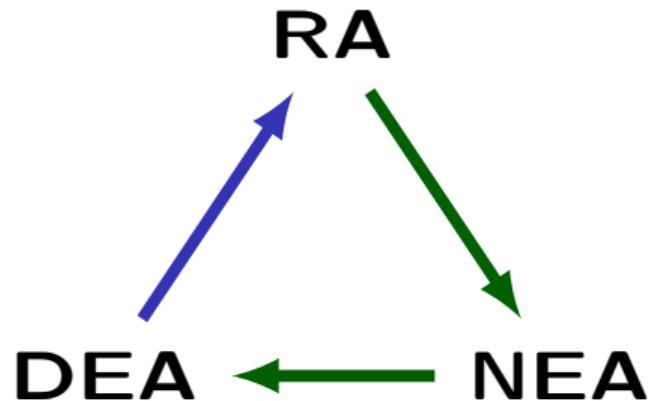
Vereinigung 1 zusätzlicher Zustand

Stern 1 zusätzlicher Zustand

Anzahl der Zustände von $\mathcal{A}(r)$:

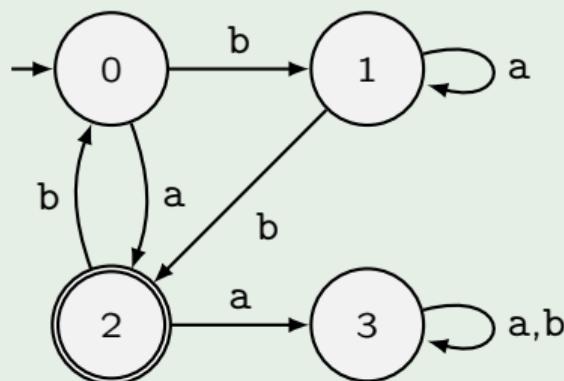
$2 \cdot \text{Anzahl der Buchstaben} + \text{Anzahl von } \emptyset, \varepsilon, +, ^*$ in r

- Behauptung: NEA, DEA und RA beschreiben dieselbe Sprachklasse
- Bereits gezeigt:
 - Transformation NEA \rightarrow DEA
 - Transformation RA \rightarrow NEA
- Jetzt: Transformation **DEA \rightarrow RA**
 - Erzeuge aus DEA \mathcal{A} regulären Ausdruck $r(\mathcal{A})$ mit
$$\mathcal{L}(r(\mathcal{A})) = \mathcal{L}(\mathcal{A})$$



- Ziel: Transformiere DEA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ in RA $r(\mathcal{A})$ mit $\mathcal{L}(r(\mathcal{A})) = \mathcal{L}(\mathcal{A})$
- Konvention:
 - Zustände heißen $\{0, 1, \dots, n\}$
 - Startzustand ist 0
- Konzept:
 - Erzeuge für jeden Zustand q eine **Gleichung** für den regulären Ausdruck r_q
 - $\mathcal{L}(r_q)$ ist die Sprache, die erkannt wird, wenn man **in q startet**,
 - abhängig von den Sprachen der Nachbar-Zustände.
 - Methode:
 - Erzeuge für jeden Übergang mit c zu p eine Alternative $c \cdot r_p$
 - Für Endzustände: zusätzlich ε
- Löse das Gleichungssystem und bestimme r_0
 - Ergebnis: $\mathcal{L}(r_0) = \mathcal{L}(\mathcal{A})$

Beispiel 2.40



Gleichungssystem:

- 1 $r_0 \equiv ar_2 + br_1$
- 2 $r_1 \equiv ar_1 + br_2$
- 3 $r_2 \equiv ar_3 + br_0 + \varepsilon$
- 4 $r_3 \equiv (a + b)r_3$

Ähnlich linearem Gleichungssystem:

- 4 Gleichungen, 4 Unbekannte
- Aber: **keine Subtraktion**
 - wie $r_1 \equiv ar_1 + br_2$ lösen?

Lemma 2.41 (Dean Arden)

Seien r , s und t reguläre Ausdrücke und gelte

- $\varepsilon \notin \mathcal{L}(s)$ und
- $r \equiv sr + t$.

Dann gilt:

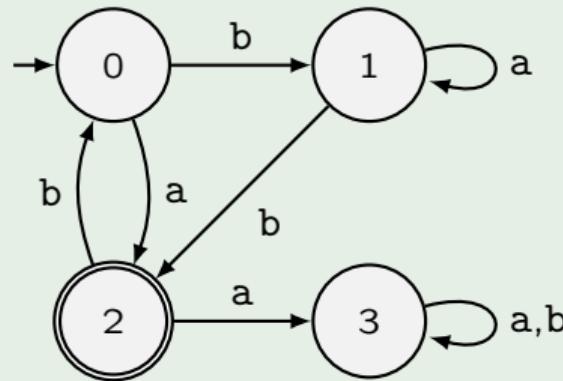
$$r \equiv s^*t$$

Intuition:

- Jedes Wort aus $\mathcal{L}(t)$ gehört zu $\mathcal{L}(r)$.
- Man kann einem Wort aus $\mathcal{L}(r)$ ein Wort aus $\mathcal{L}(s)$ voranstellen und bleibt in $\mathcal{L}(r)$.
- Dann besteht jedes Wort in $\mathcal{L}(r)$ aus der Konkatenation beliebig vieler Wörter aus $\mathcal{L}(s)$, gefolgt von einem Wort aus $\mathcal{L}(t)$.

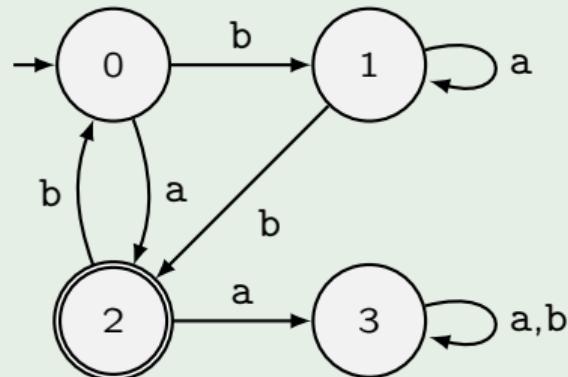


Dean N. Arden
(1925 – 2018)



- $r_0 \equiv ar_2 + br_1$
- $r_1 \equiv ar_1 + br_2$
- $r_2 \equiv ar_3 + br_0 + \varepsilon$
- $r_3 \equiv (a + b)r_3$

r_3	$\equiv (a + b)r_3 + \emptyset$	[neutrales Element]
	$\equiv (a + b)^*\emptyset$	[Arden]
	$\equiv \emptyset$	[absorbierendes Element]
r_2	$\equiv a\emptyset + br_0 + \varepsilon$	[einsetzen r_3]
	$\equiv \emptyset + br_0 + \varepsilon$	[absorbierendes Element]
	$\equiv br_0 + \varepsilon$	[neutrales Element]
r_1	$\equiv ar_1 + b(br_0 + \varepsilon)$	[einsetzen r_2]
	$\equiv a^*b(br_0 + \varepsilon)$	[Arden]



- $r_0 \equiv ar_2 + br_1$
- $r_1 \equiv a^*b(br_0 + \varepsilon)$
- $r_2 \equiv br_0 + \varepsilon$
- $r_3 \equiv \emptyset$

$$\begin{aligned}
 r_0 &\equiv a(br_0 + \varepsilon) + b(a^*b(br_0 + \varepsilon)) && [\text{einsetzen } r_1, r_2] \\
 &\equiv abr_0 + a + ba^*bbbr_0 + ba^*b && [\text{Distributivgesetz}] \\
 &\equiv (ab + ba^*bb)r_0 + a + ba^*b && [\text{Kommutativ-, Distributivgesetz}] \\
 &\equiv (ab + ba^*bb)^*(a + ba^*b) && [\text{Arden}]
 \end{aligned}$$

Ergebnis: $\mathcal{L}(\mathcal{A}) = \mathcal{L}((ab + ba^*bb)^*(a + ba^*b))$

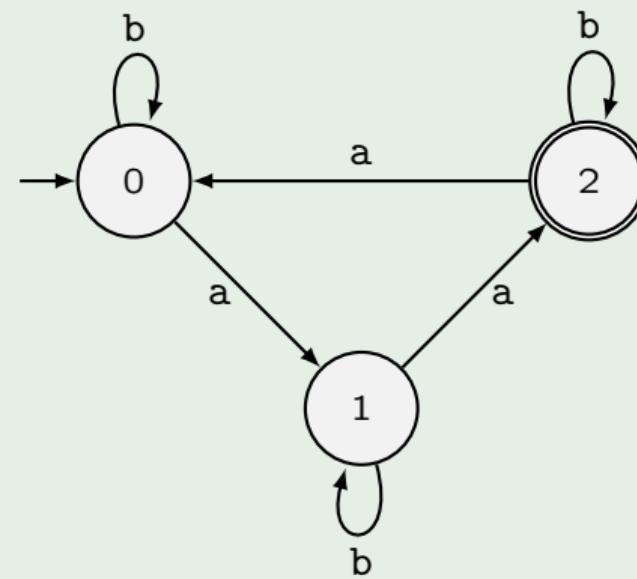
Eingabe: System mit n Gleichungen für reguläre Ausdrücke r_0 bis r_{n-1}

Ausgabe: Regulärer Ausdruck r_0

```
1: for  $i := n - 1$  to 0 do
2:   Mit der Gleichung für  $r_i$ :
3:   if  $r_i$  kommt auf der rechten Seite vor then
4:     Sortiere rechte Seite nach Termen mit und ohne  $r_i$ 
5:     Klammere  $r_i$  aus
6:     Wende Arden-Lemma an
7:   end if
8:   Ersetze  $r_i$  in allen Gleichungen  $r_0$  bis  $r_{i-1}$  durch rechte Seite
9:   // Ergebnis: Gleichungssystem ohne  $r_i$ 
10: end for
11: return  $r_0$ 
```

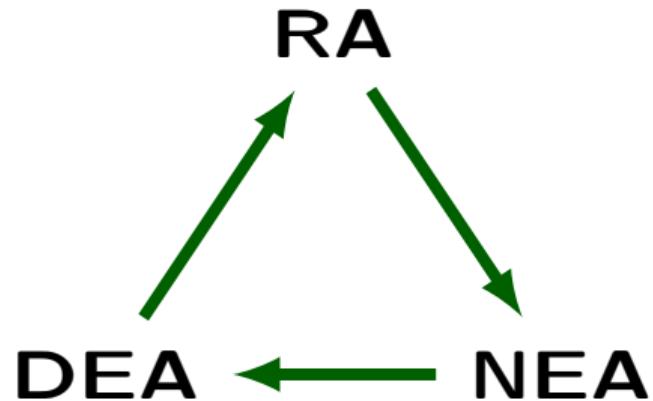
Übung 2.42

Verwenden Sie den gezeigten Algorithmus, um für den folgenden Automaten einen äquivalenten regulären Ausdruck zu erzeugen.



Transformationen

- NEA zu äquivalentem DEA ✓
 - Potenzmengenkonstruktion
- RA zu äquivalentem NEA ✓
 - ε -Übergänge
- DEA zu äquivalentem RA ✓
 - Gleichungssystem
 - Arden-Lemma



Satz 2.43 (Kleene)

Reguläre Ausdrücke, deterministische und nichtdeterministische Automaten beschreiben dieselbe Sprachklasse, nämlich die regulären Sprachen.

Inhalt

1. Einführung

2. Reguläre Sprachen und endliche Automaten

2.1 Reguläre Ausdrücke

2.2 Endliche Automaten

2.2.1 Deterministische endliche Automaten

2.2.2 Nicht-deterministische endliche Automaten

2.2.3 Endliche Automaten und reguläre Ausdrücke

2.2.4 Minimierung

2.3 Nicht-reguläre Sprachen und das Pumping-Lemma

2.4 Eigenschaften regulärer Sprachen

3. Chomsky-Grammatiken und kontextfreie Sprachen

4. Turing-Maschinen

5. Entscheidbarkeit

6. Berechenbarkeit

7. Komplexität

Anwendungsproblem: Matching mit regulären Ausdrücken

- Gegeben: Sprache als RA r
- Gewünscht: DEA \mathcal{A} zur automatischen Suche

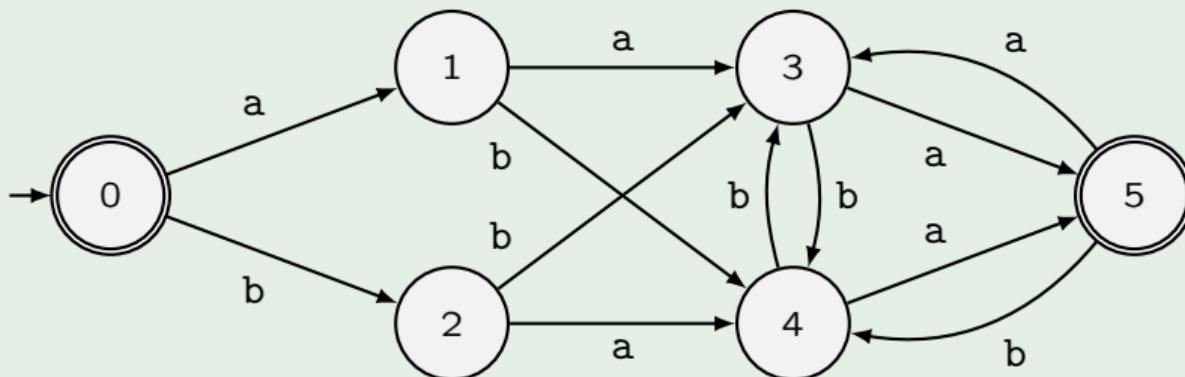
Mit bisherigen Algorithmen:

- Transformation r zu NEA $\mathcal{A}(r)$: $\mathcal{O}(|r|)$ Zustände
- Transformation von $\mathcal{A}(r)$ in DEA: $\mathcal{O}(2^{|r|})$ Zustände

Ziel:

- aus gegebenem DEA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$
- einen DEA $\mathcal{A}^- = (Q^-, \Sigma, \delta^-, q_0, F^-)$ ableiten, so dass
- $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}^-)$ gilt und
- $|Q^-|$ minimal ist
 - kein DEA mit weniger Zuständen erkennt $\mathcal{L}(\mathcal{A})$

Beispiel 2.44 (Automat \mathcal{A}_e)



Wie viele Zustände benötigen Sie für $\mathcal{L}(\mathcal{A}_e)$?

Definition 2.45 (Erreichbar)

Sei $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ ein DEA.

Ein Zustand $q \in Q$ heißt **erreichbar**, wenn es einen Lauf $((q_0, w), \dots, (q, \varepsilon))$ von \mathcal{A} auf einem Wort $w \in \Sigma^*$ gibt.

Unerreichbare Zustände

- treten häufig bei maschinell erzeugten Automaten auf
 - z.B. Potenzmengenkonstruktion
- sind offensichtlich irrelevant für erkannte Sprache
- Können leicht entfernt werden
 - 1 markiere Startzustand als erreichbar
 - 2 markiere iterativ alle Zustände als erreichbar,
zu denen es in δ einen Übergang von einem erreichbaren Zustand gibt
 - 3 wenn keine neuen erreichbaren Zustände gefunden werden,
markiere verbleibende Zustände als unerreichbar
 - 4 entferne unerreichbare Zustände aus Q , δ und F .

Zwei Zustände p, q sind **unterscheidbar**,

- Basisfall: wenn einer ein Endzustand ist, der andere nicht
- Induktiver Fall: wenn
 - es ein $c \in \Sigma$ gibt, so dass $\delta(p, c) = p', \delta(q, c) = q'$
 - und p', q' unterscheidbar sind.

Definition 2.46 (Unterscheidbare Zustände)

Sei $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ ein DEA.

Die Menge $U \subseteq Q \times Q$ der **unterscheidbaren Zustände** ist die kleinste Menge, die die folgenden Bedingungen erfüllt:

- $\{(p, q) \mid p \in F, q \notin F\} \subseteq U$
- $\{(p, q) \mid p \notin F, q \in F\} \subseteq U$
- wenn für ein $c \in \Sigma$ gilt: $\delta(p, c) = p', \delta(q, c) = q', (p', q') \in U$, dann gilt auch $(p, q) \in U$.

- Alle Paare $(p, q) \notin U$ sind äquivalent, d.h. für jedes Wort $w \in \Sigma^*$ gilt:
 - Von p ausgehend akzeptiert \mathcal{A} das Wort w gdw.
 - von q ausgehend akzeptiert \mathcal{A} das Wort w .
- Äquivalente Zustände p, q können vereinigt werden:
 - Ersetze Übergänge zu q durch Übergänge zu p
 - Entferne q (und alle von dort ausgehenden Übergänge)
- Der so erzeugte Automat wird Quotientenautomat genannt

Algorithmus zur Minimierung

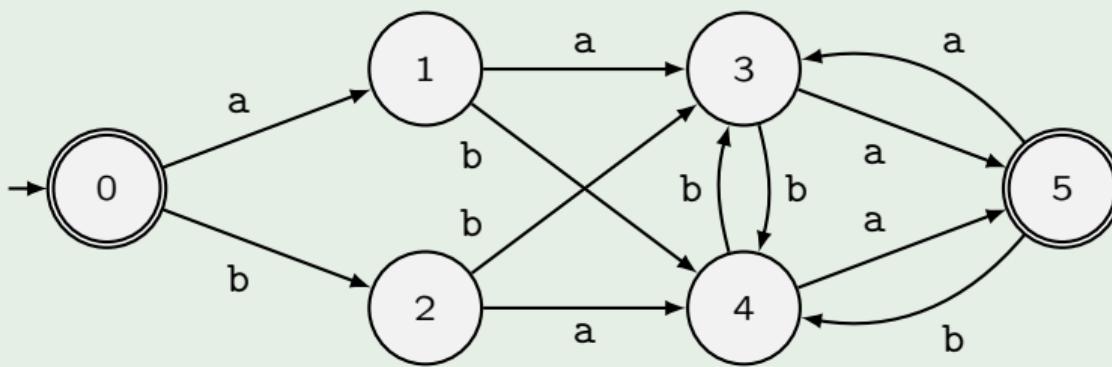
Eingabe: DEA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$

Ausgabe: Minimierter DEA

```
1:  $U := \{(p, q) \in Q \times Q \mid (p \in F \wedge q \notin F) \vee (p \notin F \wedge q \in F)\}$ 
2: repeat
3:    $U_{\text{neu}} := \{\}$ 
4:   for all  $(p, q) \notin U$  und  $c \in \Sigma$  do
5:     if  $(\delta(p, c), \delta(q, c)) \in U$  then
6:        $U_{\text{neu}} := U_{\text{neu}} \cup \{(p, q), (q, p)\}$ 
7:     end if
8:   end for
9:    $U := U \cup U_{\text{neu}}$ 
10:  until  $U_{\text{neu}} = \{\}$ 
11:  for all  $(p, q) \notin U$  do
12:    Vereinige  $p$  und  $q$ 
13:  end for
14:  return  $\mathcal{A}$ 
```

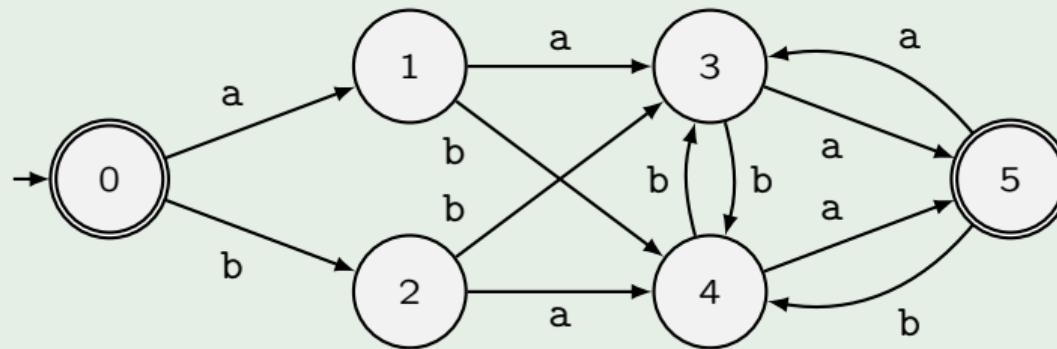
U wird dargestellt als Tabelle

- Zeilen und Spalten stehen für Zustände
- Paare in U werden mit Zahlen markiert
 - Iteration, in der Unterscheidbarkeit erkannt wurde
- Paare, die sicher **nicht** zu U gehören, werden mit \equiv markiert
 - Äquivalente Zustände



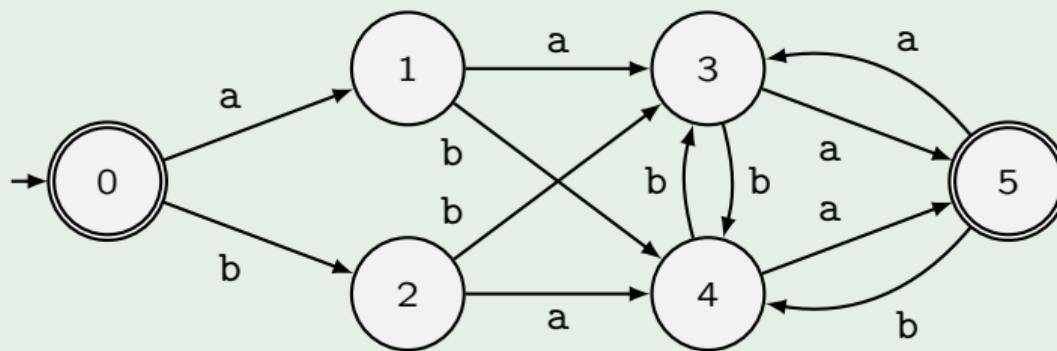
	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

- 1 Jeder Zustand ist äquivalent zu sich selbst
Paare $\{(i, i) \mid i \in \{0, \dots, 4\}\}$ gehören nicht zu U



	0	1	2	3	4	5
0	≡					
1		≡				
2			≡			
3				≡		
4					≡	
5						≡

- 2 Endzustände $F = \{0, 5\}$ sind unterscheidbar von Nichtendzuständen $Q \setminus F = \{1, 2, 3, 4\}$:



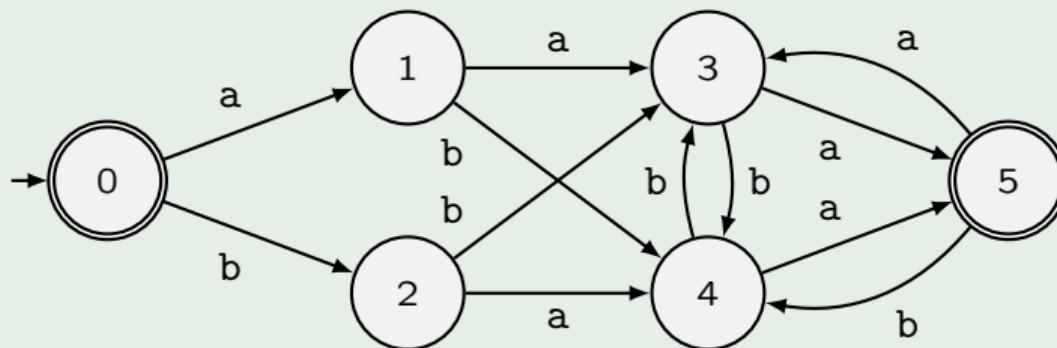
	0	1	2	3	4	5
0	\equiv	0	0	0	0	
1	0	\equiv				0
2	0		\equiv			0
3	0			\equiv		0
4	0				\equiv	0
5		0	0	0	0	\equiv

- 14 Felder bleiben frei
- 7 zu testende Paare wegen Symmetrie

Erste Iteration

- 3 Teste Übergänge von jedem Zustandspaar für jeden Buchstaben

- | | | |
|---|--|--|
| 1 | $\delta(0, a) = 1; \delta(5, a) = 3; (1, 3) \notin U$ (bisher) | $\delta(0, b) = 2; \delta(5, b) = 4; (2, 4) \notin U$ (bisher) |
| 2 | $\delta(1, a) = 3; \delta(2, a) = 4; (3, 4) \notin U$ (bisher) | $\delta(1, b) = 4; \delta(2, b) = 3; (4, 3) \notin U$ (bisher) |
| 3 | $\delta(1, a) = 3; \delta(3, a) = 5; (3, 5) \in U$ | $\rightsquigarrow \{(1, 3), (3, 1)\} \subset U$ |
| 4 | $\delta(1, a) = 3; \delta(4, a) = 5; (3, 5) \in U$ | $\rightsquigarrow \{(1, 4), (4, 1)\} \subset U$ |
| 5 | $\delta(2, a) = 4; \delta(3, a) = 5; (4, 5) \in U$ | $\rightsquigarrow \{(2, 3), (3, 2)\} \subset U$ |
| 6 | $\delta(2, a) = 4; \delta(4, a) = 5; (4, 5) \in U$ | $\rightsquigarrow \{(2, 4), (4, 2)\} \subset U$ |
| 7 | $\delta(3, a) = 5; \delta(4, a) = 5; (5, 5) \notin U$ | $\delta(3, b) = 4; \delta(4, b) = 3; (4, 3) \notin U$ (bisher) |



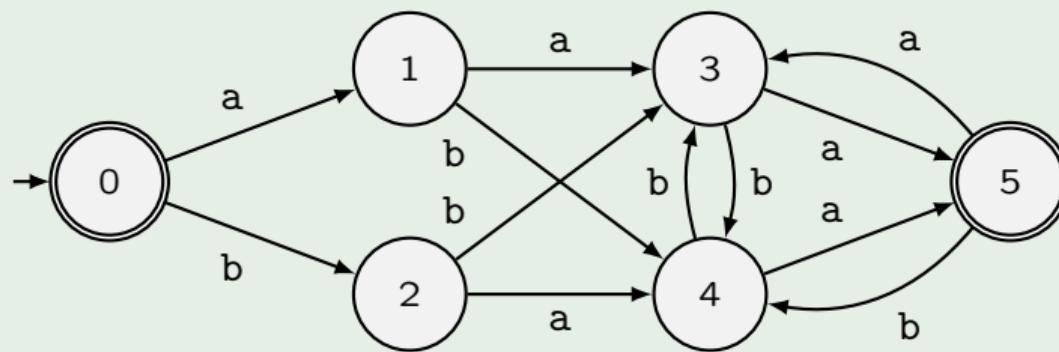
	0	1	2	3	4	5
0	\equiv	0	0	0	0	
1	0	\equiv		1	1	0
2	0		\equiv	1	1	0
3	0	1	1	\equiv		0
4	0	1	1		\equiv	0
5		0	0	0	0	\equiv

- 3 zu testende Paare

5 Teste verbleibende Paare

- 1 $\delta(0, a) = 1; \delta(5, a) = 3; (1, 3) \in U$
- 2 $\delta(1, a) = 3; \delta(2, a) = 4; (3, 4) \notin U$ (bisher)
- 3 $\delta(3, a) = 5; \delta(4, a) = 5; (5, 5) \notin U$

- $\rightsquigarrow \{(0, 5), (5, 0)\} \subset U$
- $\delta(1, b) = 4; \delta(2, b) = 3; (4, 3) \notin U$ (bisher)
 - $\delta(3, b) = 4; \delta(4, b) = 3; (4, 3) \notin U$ (bisher)



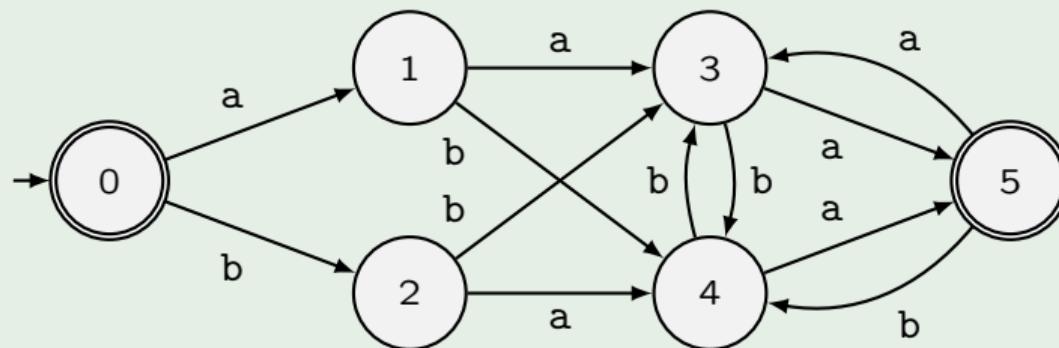
	0	1	2	3	4	5
0	\equiv	0	0	0	0	2
1	0	\equiv		1	1	0
2	0		\equiv	1	1	0
3	0	1	1	\equiv		0
4	0	1	1		\equiv	0
5	2	0	0	0	0	\equiv

■ 2 zu testende Paare

5 Teste verbleibende Paare

1 $\delta(1, a) = 3; \delta(2, a) = 4; (3, 4) \notin U$ (bisher)
 2 $\delta(3, a) = 5; \delta(4, a) = 5; (5, 5) \notin U$

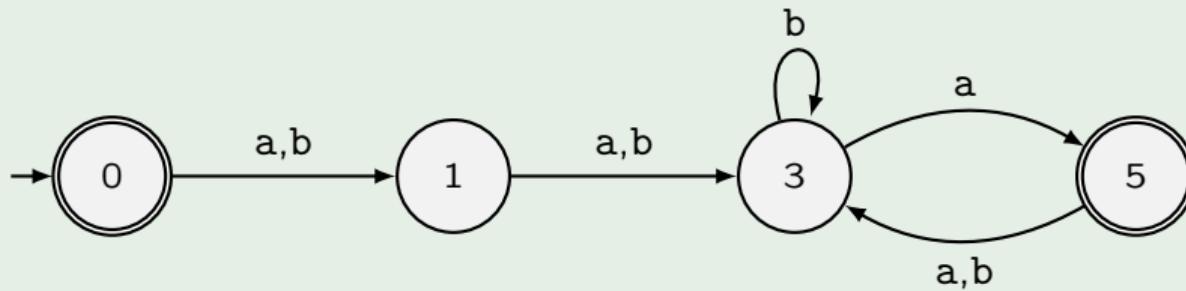
$\delta(1, b) = 4; \delta(2, b) = 3; (4, 3) \notin U$ (bisher)
 $\delta(3, b) = 4; \delta(4, b) = 3; (4, 3) \notin U$ (bisher)



	0	1	2	3	4	5
0	\equiv	0	0	0	0	2
1	0	\equiv	\equiv	1	1	0
2	0	\equiv	\equiv	1	1	0
3	0	1	1	\equiv	\equiv	0
4	0	1	1	\equiv	\equiv	0
5	2	0	0	0	0	\equiv

6 Keine neuen unterscheidbaren Paare gefunden ↵ markiere verbleibende Felder mit \equiv

- 7 Vereinige Paare (1, 2) und (3, 4)



Übung 2.47

Entwickeln Sie einen minimalen DEA für die Sprache

$$L = \mathcal{L}(a(ba)^*).$$

Drei Schritte:

- 1 Erzeugen Sie einen NEA für L .
- 2 Transformieren Sie den NEA in einen DEA.
- 3 Minimieren Sie den DEA.

Satz 2.48

Sei \mathcal{A} ein Automat ohne unerreichbare Zustände und \mathcal{A}_Q sein Quotientenautomat. Dann gilt $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_Q)$.

Beweis.

Sei $(p, q) \notin U$.

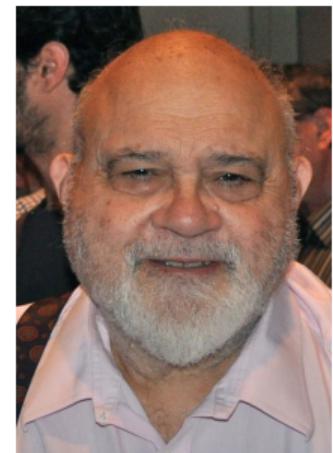
Da es kein Wort gibt, mit dem von p aus ein Endzustand und von q aus ein Nichtendzustand erreicht wird, macht es für die erkannte Sprache keinen Unterschied, ob sich \mathcal{A} im Zustand p oder q befindet. □

Satz 2.49

Der Quotientenautomat \mathcal{A}_Q ist minimal, d.h. es gibt keinen Automaten, der $\mathcal{L}(\mathcal{A}_Q)$ erkennt und weniger Zustände hat.

Beweisskizze.

- Definiere **Nerode-Rechtskongruenz**:
Äquivalenzrelation auf Wörtern über dem Alphabet Σ
 - $w_1 \sim w_2$ gdw. $\forall u \in \Sigma^* : \{w_1 \cdot u, w_2 \cdot u\} \subseteq L \vee \{w_1 \cdot u, w_2 \cdot u\} \subseteq \overline{L}$
- Jeder DEA für L hat mindestens so viele Zustände wie die Relation \sim Äquivalenzklassen hat
- Die Anzahl der Äquivalenzklassen ist gleich der Anzahl der Zustände des Quotientenautomaten □



Anil Nerode
(*1932)

Korollar 2.50

Sei L eine reguläre Sprache.

Dann gibt es einen (bis auf Zustandsumbenennung) eindeutigen minimalen DEA \mathcal{A}_L mit $\mathcal{L}(\mathcal{A}_L) = L$.

Daraus folgt:

- Der minimale DEA für eine Sprache L kann durch Minimierung eines beliebigen DEA für L gefunden werden.
- Durch systematische Benennung der Zustände kann ein kanonischer Automat für L erzeugt werden.

- algorithmisch erzeugte Automaten oft redundant
 - ε -Übergänge
 - Potenzmengenkonstruktion
- gewünscht: minimaler DEA für effiziente Implementierung
 - unerreichbare Zustände entfernen
 - Quotientenautomat
 - iterativ nach unterscheidbaren Zuständen suchen (\rightsquigarrow Tabelle)
 - nicht unterscheidbare Zustände sind äquivalent
 - äquivalente Zustände verschmelzen

1. Einführung

2. Reguläre Sprachen und endliche Automaten

2.1 Reguläre Ausdrücke

2.2 Endliche Automaten

2.3 Nicht-reguläre Sprachen und das Pumping-Lemma

2.4 Eigenschaften regulärer Sprachen

3. Chomsky-Grammatiken und kontextfreie Sprachen

4. Turing-Maschinen

5. Entscheidbarkeit

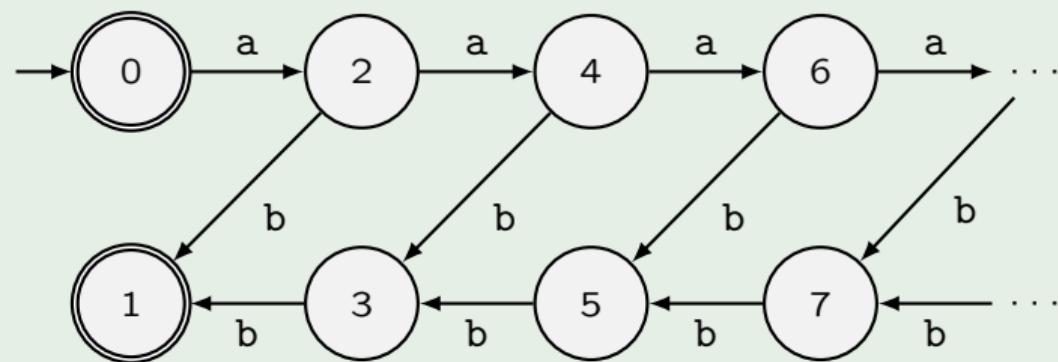
6. Berechenbarkeit

7. Komplexität

Für viele einfache Sprachen gibt es keinen offensichtlichen EA:

Beispiel 2.51 (Naiver Automat \mathcal{A} für $L = \{a^n b^n \mid n \in \mathbb{N}\}$)

\mathcal{A} hat unendlich viele Zustände:

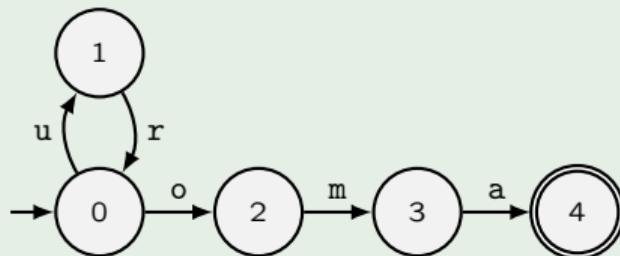


- Gibt es einen **endlichen** Automaten?
- Wenn nicht, wie kann man das beweisen?



- 1 Jede reguläre Sprache L wird von einem **endlichen** Automaten \mathcal{A}_L erkannt.
- 2 Wenn L beliebig lange Wörter enthält, muss \mathcal{A}_L einen **Zyklus** enthalten.
 - L enthält beliebig lange Wörter gdw L unendlich ist.
- 3 Wenn \mathcal{A}_L einen Zyklus enthält, kann dieser **beliebig oft** durchlaufen werden (ohne Einfluss auf Akzeptanz).

Beispiel 2.52 (Zyklischer Automat \mathcal{C})



- \mathcal{C} akzeptiert **uroma**
- \mathcal{C} akzeptiert auch **ururur...oma**

Lemma 2.53

Sei L eine reguläre Sprache.

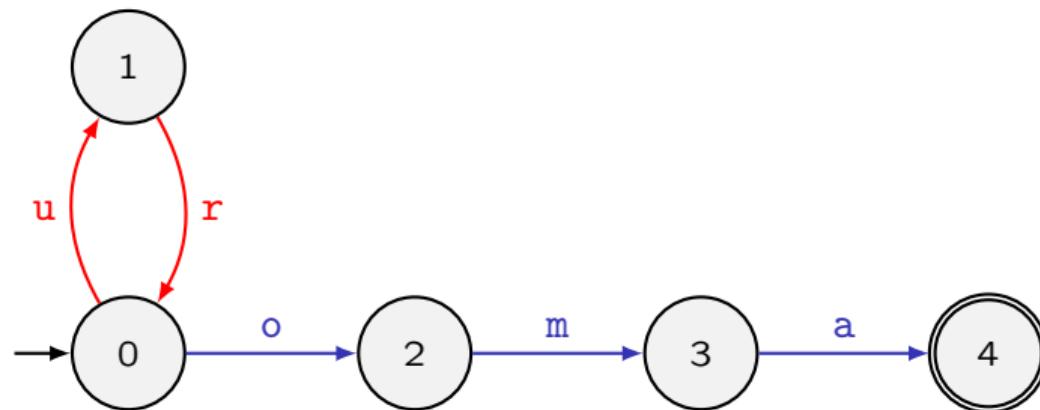
Dann gibt es ein $k \in \mathbb{N}^{\geq 1}$, so dass für jedes Wort $s \in L$ mit $|s| \geq k$ gilt:

- 1 $\exists u, v, w \in \Sigma^*(s = u \cdot v \cdot w)$,
d.h. s besteht aus Prolog u , Zyklus v und Epilog w ,
- 2 $v \neq \epsilon$,
d.h. der Zyklus hat mindestens die Länge 1,
- 3 $|u \cdot v| \leq k$,
d.h. Prolog und Zyklus zusammen haben höchstens die Länge k ,
- 4 $\forall h \in \mathbb{N}(u \cdot v^h \cdot w \in L)$,
d.h. eine beliebige Anzahl von Zyklus-Durchläufen erzeugt ein Wort der Sprache L .

Beweis.

Sei L eine reguläre Sprache und $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ ein DEA, der L erkennt.

- $k := |Q|$
- Für jedes Wort $s \in L$ mit $|s| \geq k$ gilt:
Der Lauf r von \mathcal{A} auf s enthält einen Zustand $q \in Q$ mehrfach.
 - u sei der Teil von s , der vor dem ersten Besuch von q gelesen wird;
 - v sei der Teil von s , der zwischen dem ersten und zweiten Besuch von q gelesen wird;
 - w sei der Teil von s , der nach dem zweiten Besuch von q gelesen wird.
- Dann gilt: $s = uvw$; $v \neq \epsilon$; $|uv| \leq k$.
- Dann gilt: $l = ((q_0, uw), \dots, (q, vw), \dots, (q, w), \dots, (q_f, \epsilon))$ mit $q_f \in F$.
- Dann sind weitere akzeptierende Läufe:
 - $l_0 = ((q_0, uw), \dots, (q, w), \dots, (q_f, \epsilon))$
 - $l_2 = ((q_0, uvvw), \dots, (q, vvw), \dots, (q, vw), \dots, (q, w), \dots, (q_f, \epsilon))$
 - $l_3 = ((q_0, uvvuw), \dots, (q, vvuw), \dots, (q, vvw), \dots, (q, vw), \dots, (q, w), \dots, (q_f, \epsilon))$
 - ...
- Jedes Wort $uv^h w$ mit $h \in \mathbb{N}$ wird von \mathcal{A} akzeptiert. □



- \mathcal{C} hat 5 Zustände
- $urom\alpha$ hat 5 Buchstaben
- Es gibt eine Zerlegung $s = u \cdot v \cdot w$
- so dass $|v| \neq \epsilon$
- und $|u \cdot v| \leq k$
- und $\forall h \in \mathbb{N} (u \cdot v^h \cdot w \in \mathcal{L}(\mathcal{C}))$

$$k = 5$$

$$s = urom\alpha$$

$$u = \epsilon \quad v = ur \quad w = oma$$

$$v = ur$$

$$|\epsilon \cdot ur| = 2 \leq 5$$

$$(ur)^*oma \subseteq \mathcal{L}(\mathcal{C})$$

- Das Pumping-Lemma beschreibt eine Eigenschaft **regulärer Sprachen**
 - Wenn L regulär ist, können lange Wörter aufgepumpt werden.
- Ziel: Beweis der **Irregulärität** einer Sprache
 - Wenn L die Eigenschaft X hat, dann ist L nicht regulär.
- Wie kann das Pumping-Lemma helfen?

Satz 2.54 (Kontraposition)

$$A \rightarrow B \quad \equiv \quad \neg B \rightarrow \neg A$$

Kontraposition des Pumping-Lemmas

Das Pumping-Lemma in Prädikatenlogik:

$$\text{reg}(L) \rightarrow \exists k \in \mathbb{N}^{\geq 1} \forall s \in L : (|s| \geq k \rightarrow \\ \exists u, v, w : (s = u \cdot v \cdot w \wedge v \neq \varepsilon \wedge |u \cdot v| \leq k \wedge \\ \forall h \in \mathbb{N} : (u \cdot v^h \cdot w \in L)))$$

Kontraposition des PL:

$$\neg(\exists k \in \mathbb{N}^{\geq 1} \forall s \in L (|s| \geq k \rightarrow \\ \exists u, v, w (s = u \cdot v \cdot w \wedge v \neq \varepsilon \wedge |u \cdot v| \leq k \wedge \\ \forall h \in \mathbb{N} (u \cdot v^h \cdot w \in L)))) \rightarrow \neg \text{reg}(L)$$

Nach Anwendung der de-Morgan-Regeln und aussagenlogischen Transformationen:

$$\forall k \in \mathbb{N}^{\geq 1} \exists s \in L (|s| \geq k \wedge \\ \forall u, v, w (s = u \cdot v \cdot w \wedge v \neq \varepsilon \wedge |u \cdot v| \leq k \rightarrow \\ \exists h \in \mathbb{N} (u \cdot v^h \cdot w \notin L))) \rightarrow \neg \text{reg}(L)$$

Was bedeutet das?

$$\begin{aligned} & \forall k \in \mathbb{N}^{\geq 1} \exists s \in L (|s| \geq k \wedge \\ & \forall u, v, w (s = u \cdot v \cdot w \wedge v \neq \varepsilon \wedge |u \cdot v| \leq k \rightarrow \\ & \exists h \in \mathbb{N} (u \cdot v^h \cdot w \notin L))) \rightarrow \neg \text{reg}(L) \end{aligned}$$

Wenn für jede Zahl k ein Wort s existiert, das mindestens Länge k hat,
und für jede Zerlegung $u \cdot v \cdot w$ von s (mit $v \neq \varepsilon$ und $|u \cdot v| \leq k$)
eine Zahl h existiert, so dass $u \cdot v^h \cdot w$ nicht zu L gehört,
dann ist L nicht regulär.

Wir müssen zeigen:

- Für jede natürliche Zahl k
- existiert ein Wort $s \in L$, das länger ist als k ,
- so dass jede Zerlegung $u \cdot v \cdot w = s$ mit $|u \cdot v| \leq k$ und $v \neq \varepsilon$
- zu einem Wort $u \cdot v^h \cdot w \notin L$ aufgepumpt werden kann.

Beispiel 2.55 ($L = \{a^n b^n \mid n \in \mathbb{N}\}$)

- Sei $k \in \mathbb{N}^{\geq 1}$ (beliebig).
- Wähle $s = a^k b^k$.
- Sei $s = uvw$ mit $v \neq \varepsilon$ und $|uv| \leq k$. Es gilt:

$$s = \underbrace{a^i}_u \cdot \underbrace{a^j}_v \cdot \underbrace{a^l \cdot b^k}_w$$

- $i + j + l = k$
- wegen $|u \cdot v| \leq k$ bestehen u und v nur aus a-Folgen
- aus $v \neq \varepsilon$ folgt $j \geq 1$
- Wähle $h = 0$. Dann folgt:
 - $u \cdot v^h \cdot w = u \cdot w = a^{i+l} b^k$
 - Aus $j \geq 1$ folgt $i + l < k$
 - $a^{i+l} b^k \notin L$

Vier Quantoren:

- Im Lemma:

$$\exists k \forall s \exists u, v, w \forall h (u \cdot v^h \cdot w \in L)$$

- Um Irregularität zu zeigen:

$$\forall k \exists s \forall u, v, w \exists h (u \cdot v^h \cdot w \notin L)$$

Vorgehen:

- 1 Finde ein Wort s abhängig von Mindestlänge k .
- 2 Finde ein h abhängig von der Zerlegung $u \cdot v \cdot w$.
- 3 Zeige, dass $u \cdot v^h \cdot w \notin L$ gilt.

Übung 2.56

Verwenden Sie das Pumping-Lemma, um zu zeigen, dass

$$L = \{a^n b^m \mid n < m\}$$

nicht regulär ist.

Erinnerung:

- 1 Finden Sie ein Wort s abhängig von Mindestlänge k .
- 2 Finden Sie ein h abhängig von der Zerlegung $u \cdot v \cdot w$.
- 3 Zeigen Sie, dass $u \cdot v^h \cdot w \notin L$ gilt.

Wir versuchen, Irregularität zu zeigen:

- wählen existentiell quantifizierte Variablen,
- gewinnen, wenn $u \cdot v^h \cdot w \notin L$ gilt.

Gegenspieler versucht, Regularität zu zeigen:

- wählt universell quantifizierte Variablen,
- gewinnt, wenn $u \cdot v^h \cdot w \in L$ gilt.

Übung 2.57

Spielen Sie das Pumping Game auf

<http://weitz.de/pump/>

Übung 2.58

Sei L_P die Sprache aller Wörter der Form a^p , wobei p eine Primzahl ist:

$$L_P = \{a^p \mid p \in \mathbb{P}\}.$$

Zeigen Sie, dass L_P keine reguläre Sprache ist.

Tip: Wählen Sie $h = p + 1$

Endliche Automaten können nicht beliebig hoch [zählen](#).

Beispiele 2.59 (Nicht-reguläre Sprachen mit verschachtelten Strukturen)

C für jedes { gibt es ein }

XML für jedes <token> gibt es ein </token>

LATEX für jedes \begin{env} gibt es ein \end{env}

Deutsch für jedes Subjekt gibt es ein Prädikat

Erinnern Sie sich,
wie der Krieger,
der die Botschaft,
die den Sieg,
den die Griechen bei Marathon
errungen hatten,
verkündete,
brachte,
starb!

- Jede reguläre Sprache wird von einem DEA \mathcal{A} (mit k Zuständen) erkannt.
- Pumping-Lemma: Wörter mit mindestens k Buchstaben kann man **aufpumpen**.
- Wenn man ein Wort $w \in L$ für jede Zerlegung so aufpumpen kann, dass man ein Wort $w' \notin L$ erhält, dann ist L **nicht regulär**.
 - Auf Quantoren achten!
- Praktische Bedeutung
 - EA können nicht **beliebig hoch zählen**.
 - **Verschachtelte Strukturen** sind nicht regulär.
 - Programmiersprachen
 - Natürliche Sprachen
 - Für diese Sprachen sind mächtigere Formalismen nötig (Kapitel 3).

1. Einführung

2. Reguläre Sprachen und endliche Automaten

2.1 Reguläre Ausdrücke

2.2 Endliche Automaten

2.3 Nicht-reguläre Sprachen und das Pumping-Lemma

2.4 Eigenschaften regulärer Sprachen

3. Chomsky-Grammatiken und kontextfreie Sprachen

4. Turing-Maschinen

5. Entscheidbarkeit

6. Berechenbarkeit

7. Komplexität

- Eine formale Sprache L ist **regulär**, wenn
 - es einen regulären Ausdruck r gibt mit $\mathcal{L}(r) = L$ oder
 - es einen NEA \mathcal{A} gibt mit $\mathcal{L}(\mathcal{A}) = L$ oder
 - es einen DEA \mathcal{A} gibt mit $\mathcal{L}(\mathcal{A}) = L$.
- Pumping-Lemma: Nicht alle Sprachen sind regulär
- Frage: Welche **Operationen** kann man auf reguläre Sprachen anwenden, ohne dass sie **irregulär** werden?

Konkret: Wenn L_1 und L_2 reguläre Sprachen sind, gilt das auch für

- | | |
|-------------------|---|
| $L_1 \cup L_2?$ | (Abschluss unter Vereinigung) |
| $L_1 \cap L_2?$ | (Abschluss unter Durchschnitt) |
| $L_1 \cdot L_2?$ | (Abschluss unter Konkatenation) |
| $\overline{L_1}?$ | (Abschluss unter Komplement) |
| $L_1^*?$ | (Abschluss unter Kleene-Stern) |

Satz 2.60

Seien L_1 und L_2 reguläre Sprachen.

Dann sind die folgenden Sprachen ebenfalls regulär:

- $L_1 \cup L_2$
- $L_1 \cap L_2$
- $L_1 \cdot L_2$
- $\overline{L_1}$
- L_1^*

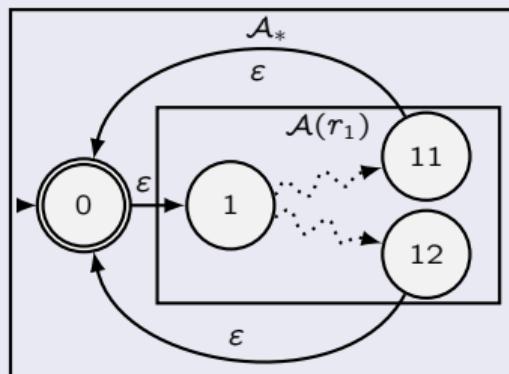
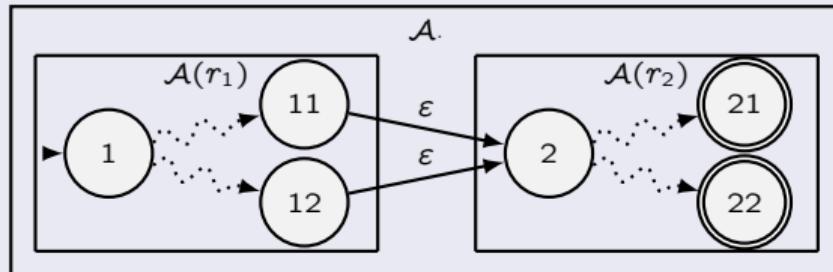
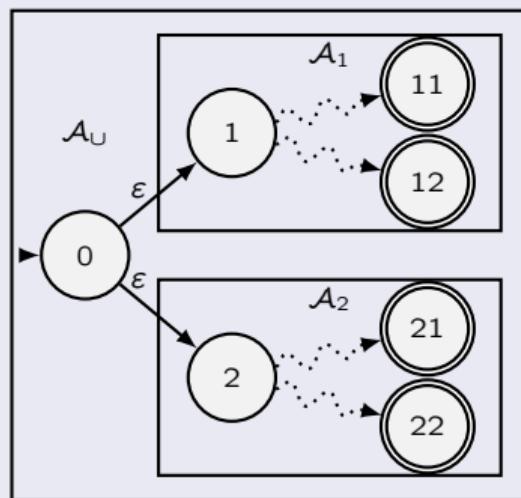
Beweisidee

- verwende (disjunkte) Automaten für L_1 und L_2
- konstruiere daraus Automat für jeweilige Sprache

Beweis.

Seien \mathcal{A}_1 und \mathcal{A}_2 Automaten für L_1 und L_2 .

Dann erkennen \mathcal{A}_\cup , $\mathcal{A}_.$ und \mathcal{A}_* die Sprachen $L_1 \cup L_2$, $L_1 \cdot L_2$ bzw. $(L_1)^*$.



- Gegeben: Automaten $\mathcal{A}_1, \mathcal{A}_2$
- Gesucht: Automat \mathcal{A}_{\cap} , für alle Wörter, die von \mathcal{A}_1 und \mathcal{A}_2 akzeptiert werden
- Konzept: **Produktautomat**
 - Zustandsmenge: **Kreuzprodukt** der Zustände von \mathcal{A}_1 und \mathcal{A}_2
 - startet in Startzuständen von \mathcal{A}_1 und \mathcal{A}_2
 - simuliert parallel Läufe von \mathcal{A}_1 (in erster Komponente) und \mathcal{A}_2 (in zweiter Komponente)
 - akzeptiert, wenn **beide** Automaten akzeptieren

Produktautomat

Definition 2.61 (Produktautomat)

Seien $\mathcal{A}_1 = (Q_1, \Sigma, \Delta_1, q_1, F_1)$ und $\mathcal{A}_2 = (Q_2, \Sigma, \Delta_2, q_2, F_2)$ NEA.

Der Automat $\mathcal{A}_n = (Q, \Sigma, \Delta, q_0, F)$ ist wie folgt definiert:

- $Q = Q_1 \times Q_2$
- $\Delta = \{((q_i, q_j), c, (q'_i, q'_j)) \mid (q_i, c, q'_i) \in \Delta_1 \wedge (q_j, c, q'_j) \in \Delta_2\}$
- $q_0 = (q_1, q_2)$
- $F = F_1 \times F_2$

Satz 2.62

Der Produktautomat für \mathcal{A}_1 und \mathcal{A}_2 erkennt die Sprache $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.

Beweis.

Induktion über Konfigurationen. □

Korollar 2.63

Die regulären Sprachen sind abgeschlossen unter Durchschnitt.

Übung 2.64

Erzeugen Sie endliche Automaten für

- 1 $L_1 = \{w \in \Sigma_{ab}^* \mid |w|_a \text{ ist ungerade}\}$
- 2 $L_2 = \{w \in \Sigma_{ab}^* \mid |w|_b \text{ ist Vielfaches von } 3\}$

Erzeugen Sie dann

- 3 den Produktautomaten für $L_1 \cap L_2$.

- Gegeben: DEA \mathcal{A}
- Gesucht: Automat $\overline{\mathcal{A}}$ für Komplement von $\mathcal{L}(\mathcal{A})$
- Konzept: **Vertausche End- und Nichtendzustände**
 - da \mathcal{A} deterministisch ist, ist er für jedes Wort w in genau **einem** Zustand
 - dieser ist in $\overline{\mathcal{A}}$ Endzustand gdw. er in \mathcal{A} kein Endzustand ist
 - funktioniert **nicht** für NEA!

Satz 2.65

Sei L eine reguläre Sprache über Σ .

Dann ist $\overline{L} = \Sigma^* \setminus L$ regulär.

Beweis.

Sei $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ ein DEA für L .

Dann erkennt der Automat $\overline{\mathcal{A}} = (Q, \Sigma, \delta, q_0, Q \setminus F)$ die Sprache \overline{L} :

- aus $w \in \mathcal{L}(\mathcal{A})$ folgt, dass der Lauf von \mathcal{A} auf w in einem Zustand $q \in F$ endet, damit gilt $q \notin Q \setminus F$ und $w \notin \mathcal{L}(\overline{\mathcal{A}})$.
- aus $w \notin \mathcal{L}(\mathcal{A})$ folgt, dass der Lauf von \mathcal{A} auf w in einem Zustand $q \notin F$ endet, damit gilt $q \in Q \setminus F$ und $w \in \mathcal{L}(\overline{\mathcal{A}})$.



Übung 2.66

Zeigen Sie, dass $L_1 = \{w \in \Sigma_{ab}^* \mid |w|_a = |w|_b\}$ nicht regulär ist.

Tip: Führen Sie einen indirekten Beweis und verwenden Sie hierzu:

- Irregularität von $L_2 = \{a^n b^n \mid n \in \mathbb{N}\}$ (Pumping-Lemma);
- Regularität von $L_3 = \mathcal{L}(a^* b^*)$ (Regulärer Ausdruck);
- eine der in diesem Kapitel gezeigten Abschlusseigenschaften.

Satz 2.67

Jede endliche Sprache, d.h. jede Sprache, die nur endlich viele Wörter enthält, ist regulär.

Beweis.

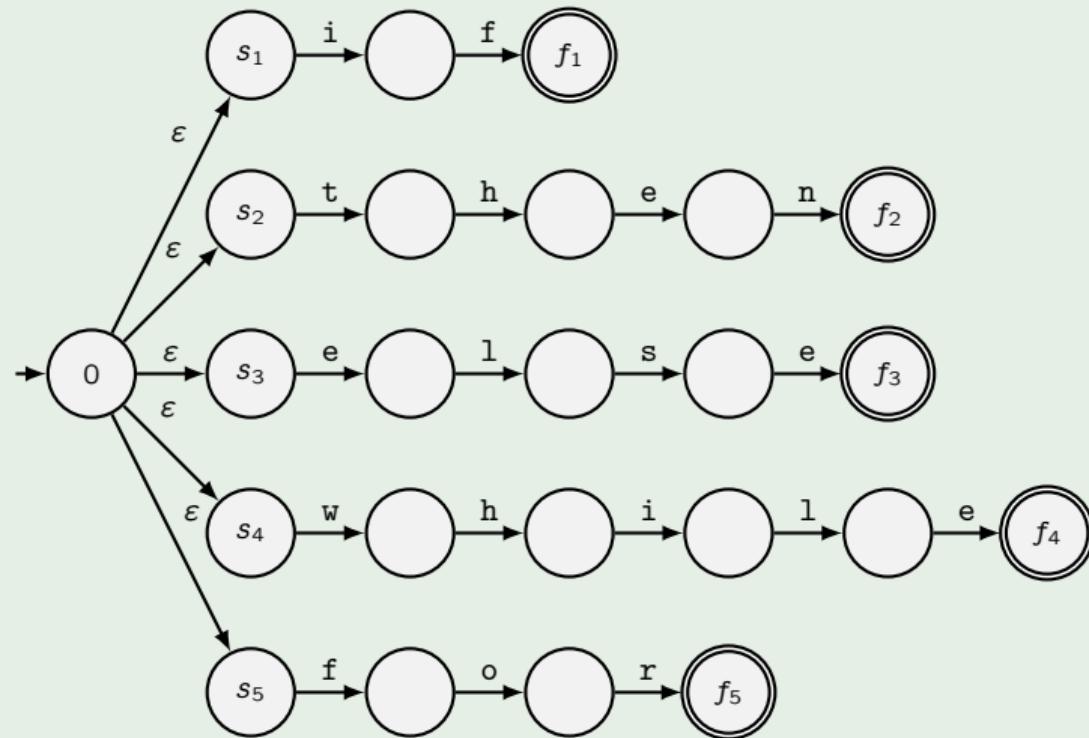
Sei $L = \{w_1, \dots, w_n\}$.

- Erzeuge für jedes w_i einen Automaten \mathcal{A}_i mit Startzustand s_i und Endzustand f_i .
- Erzeuge weiteren Zustand 0, von dem aus es zu jedem s_i einen ε -Übergang gibt.

Dann erkennt der resultierende Automat mit Startzustand 0 und Endzustandsmenge $\{f_i \mid 1 \leq i \leq n\}$ die Sprache L . □

Beispiel: Endliche Sprache

Beispiel 2.68 ($L = \{\text{if, then, else, while, for}\}$ über Σ_{ASCII})



Alternativer Beweis für Satz 2.67.

Sei $L = \{w_1, w_2, \dots, w_n\}$.

Schreibe L als regulären Ausdruck $w_1 + w_2 + \dots + w_n$. □

Korollar 2.69

Die Klasse der endlichen Sprachen wird charakterisiert durch

- azyklische NEA,
- DEA, die keine Zyklen auf Pfaden vom Start- zu einem Endzustand haben,
- reguläre Ausdrücke ohne Kleene-Stern.

Seien L_1 und L_2 reguläre Sprachen und w ein Wort.

Enthält L_1 irgendein Wort?	Leerheitsproblem
Gehört w zu L_1 ?	Wortproblem
Ist L_1 gleich L_2 ?	Äquivalenzproblem
Ist L_1 endlich?	Endlichkeitsproblem

Anwendungen:

Wortproblem Muster-Suche, Scanner

Äquivalenzproblem Beschreibt der neue und einfachere RA/EA dieselbe Sprache wie der ursprüngliche?

Satz 2.70

Das Leerheitsproblem für reguläre Sprachen ist entscheidbar.

Beweis.

Algorithmus: Sei \mathcal{A} ein NEA, der die Sprache L erkennt.

- 1 Markiere den Startzustand von \mathcal{A} als **erreichbar**
- 2 Markiere iterativ alle Zustände als erreichbar, zu denen es von einem erreichbaren Zustand mit irgendeinem Symbol einen Übergang gibt
- 3 Stoppe, wenn ein Endzustand erreicht wird oder wenn keine neuen erreichbaren Zustände gefunden werden
- 4 Falls ein Endzustand erreichbar ist: Ausgabe „nicht leer“
- 5 Sonst: Ausgabe „leer“



Satz 2.71

Das Wortproblem für reguläre Sprachen ist entscheidbar.

Beweis.

Sei $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ ein DEA, der L erkennt, und $w = c_1c_2 \dots c_n$.

Algorithmus:

- 1 $q_1 := \delta(q_0, c_1)$
- 2 $q_2 := \delta(q_1, c_2)$
- 3 \dots
- n $q_n := \delta(q_{n-1}, c_n)$

\mathcal{A} akzeptiert die Eingabe w gdw $q_n \in F$ gilt.



Man simuliert einfach den Lauf von \mathcal{A} auf w .

- Unterschiedliche reguläre Ausdrücke können dieselbe Sprache beschreiben
- Äquivalenz kann durch algebraische Regeln gezeigt werden
 - Regel muss geraten werden
 - Beweis kann sehr umfangreich werden
 - Nicht-Äquivalenz kann nicht gezeigt werden
 - \rightsquigarrow kein Entscheidungsalgorithmus

Satz 2.72

Das Äquivalenzproblem für reguläre Sprachen ist entscheidbar.

Beweis mit Korollar 2.50 aus Kapitel 2.2.4.

Seien r_1 und r_2 reguläre Ausdrücke.

- 1 Erzeuge NEA \mathcal{A}_1 und \mathcal{A}_2 mit $\mathcal{L}(r_1) = \mathcal{L}(\mathcal{A}_1)$ und $\mathcal{L}(r_2) = \mathcal{L}(\mathcal{A}_2)$.
- 2 Transformiere \mathcal{A}_1 und \mathcal{A}_2 zu DEA \mathcal{D}_1 und \mathcal{D}_2 .
- 3 Minimiere DEA \mathcal{D}_1 und \mathcal{D}_2 zu \mathcal{M}_1 und \mathcal{M}_2 .
- 4 Benenne Zustände von \mathcal{M}_1 und \mathcal{M}_2 systematisch um.

Dann gilt $r_1 \equiv r_2$ gdw $\mathcal{M}_1 = \mathcal{M}_2$ gilt. □

Alternativer Beweis mit Abschlusseigenschaften und Leerheitsproblem.

$$L_1 = L_2 \quad \text{gdw} \quad \underbrace{(L_1 \cap \overline{L_2})}_{\substack{\text{Wörter, die zu } L_1, \\ \text{aber nicht zu } L_2 \text{ gehören}}} \cup \underbrace{(\overline{L_1} \cap L_2)}_{\substack{\text{Wörter, die nicht zu } L_1, \\ \text{aber zu } L_2 \text{ gehören}}} = \{\}$$



Übung 2.73

Zeigen Sie die Äquivalenz

$$ab(ab)^* \equiv a(ba)^*b$$

unter Verwendung von endlichen Automaten.

- Bekannt: Jede endliche Sprache wird von einem azyklischen NEA erkannt.
 - aber: äquivalente zyklische Automaten möglich
 - z.B. Mülleimerzustand, unerreichbarer Zyklus
- Gesucht: Endlichkeitstest für beliebigen Automaten

Satz 2.74

Das Endlichkeitsproblem für reguläre Sprachen ist entscheidbar.

Beweis.

Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NEA.

Algorithmus:

- 1 Markiere iterativ alle von q_0 aus erreichbaren Zustände (siehe Definition 2.45).
- 2 Markiere iterativ alle Zustände, von denen aus ein $q \in F$ erreichbar ist, als terminierend.
- 3 Sei \mathcal{A}_r der Automat, der nur die erreichbaren und terminierenden Zustände von \mathcal{A} enthält.
 $\mathcal{L}(\mathcal{A})$ ist unendlich gdw \mathcal{A}_r zyklisch ist.

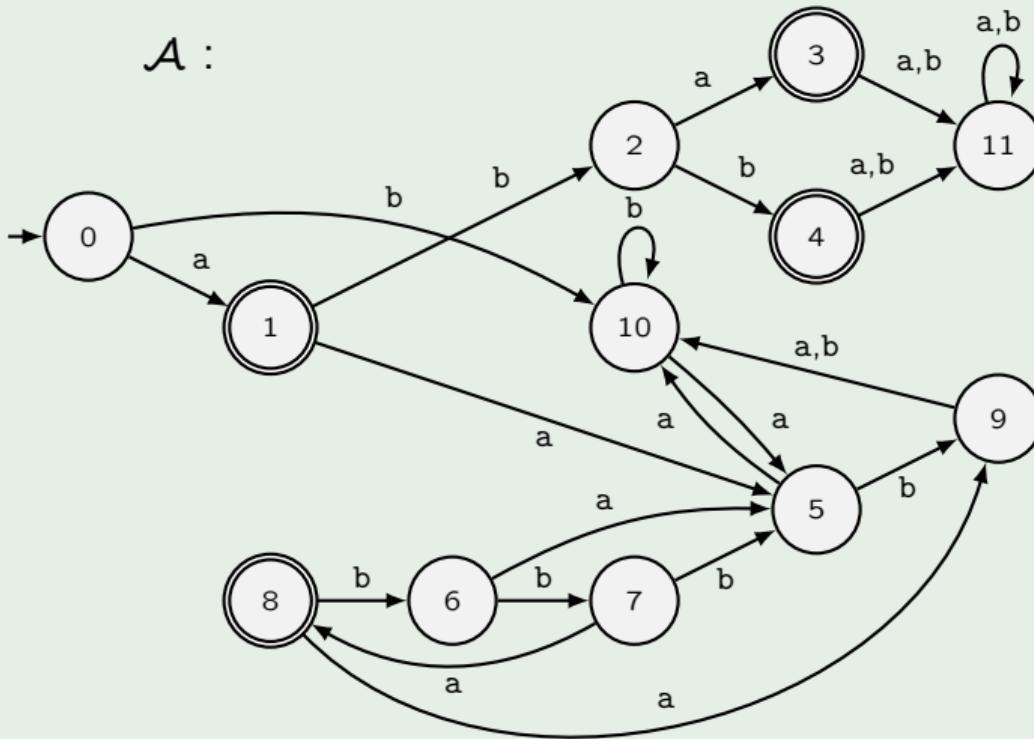


Übung 2.75

Gegeben sei der DEA \mathcal{A} .

- 1 Testen Sie mit dem gezeigten Algorithmus, ob $\mathcal{L}(\mathcal{A})$ endlich ist.
- 2 Beschreiben Sie $\mathcal{L}(\mathcal{A})$ formal als Menge.

$\mathcal{A} :$



- charakterisiert durch
 - DEA
 - NEA
 - reguläre Ausdrücke
 - rechtslineare Grammatiken
- können von einem Formalismus in einen anderen übersetzt werden
- abgeschlossen unter allen (betrachteten) Operationen
- alle (betrachteten) Probleme sind entscheidbar
- nützlich für viele Anwendungen
 - Suche nach Mustern
 - Compilerbau: Schlüsselwörter, zulässige Bezeichner
- Ausdrucksstärke begrenzt
 - kein unbeschränktes Zählen ($\{a^n b^n\}$)
 - keine verschachtelten Strukturen
 - mächtiger: Grammatiken

1. Einführung
2. Reguläre Sprachen und endliche Automaten
- 3. Chomsky-Grammatiken und kontextfreie Sprachen**
 - 3.1 Chomsky-Grammatiken
 - 3.2 Die Chomsky-Hierarchie
 - 3.3 Rechtslineare Grammatiken
 - 3.4 Kontextfreie Grammatiken
 - 3.5 Entscheidungsprobleme
 - 3.6 Kellerautomaten
 - 3.7 Eigenschaften kontextfreier Sprachen
4. Turing-Maschinen
5. Entscheidbarkeit
6. Berechenbarkeit
7. Komplexität

Inhalt

- 1. Einführung
- 2. Reguläre Sprachen und endliche Automaten
- 3. Chomsky-Grammatiken und kontextfreie Sprachen**
 - 3.1 Chomsky-Grammatiken**
 - 3.2 Die Chomsky-Hierarchie
 - 3.3 Rechtslineare Grammatiken
 - 3.4 Kontextfreie Grammatiken
 - 3.5 Entscheidungsprobleme
 - 3.6 Kellerautomaten
 - 3.7 Eigenschaften kontextfreier Sprachen
- 4. Turing-Maschinen
- 5. Entscheidbarkeit
- 6. Berechenbarkeit
- 7. Komplexität

- Reguläre Ausdrücke **beschreiben** reguläre Sprachen
- Endliche Automaten **erkennen** reguläre Sprachen

Grammatiken

- **erzeugen** formale Sprachen
- andere Sichtweise, oft intuitiver (BNF)
- für reguläre Sprachen und mächtigere Sprachklassen
- Konzept: **Regeln** für Ersetzung von Symbolen durch andere Symbole

Definition 3.1 (Grammatik)

Eine (Chomsky-)Grammatik ist ein Quadrupel

$$G = (N, \Sigma, P, S)$$

mit

- 1 einer Menge N von Nichtterminalsymbolen,
- 2 einer Menge Σ von Terminalsymbolen,
- 3 einer Menge P von (Produktions-)Regeln der Form

$$\alpha \rightarrow \beta$$

mit $V = N \cup \Sigma$, $\alpha \in V^* N V^*$, $\beta \in V^*$

- 4 dem Startsymbol $S \in N$,

so dass gilt: $N \cap \Sigma = \{\}$.

- α enthält mindestens ein NTS
- β kann leeres Wort sein



Noam Chomsky
(* 1928)

Beispiel 3.2 (C-Bezeichner)

$G_C = (N, \Sigma, P, S)$ erzeugt zulässige C-Bezeichner:

- Wörter aus Buchstaben, Ziffern und Unterstrich (_),
- die nicht mit einer Ziffer anfangen.

$N = \{S, R, B, Z\}$ (Start, Rest, Buchstabe, Ziffer),

$\Sigma = \{a, \dots, z, A, \dots, Z, 0, \dots, 9, _\}$,

$$P = \left\{ \begin{array}{l} S \rightarrow BR|_R \\ R \rightarrow BR|ZR|_R|\varepsilon \\ B \rightarrow a|\dots|z|A|\dots|Z \\ Z \rightarrow 0|\dots|9 \end{array} \right\}$$

- $\alpha \rightarrow \beta_1 | \dots | \beta_n$ ist Abkürzung für $\alpha \rightarrow \beta_1, \dots, \alpha \rightarrow \beta_n$.

Definition 3.3 (Ableitung, erzeugte Sprache)

Sei $G = (N, \Sigma, P, S)$ eine Grammatik und $x, y \in V^* = (\Sigma \cup N)^*$ Wörter.

- y ist aus x direkt ableitbar ($x \Rightarrow y$), wenn es Wörter $u, v, \alpha, \beta \in V^*$ gibt, so dass $(x = u\alpha v) \wedge (\alpha \rightarrow \beta \in P) \wedge (y = u\beta v)$ gilt.
 - y ist aus x ableitbar ($x \Rightarrow^* y$), wenn es Wörter w_0, \dots, w_n gibt, so dass $w_0 = x \wedge w_n = y \wedge w_{i-1} \Rightarrow w_i$ für alle $i \in \{1, \dots, n\}$ gilt.
 - Die von G erzeugte Sprache ist $\mathcal{L}(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$.
 - Zwei Grammatiken sind äquivalent, wenn sie dieselbe Sprache erzeugen.
-
- Wörter, die Nichtterminalsymbole enthalten
 - sind ableitbar,
 - gehören aber nicht zur erzeugten Sprache!
 - Erzeugte Sprache besteht aus den aus S ableitbaren Terminalwörtern
 - Notation:
 - Regeln verwenden →
 - Ableitungen verwenden ⇒

Beispiel 3.4 (G_3)

Sei $G_3 = (N, \Sigma, P, S)$ mit

- $N = \{S\}$,
- $\Sigma = \{a\}$,
- $P = \{S \rightarrow aS|\varepsilon\}$.

Ableitungen in G_3 haben die Form

$$S \Rightarrow aS \Rightarrow aaS \Rightarrow \dots \Rightarrow a^n S \Rightarrow a^n$$

Die von G_3 erzeugte Sprache ist **regulär**:

$$\mathcal{L}(G_3) = \{a^n \mid n \in \mathbb{N}\} = \mathcal{L}(a^*)$$

Beispiel 3.5 (G_2)

Sei $G_2 = (N, \Sigma, P, S)$ mit

- $N = \{S\}$,
- $\Sigma = \{a, b\}$,
- $P = \{S \rightarrow aSb | \varepsilon\}$

Ableitungen von G_2 :

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow \dots \Rightarrow a^n S b^n \Rightarrow a^n b^n$$

$\mathcal{L}(G_2) = \{a^n b^n \mid n \in \mathbb{N}\}$ ist **nicht** regulär

Beispiel 3.6 (G_1)

Sei $G_1 = (N, \Sigma, P, S)$ mit

- $N = \{S, T, B, C\}$,
- $\Sigma = \{a, b, c\}$,

$$\blacksquare P = \left\{ \begin{array}{ll} S \rightarrow \varepsilon & 1 \\ S \rightarrow T & 2 \\ T \rightarrow aBC & 3 \\ T \rightarrow aTBC & 4 \\ CB \rightarrow BC & 5 \end{array} \quad \begin{array}{ll} aB \rightarrow ab & 6 \\ bB \rightarrow bb & 7 \\ bC \rightarrow bc & 8 \\ cC \rightarrow cc & 9 \end{array} \right\}$$

Übung 3.7

- 1 Geben Sie Ableitungen für 3 Wörter in $\mathcal{L}(G_1)$ an.
- 2 Beschreiben Sie $\mathcal{L}(G_1)$ formal als Menge.

Beispiel 3.6 (Fortsetzung)

Ableitungen von G_1 :

$$S \Rightarrow_2 T \Rightarrow_4 aTBC \Rightarrow_4 aaTBCBC \Rightarrow_4 \dots \Rightarrow_4 a^{n-1}S(BC)^{n-1}$$

$$\Rightarrow_3 a^n(BC)^n \Rightarrow_5^* a^nB^nC^n \Rightarrow_{6,7}^* a^n b^n C^n \Rightarrow_{8,9}^* a^n b^n c^n$$

$$\mathcal{L}(G_1) = \{a^n b^n c^n \mid n \in \mathbb{N}\}.$$

- **Terminalsymbole:** Alphabet
- **Nichtterminalsymbole:** kommen in erzeugter Sprache nicht mehr vor
- **Regeln:** Ersetzen von Teilwörtern durch andere Wörter
- **Erzeugte Sprache:** ableitbare Terminalwörter
- Beispiele repräsentieren unterschiedliche **Klassen** von Grammatiken und Sprachen
- Klassen sind charakterisiert durch Eigenschaften der Regeln
- **Chomsky-Hierarchie** (1956) ordnet Klassen nach ihrer Ausdrucksstärke

1. Einführung
2. Reguläre Sprachen und endliche Automaten
- 3. Chomsky-Grammatiken und kontextfreie Sprachen**
 - 3.1 Chomsky-Grammatiken
 - 3.2 Die Chomsky-Hierarchie**
 - 3.3 Rechtslineare Grammatiken
 - 3.4 Kontextfreie Grammatiken
 - 3.5 Entscheidungsprobleme
 - 3.6 Kellerautomaten
 - 3.7 Eigenschaften kontextfreier Sprachen
4. Turing-Maschinen
5. Entscheidbarkeit
6. Berechenbarkeit
7. Komplexität

Definition 3.8 (Grammatik vom Typ 0)

Jede Chomsky-Grammatik $G = (N, \Sigma, P, S)$ ist vom Typ 0 oder unbeschränkt.

Definition 3.9 (monotone Grammatik)

Eine Chomsky-Grammatik $G = (N, \Sigma, P, S)$ ist vom Typ 1 (**monoton**), wenn alle Regeln die folgende Form haben:

$$\alpha \rightarrow \beta \quad \text{mit} \quad |\alpha| \leq |\beta|$$

Ausnahme: Die Regel $S \rightarrow \varepsilon$ ist erlaubt, wenn S auf keiner rechten Regelseite vorkommt.

- Regeln leiten keine kürzeren Wörter ab
 - Wortlänge bleibt gleich oder steigt
 - **monoton** steigende Folge
- Aber: Im ersten Schritt darf leeres Wort erzeugt werden
 - sonst gäbe es z.B. keine monotone Grammatik für $\mathcal{L}(a^*)$

Definition 3.10 (kontextfreie Grammatik)

Eine Chomsky-Grammatik $G = (N, \Sigma, P, S)$ ist vom Typ 2 (kontextfrei), wenn alle Regeln die folgende Form haben:

$$A \rightarrow \beta \text{ mit } A \in N; \beta \in V^*$$

- Regeln ersetzen nur **einzelne** Nichtterminalsymbole
 - unabhängig von ihrem Kontext
- Kürzende Regeln sind **erlaubt!**
 - kontextfreie Grammatiken sind **keine** Teilmenge der monotonen Grammatiken
 - aber: kontextfreie **Sprachen** sind Teilmenge der von monotonen Grammatiken erzeugten **Sprachen**
 - Grund: kürzende Regeln können aus kontextfreien Grammatiken entfernt werden, aber nicht aus monotonen

Definition 3.11 (rechtslineare Grammatik)

Eine Chomsky-Grammatik $G = (N, \Sigma, P, S)$ ist vom Typ 3 (rechtslinear oder regulär), wenn alle Regeln die folgende Form haben:

$$A \rightarrow cB$$

mit $A \in N; B \in N \cup \{\varepsilon\}; c \in \Sigma \cup \{\varepsilon\}$

- Linke Seite: nur ein NTS
- rechte Seite: ein TS, dann ein NTS (beide optional)
- Analogie mit Automaten ist offensichtlich

Definition 3.12 (Sprachklassen)

Eine Sprache heißt

rekursiv aufzählbar, kontextsensitiv, kontextfrei oder regulär,

wenn sie durch eine

unbeschränkte, monotone, kontextfreie bzw. rechtslineare

Grammatik erzeugt werden kann.

- Chomsky definiert zusätzlich **kontextsensitive** Grammatiken als Unterklasse der monotonen
 - zusätzliche Bedingung für kontextsensitive Regeln:
$$uAv \rightarrow u\beta v \text{ mit } A \in N; u, v \in V^*, \beta \in VV^*$$
 - nur ein **NTS** darf verändert werden (wie bei Typ 2)
 - Regel-Anwendbarkeit hängt ab von Kontext u, v
 - Kontext darf **nicht verändert** werden
 - $CB \rightarrow BC$ (Beispiel 3.6) ist **nicht** kontextsensitiv, aber monoton
- jede monotone Grammatik kann in äquivalente kontextsensitive transformiert werden
 - $CB \rightarrow BC$ wird simuliert durch $CB \rightarrow CY \rightarrow XY \rightarrow XC \rightarrow BC$
 - kontextsensitive und monotone Grammatiken erzeugen dieselbe Sprachklasse
- Terminologie der Vorlesung
 - Für Grammatiken: **monoton**, da Kontext-Bedingungen für Sprachklasse irrelevant
 - Für Sprachen: **kontextsensitiv**, da gebräuchlicher Begriff in der Literatur

Typ	Grammatik	Sprache	Maschinenmodell
0	unbeschränkt	rekursiv aufzählbar	Turingmaschine
1	monoton	kontextsensitiv	nichtdeterministischer Linear beschränkter Automat
2	kontextfrei	kontextfrei	nichtdeterministischer Kellerautomat
3	rechtslinear	regulär	endlicher Automat

Beispiel 3.2 (Fortsetzung)

$$\begin{array}{lcl} S & \rightarrow & BR|_R \\ R & \rightarrow & BR|ZR|_R|\varepsilon \\ B & \rightarrow & a|\dots|z|\bar{A}|\dots|Z \\ Z & \rightarrow & 0|\dots|9 \end{array}$$

Regeln sind kontextfrei, aber nicht rechtslinear.

Äquivalente rechtslineare Regeln:

$$\begin{array}{lcl} S & \rightarrow & AR|\dots|ZR|aR|\dots|zR|_R \\ R & \rightarrow & AR|\dots|ZR|aR|\dots|zR|0R|\dots|9R|_R|\varepsilon \end{array}$$

Beispiele 3.4 bis 3.6 (Fortsetzung)

- G_3 mit $P = \{S \rightarrow aS, S \rightarrow \epsilon\}$
 - G_3 ist rechtslinear (Typ 3).
 - $\mathcal{L}(G_3) = \{a^n \mid n \in \mathbb{N}\}$ ist regulär (Typ 3).
- G_2 mit $P = \{S \rightarrow aSb, S \rightarrow \epsilon\}$
 - G_2 ist kontextfrei (Typ 2).
 - $\mathcal{L}(G_2) = \{a^n b^n \mid n \in \mathbb{N}\}$ ist kontextfrei (Typ 2).
- G_1 mit $P = \{S \rightarrow \epsilon \mid T, T \rightarrow aBC \mid aTBC, CB \rightarrow BC, \dots\}$
 - G_1 ist monoton (Typ 1).
 - $\mathcal{L}(G_1) = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ ist kontextsensitiv (Typ 1).

Übung 3.13

Sei $G = (N, \Sigma, P, S)$ mit

- $N = \{S, A, B\}$,
- $\Sigma = \{a\}$,

- $P = \left\{ \begin{array}{ll} S \rightarrow \varepsilon & 1 \\ S \rightarrow ABA & 2 \\ AB \rightarrow aa & 3 \\ aA \rightarrow aaaA & 4 \\ A \rightarrow a & 5 \end{array} \right\}$

- 1 Was ist der höchste Typ von G ?
- 2 Geben Sie eine Ableitung des Worts aaaaa in G an.
- 3 Beschreiben Sie die Sprache $\mathcal{L}(G)$ formal als Menge.
- 4 Definieren Sie eine zu G äquivalente Grammatik G' mit dem höchsten möglichen Typ.

Übung 3.14

Eine **Oktalzahl** ist eine endliche Folge von Ziffern von 0 bis 7, die mit 0 beginnt, gefolgt von einer Ziffer ungleich 0 und anschließend beliebigen Ziffern.

Geben Sie eine rechtslineare Grammatik für die Sprache der Oktalzahlen an.

Übung 3.15

Sei $G = (N, \Sigma, P, S)$ mit

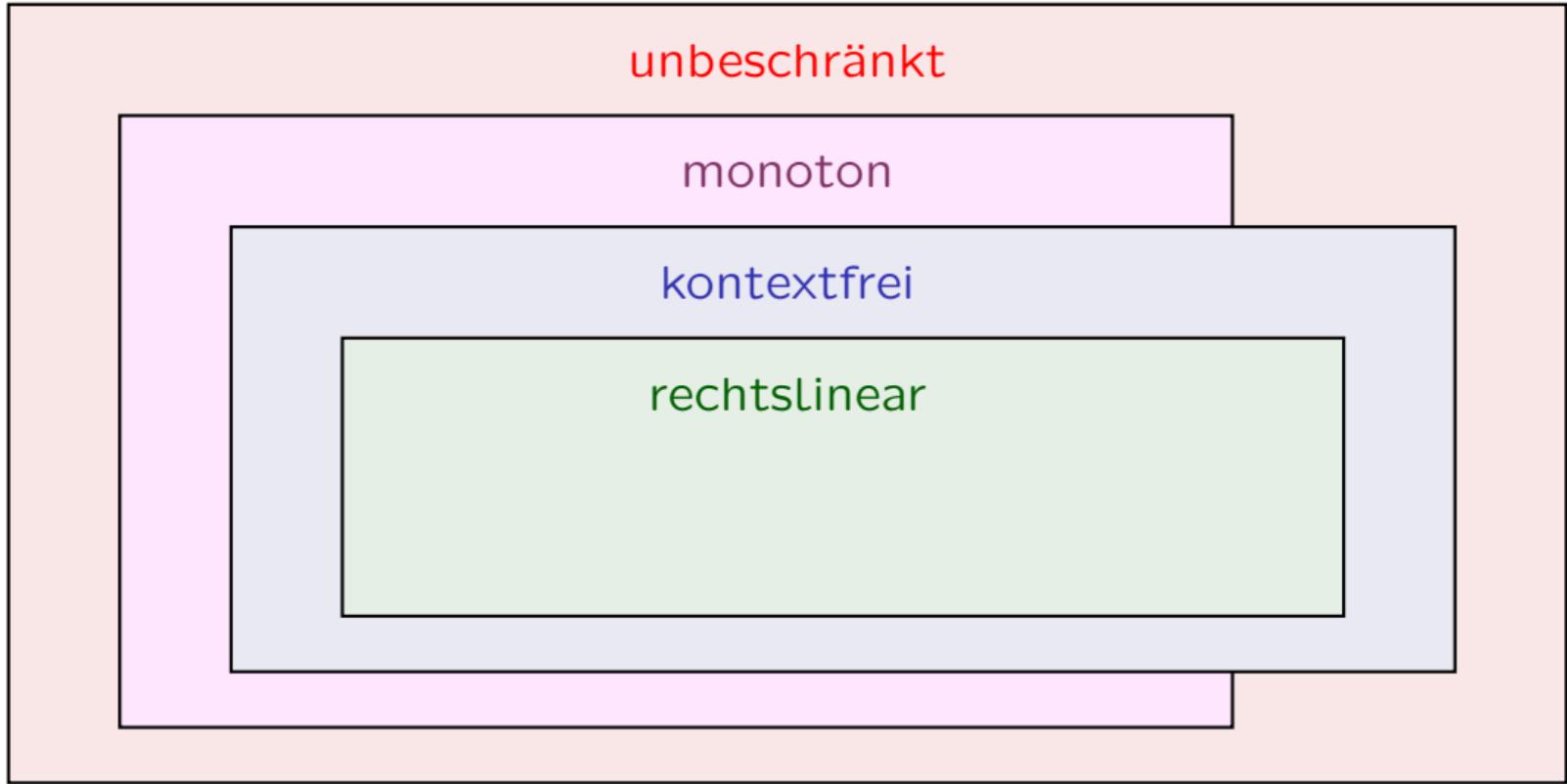
- $N = \{S, T, A, B, X, Y\}$,

- $\Sigma = \Sigma_{ab}$,

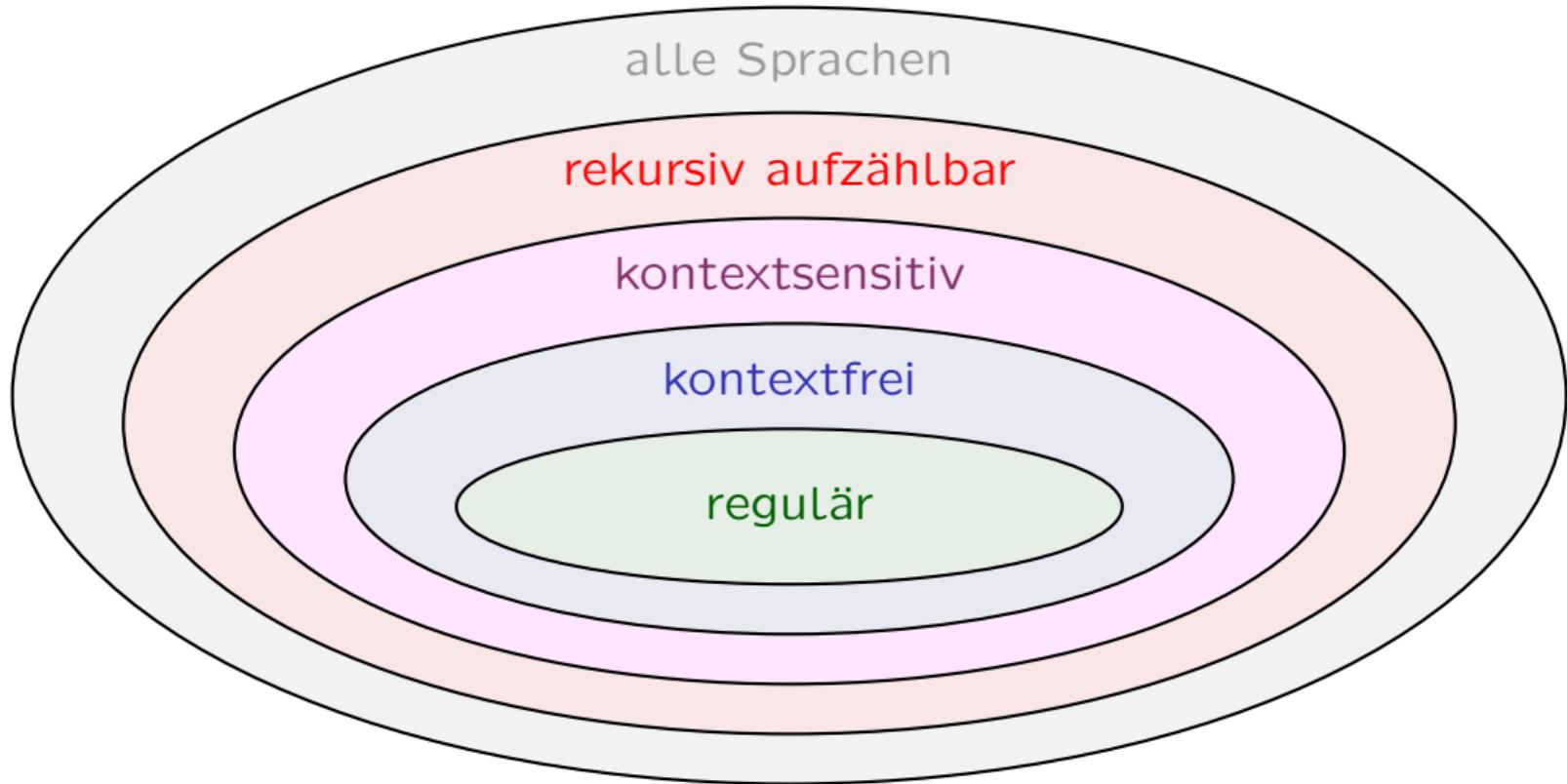
$$\blacksquare P = \left\{ \begin{array}{llll} \begin{array}{ll} S \rightarrow \varepsilon & 1 \\ S \rightarrow T & 2 \end{array} & \begin{array}{ll} T \rightarrow aAT & 3 \\ T \rightarrow bBT & 4 \\ T \rightarrow aX & 5 \\ T \rightarrow bY & 6 \end{array} & \begin{array}{ll} Aa \rightarrow aA & 7 \\ Ab \rightarrow bA & 8 \\ Ba \rightarrow aB & 9 \\ Bb \rightarrow bB & 10 \end{array} & \begin{array}{ll} AX \rightarrow Xa & 11 \\ BX \rightarrow Ya & 12 \\ AY \rightarrow Xb & 13 \\ BY \rightarrow Yb & 14 \end{array} & \begin{array}{ll} aX \rightarrow aa & 15 \\ bX \rightarrow ba & 16 \\ aY \rightarrow ab & 17 \\ bY \rightarrow bb & 18 \end{array} \\ \text{(Erzeugen)} & \text{(Sortieren)} & \text{(Umwandeln)} & \text{(Abschluss)} \end{array} \right\}$$

1 Was ist der höchste Typ von G ?

2 Beschreiben Sie die Sprache $\mathcal{L}(G)$ formal als Menge.



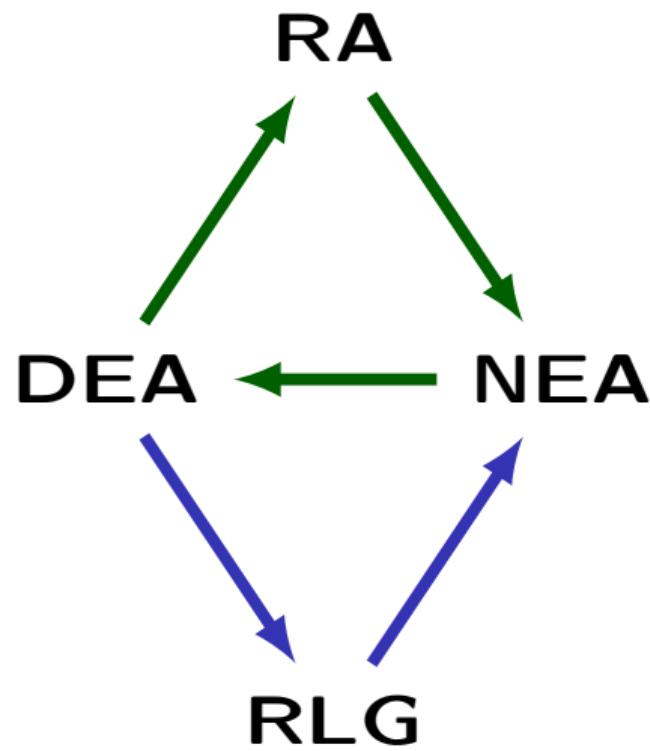
Zusammenfassung: Chomsky-Hierarchie für Sprachen



Inhalt

1. Einführung
2. Reguläre Sprachen und endliche Automaten
- 3. Chomsky-Grammatiken und kontextfreie Sprachen**
 - 3.1 Chomsky-Grammatiken
 - 3.2 Die Chomsky-Hierarchie
 - 3.3 Rechtslineare Grammatiken**
 - 3.4 Kontextfreie Grammatiken
 - 3.5 Entscheidungsprobleme
 - 3.6 Kellerautomaten
 - 3.7 Eigenschaften kontextfreier Sprachen
4. Turing-Maschinen
5. Entscheidbarkeit
6. Berechenbarkeit
7. Komplexität

- Bekannt: Äquivalenz von DEA, NEA und RA
 - Transformation RA → NEA ✓
 - Transformation NEA → DEA ✓
 - Transformation DEA → RA ✓
- Zu zeigen: Äquivalenz rechtslinearer Grammatiken
 - Transformation DEA → RLG
 - Transformation RLG → NEA



Satz 3.16

Die Klasse der regulären Sprachen ist identisch mit der Klasse der Sprachen, die von rechtslinearen Grammatiken erzeugt werden können.

Beweis.

Konstruktiv:

- Transformation von DEA zu rechtslinearer Grammatik
- Transformation einer rechtslinearen Grammatik zu NEA



Algorithmus zur Transformation eines DEA

$$\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$$

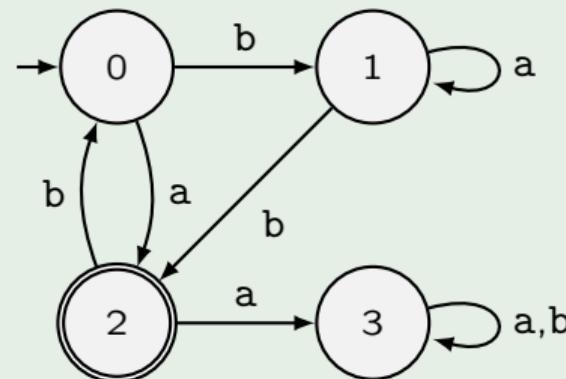
zu einer Grammatik

$$G = (N, \Sigma, P, S)$$

- $N = Q$
- $S = q_0$
- $P = \{p \rightarrow cq \mid \delta(p, c) = q\} \cup \{p \rightarrow \varepsilon \mid p \in F\}$

Übung 3.17

Gegeben sei der DEA \mathcal{A} :



Verwenden Sie den gezeigten Algorithmus, um eine rechtslineare Grammatik $G_{\mathcal{A}}$ mit $\mathcal{L}(G_{\mathcal{A}}) = \mathcal{L}(\mathcal{A})$ zu erzeugen.

Algorithmus zur Transformation einer rechtslinearen Grammatik

$$G = (N, \Sigma, P, S)$$

zu einem NEA

$$\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$$

- $Q = N \cup \{f\}$ ($f \notin N$)
- $q_0 = S$
- $F = \{f\}$
- $\Delta = \{(A, c, B) \mid A \rightarrow cB \in P\} \cup$
 $\{(A, c, f) \mid A \rightarrow c \in P\} \cup$
 $\{(A, \varepsilon, B) \mid A \rightarrow B \in P\} \cup$
 $\{(A, \varepsilon, f) \mid A \rightarrow \varepsilon \in P\}$

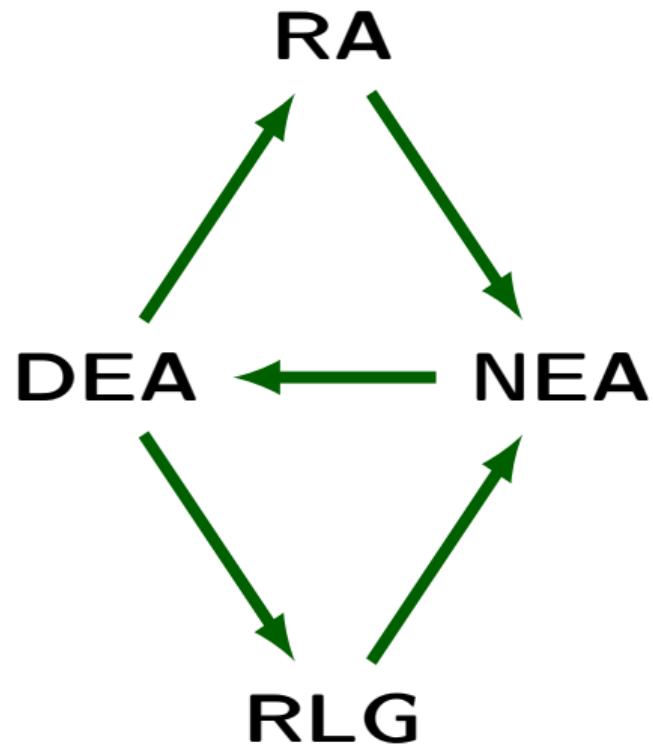
Übung 3.18

Gegeben sei die Grammatik $G = (\{S, A, B\}, \Sigma_{ab}, P, S)$, wobei P die folgenden Regeln enthält:

$$\begin{array}{l} S \rightarrow aB|\varepsilon \\ A \rightarrow aB|b \\ B \rightarrow A \end{array}$$

- 1 Verwenden Sie den gezeigten Algorithmus, um G in einen äquivalenten NEA \mathcal{A}_G zu transformieren.
- 2 Geben Sie die von \mathcal{A}_G erkannte Sprache formal als Menge an.

- Bekannt:
 - Transformation RA → NEA ✓
 - Transformation NEA → DEA ✓
 - Transformation DEA → RA ✓
 - Transformation DEA → RLG ✓
 - Transformation RLG → NEA ✓
- RA, DEA, NEA und RLG
 - sind jeweils zueinander äquivalent
 - beschreiben jeweils die Klasse der regulären Sprachen



- 1. Einführung
- 2. Reguläre Sprachen und endliche Automaten
- 3. Chomsky-Grammatiken und kontextfreie Sprachen**
 - 3.1 Chomsky-Grammatiken
 - 3.2 Die Chomsky-Hierarchie
 - 3.3 Rechtslineare Grammatiken
 - 3.4 Kontextfreie Grammatiken**
 - 3.5 Entscheidungsprobleme
 - 3.6 Kellerautomaten
 - 3.7 Eigenschaften kontextfreier Sprachen
- 4. Turing-Maschinen
- 5. Entscheidbarkeit
- 6. Berechenbarkeit
- 7. Komplexität

Wiederholung:

- $G = (N, \Sigma, P, S)$ ist kontextfrei, wenn alle Regeln in P die Form $A \rightarrow \beta$ haben mit
 - $A \in N$ und
 - $\beta \in (\Sigma \cup N)^*$
- Relevanz für praktische Anwendungen
 - Programmiersprachen (BNF)
 - XML
 - Algebraische Ausdrücke
 - Viele Aspekte natürlicher Sprache
- Effiziente Algorithmen sind wichtig
 - Transformation einer beliebigen kontextfreien Grammatik
 - ... in eine äquivalente mit besseren algorithmischen Eigenschaften

Visualisierung von Ableitungen in kontextfreien Grammatiken

Wurzel Startsymbol

innere Knoten Nichtterminalsymbole; linke Regelseite

Kindknoten rechte Regelseite

Blätter Terminalsymbole; ergeben von links nach rechts gelesen erzeugtes Wort

Beispiel 3.19

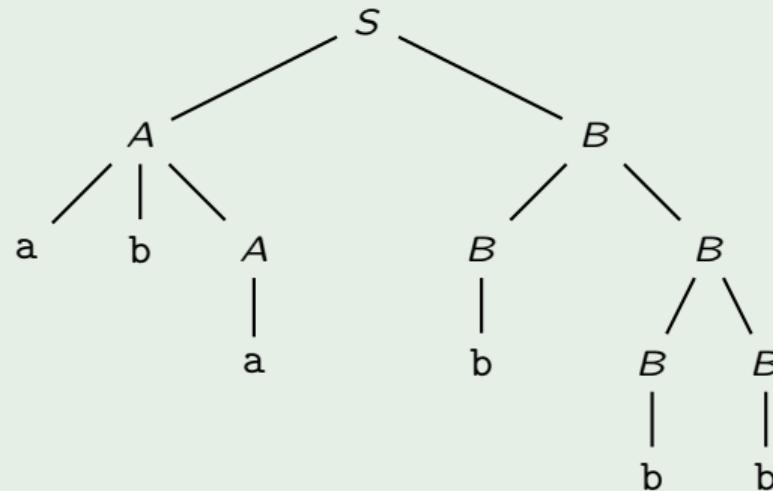
- $G = (\{S, A, B\}, \Sigma_{ab}, P, S)$ mit

$$\begin{aligned} \text{■ } P = \left\{ \begin{array}{l} S \rightarrow AB \\ A \rightarrow abA|a \\ B \rightarrow BB|b \end{array} \right\} \end{aligned}$$

Ableitung von ababbb:

$$\begin{aligned} S &\Rightarrow AB \\ &\Rightarrow abAB \\ &\Rightarrow abaB \\ &\Rightarrow abaBB \\ &\Rightarrow ababB \\ &\Rightarrow ababBB \\ &\Rightarrow ababbB \\ &\Rightarrow ababbb \end{aligned}$$

Ableitungsbäum von ababbb:



Ziel:

- möglichst wenige Arten von Regeln
- systematischer Test, ob ein Wort erzeugt werden kann
- effizienter Entscheidungsalgorithmus für Wort- und Leerheitsproblem

Definition 3.20 (Chomsky-Normalform)

Eine kontextfreie Grammatik $G = (N, \Sigma, P, S)$ ist in Chomsky-Normalform (CNF), wenn alle Regeln von einer der folgenden Formen sind:

- $N \rightarrow c$ mit $c \in \Sigma$
- $N \rightarrow AB$ mit $A, B \in N$
- $S \rightarrow \varepsilon$, wenn S auf keiner rechten Regelseite vorkommt

Transformation einer beliebigen kontextfreien Grammatik in CNF:

- 1 entferne ε -Regeln ($A \rightarrow \varepsilon$)
- 2 entferne Kettenregeln ($A \rightarrow B$)
- 3 entferne überflüssige Symbole
- 4 führe Hilfssymbole für Terminalsymbole ein
- 5 führe Hilfssymbole für zu lange rechte Regelseiten ein

Satz 3.21

Jede kontextfreie Grammatik kann in eine äquivalente ε -freie KFG transformiert werden, d.h. eine KFG, die keine Regeln der Form $A \rightarrow \varepsilon$ enthält.

(Ausnahme: $S \rightarrow \varepsilon$, wenn S auf keiner rechten Regelseite vorkommt)

Sei $G = (N, \Sigma, P, S)$.

- 1 $L := \{A \in N \mid A \rightarrow \varepsilon \in P\}$
- 2 füge zu L alle Symbole M hinzu, für es eine Regel $M \rightarrow \beta$ mit $\beta \in L^*$ gibt
- 3 Wiederhole Schritt 2, bis keine neuen Symbole gefunden werden
- 4 ersetze jede Regel $C \rightarrow w_0 B_1 w_1 B_2 w_2 \dots w_{n-1} B_n w_n$ mit $B_i \in L$ für alle $1 \leq i \leq n$ durch alle möglichen Regeln $C \rightarrow w_0 \beta_1 w_1 \beta_2 w_2 \dots w_{n-1} \beta_n w_n$ mit $\beta_i \in \{\varepsilon, B_i\}$
 - n Elemente von L auf rechter Seite $\rightsquigarrow 2^n$ Regeln
- 5 entferne alle Regeln $A \rightarrow \varepsilon$ aus P
- 6 wenn $S \in L$ gilt
 - wenn S auf keiner rechten Regelseite vorkommt: behalte Regel $S \rightarrow \varepsilon$
 - sonst: verwende neues Startsymbol S_0 , füge Regeln $S_0 \rightarrow S|\varepsilon$ zu P hinzu

- zuerst neue Regeln dazunehmen, danach ε -Regeln entfernen
- denn: neue Regeln können ε -Regeln sein

Beispiel 3.22

Sei $G = (N, \Sigma_{ab}, P, S)$ mit

- $N = \{S, A, B, C, D\}$,

$$\begin{array}{l} \boxed{\begin{array}{ll} S & \rightarrow \varepsilon | A | B | AB \\ A & \rightarrow Bb | C | a \\ B & \rightarrow AS | D | b \\ C & \rightarrow A | D | \varepsilon \\ D & \rightarrow a | b \end{array}} \end{array}$$

Schritte 1 bis 3

$$L_0 = \{S, C\}$$

$$L_1 = L_0 \cup \{A\}$$

$$L_2 = L_1 \cup \{B\}$$

$$L_3 = L_2 \cup \{\}$$

$$L = L_3 = \{S, A, B, C\}$$

$$\text{Schritt 4 } P_4 = \left\{ \begin{array}{l} S \rightarrow \varepsilon | A | B | AB \\ A \rightarrow Bb | C | a | b | \varepsilon \\ B \rightarrow AS | D | b | A | S | \varepsilon \\ C \rightarrow A | D | \varepsilon \\ D \rightarrow a | b \end{array} \right\}$$

Schritt 5

$$P_5 = \left\{ \begin{array}{l} S \rightarrow A | B | AB \\ A \rightarrow Bb | C | a | b \\ B \rightarrow AS | D | b | A | S \\ C \rightarrow A | D \\ D \rightarrow a | b \end{array} \right\}$$

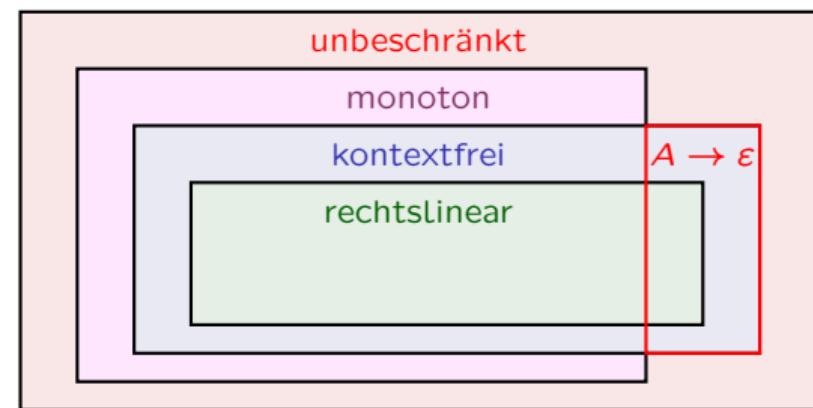
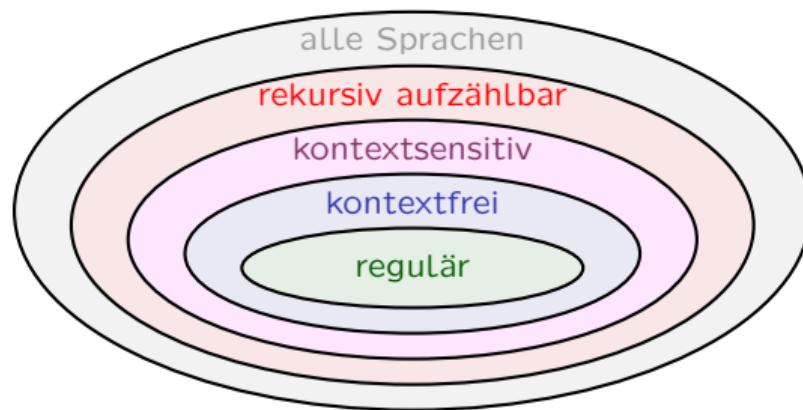
Schritt 6

$$N_L = N \cup \{S_0\}$$

$$P_L = P_5 \cup \{S_0 \rightarrow S | \varepsilon\}$$

$$G_L = (N_L, \Sigma_{ab}, P_L, S_0)$$

- Die Sprachen von Typ 0 bis Typ 3 bilden eine echte Teilmengen-Hierarchie
- Die Grammatiken nicht:
 - $A \rightarrow \epsilon$ zulässig in kontextfreien/rechtslinearen Grammatiken, aber nicht in monotonen
- Durch Entfernen der ϵ -Regeln wird dieser Konflikt gelöst



Satz 3.23

Jede ε -freie kontextfreie Grammatik kann in eine äquivalente KFG transformiert werden, die keine Regeln der Form $A \rightarrow B$ enthält.

Sei $G = (N, \Sigma, P, S)$

- 1 für jedes $A \in N$ berechne $K(A) = \{B \in N \mid A \Rightarrow^* B\}$
(iterativ, wie in den bisherigen Algorithmen)
- 2 $P_K := \{A \rightarrow \beta \mid B \rightarrow \beta \in P \text{ für ein } B \in K(A)\}$
(alle rechten Regelseiten für $K(A)$)
- 3 Entferne aus P_K alle Kettenregeln

Dann ist $G_K = (N, \Sigma, P_K, S)$ eine zu G äquivalente KFG ohne Kettenregeln.

- $K(A)$: eine Menge für **jedes** NTS!
- **zuerst** neue Regeln dazunehmen, **danach** Kettenregeln entfernen
- denn: neue Regeln können Kettenregeln sein

Beispiel 3.24

$G = (N, \Sigma_{abc}, P, S)$ mit

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

$$\begin{array}{l} \text{■ } P = \left\{ \begin{array}{l} S \rightarrow ABC \\ A \rightarrow a|B \\ B \rightarrow bb|C \\ C \rightarrow ccc \end{array} \right\} \end{array}$$

Schritt 1

$$\begin{array}{ll} K(S) &= \{S\} \\ K(A) &= \{A, B, C\} \\ K(B) &= \{B, C\} \\ K(C) &= \{C\} \end{array}$$

Schritt 2

$$P_K = \left\{ \begin{array}{l} S \rightarrow ABC \\ A \rightarrow a|B \\ B \rightarrow bb|C \\ C \rightarrow ccc \end{array} \right. \quad \left. \begin{array}{l} \text{a|B|bb|C|ccc} \\ \text{bb|C|ccc} \end{array} \right\}$$

Schritt 3

$$P_K = \left\{ \begin{array}{l} S \rightarrow ABC \\ A \rightarrow a|bb|ccc \\ B \rightarrow bb|ccc \\ C \rightarrow ccc \end{array} \right\}$$

$$G_K = (N, \Sigma_{abc}, P_K, S)$$

- Grammatiken können **unproduktive** Nichtterminalsymbole enthalten
 - kommen in keiner Ableitung eines Terminalworts vor
 - können ohne Einfluss auf die erzeugte Sprache entfernt werden
- häufig bei automatisch erzeugten Grammatiken
- ähnlich unerreichbaren und nicht terminierenden Zuständen in endlichen Automaten

Definition 3.25 (reduziert)

Sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik.

- $A \in N$ heißt **terminierend**, wenn $A \Rightarrow^* w$ für ein Wort $w \in \Sigma^*$.
- $A \in N$ heißt **erreichbar**, wenn $S \Rightarrow^* uAv$ für Wörter $\{u, v\} \subseteq V^*$.
- G heißt **reduziert**, wenn N nur erreichbare und terminierende Symbole enthält.

Algorithmus zur Berechnung der Menge der terminierenden Symbole:

- 1 $T := \{A \in N \mid \text{es gibt } A \rightarrow w \in P \text{ mit } w \in \Sigma^*\}$
- 2 füge zu T alle Symbole M hinzu, für die es eine Regel $M \rightarrow w$ mit $w \in (\Sigma \cup T)^*$ gibt
- 3 Wiederhole Schritt 2, bis keine neuen Symbole gefunden werden

Jetzt besteht T genau aus den terminierenden Symbolen.

Algorithmus zur Berechnung der Menge der erreichbaren Symbole:

- 1 $R := \{S\}$
- 2 füge zu R alle Symbole M hinzu, für die es eine Regel $A \rightarrow uMv$ mit $A \in R$ und $\{u, v\} \subseteq V^*$ gibt
- 3 Wiederhole Schritt 2, bis keine neuen Symbole gefunden werden

Jetzt besteht R genau aus den erreichbaren Symbolen.

Satz 3.26

Jede kontextfreie Grammatik G kann in eine äquivalente reduzierte kontextfreie Grammatik G_r transformiert werden.

Beweis.

- 1 Erzeuge die Grammatik G_T durch Entfernen aller **nicht terminierenden** Symbole (und der Regeln, die sie verwenden) aus G .
- 2 Erzeuge die Grammatik G_R durch Entfernen aller **unerreichbaren** Symbole (und der Regeln, die sie verwenden) aus G_T . □

Reihenfolge!

Durch Entfernen nicht terminierender Symbole können andere Symbole unerreichbar werden.

Beispiel 3.27

Sei $G = (N, \Sigma_{abc}, P, S)$ mit

- $N = \{S, A, B, C, D\}$,
- $P = \left\{ \begin{array}{l} S \rightarrow Ba|Ca|Da \\ A \rightarrow a \\ B \rightarrow bB \\ C \rightarrow c \\ D \rightarrow AB \end{array} \right\}$

- 1 $T_0 = \{A, C\}$
 - 2 $T_1 = T_0 \cup \{S\}$
 - 3 $T_2 = T_1$
 - 4 $T = \{S, A, C\}$
- \rightsquigarrow entferne B und D

$G_T = (N_T, \Sigma_{abc}, P_T, S)$ mit

- $N_T = \{S, A, C\}$
- $P_T = \left\{ \begin{array}{l} S \rightarrow Ca \\ A \rightarrow a \\ C \rightarrow c \end{array} \right\}$

- 1 $R_0 = \{S\}$
 - 2 $R_1 = R_0 \cup \{C\}$
 - 3 $R_2 = R_1$
 - 4 $R = \{S, C\}$
- \rightsquigarrow entferne A

$G_R = (N_R, \Sigma_{abc}, P_R, S)$ mit

- $N_R = \{S, C\}$
- $P_R = \left\{ \begin{array}{l} S \rightarrow Ca \\ C \rightarrow c \end{array} \right\}$

Beachte: A ist in G noch erreichbar!

(N, Σ, P, S) ist in Chomsky-Normalform, wenn alle Regeln eine der Formen haben:

- $N \rightarrow c$ mit $c \in \Sigma$
- $N \rightarrow AB$ mit $A, B \in N$
- $S \rightarrow \varepsilon$, wenn S auf keiner rechten Regelseite vorkommt

Bisher:

- ε -Regeln entfernt ✓
- Kettenregeln entfernt ✓

Aber:

- $A \rightarrow ab$: mehrere TS auf rechter Seite ✗
- $A \rightarrow aB$: TS gemeinsam mit NTS auf rechter Seite ✗
- $A \rightarrow BCD$: mehr als zwei NTS auf rechter Seite ✗

Satz 3.28

Jede kontextfreie Grammatik ohne ε -Regeln und Kettenregeln kann in eine äquivalente Grammatik in Chomsky-Normalform transformiert werden.

Beweis.

- 1 für alle Regeln $A \rightarrow \beta$ mit $\beta \notin \Sigma$:
 - ersetze auf allen rechten Regelseiten für alle $c \in \Sigma$ alle Vorkommen von c durch X_c
 - füge Regeln $X_c \rightarrow c$ hinzu
- 2 ersetze jede Regel $A \rightarrow B_1B_2 \dots B_{n-1}B_n$ für $n > 2$ durch $n - 1$ Regeln

$$\begin{array}{lll} A & \rightarrow & B_1 C_1 \\ C_1 & \rightarrow & B_2 C_2 \\ & \vdots & \\ C_{n-2} & \rightarrow & B_{n-1} B_n \end{array}$$

mit neuen Symbolen C_i .



Übung 3.29

Gegeben sei die Grammatik $G = (N, \Sigma_{ab}, P, S)$ mit

- $N = \{S, A, B, C, D, E\}$

- $P = \left\{ \begin{array}{l} S \rightarrow AB|SB|BDE \\ A \rightarrow Aa \\ B \rightarrow bB|BaB|AC|ab \\ C \rightarrow SB \\ D \rightarrow E \\ E \rightarrow \epsilon \end{array} \right\}$

Berechnen Sie die Chomsky-Normalform von G .

- Relevant für praktische Anwendung
 - Programmiersprachen, XML
 - Natürliche Sprachen
- Ableitungsbäume
- Chomsky-Normalform: Nur Regeln $A \rightarrow c$ oder $A \rightarrow BC$
 - 1 Entferne ε -Regeln
 - 2 Entferne Kettenregeln
 - 3 Reduziere Grammatik
 - 1 Entferne nicht terminierende Symbole
 - 2 Entferne unerreichbare Symbole
 - 4 Ersetze Terminalsymbole auf rechter Seite durch Nichtterminalsymbole
 - 5 Kürze lange rechte Seiten mit Hilfssymbolen

Inhalt

1. Einführung
2. Reguläre Sprachen und endliche Automaten
- 3. Chomsky-Grammatiken und kontextfreie Sprachen**
 - 3.1 Chomsky-Grammatiken
 - 3.2 Die Chomsky-Hierarchie
 - 3.3 Rechtslineare Grammatiken
 - 3.4 Kontextfreie Grammatiken
 - 3.5 Entscheidungsprobleme**
 - 3.6 Kellerautomaten
 - 3.7 Eigenschaften kontextfreier Sprachen
4. Turing-Maschinen
5. Entscheidbarkeit
6. Berechenbarkeit
7. Komplexität

Beliebige KFG

- Ableitung von w mit $|w| = n$ kann beliebig lang werden
 - Kettenregeln \rightsquigarrow Endlosschleife ($C \rightarrow B; B \rightarrow C$)
 - ε -Regeln \rightsquigarrow Zwischenschritte können länger als n sein
- kein effektiver Entscheidungsalgorithmus für das Wortproblem

Grammatik in CNF

- Ableitung von w hat genau $2n - 1$ Schritte
 - $n - 1$ Regelanwendungen $A \rightarrow BC$
 - n Regelanwendungen $A \rightarrow c$
- Entscheidungsalgorithmus für Wortproblem: Teste alle Ableitungen der Länge $2n - 1$
 - Worst-Case-Komplexität: exponentiell (2^{2n-1})
- Effizienter (kubisch): [Cocke-Younger-Kasami-Algorithmus \(CYK\)](#)

Der Cocke-Younger-Kasami-Algorithmus

- unabhängig entwickelt von
 - Tadao Kasami (1965)
 - Daniel Younger (1967)
 - John Cocke (1970)
- entscheidet das Wortproblem für kontextfreie Grammatiken in CNF
 - Nur Ja/Nein-Antwort
 - keine Ableitung
- Komplexität: $|w|^3 \cdot |P|$ (kubisch)
- Prinzip: **Dynamische Programmierung**
 - Gesamtlösung (Herleitbarkeit von w) wird zurückgeführt auf Teillösungen (Herleitbarkeit von Teilwörtern)
- Datenstruktur: $n \times n$ -Tabelle ($n = |w|$)
 - Feld (i, j) enthält NTS, die $w[i] \dots w[j]$ erzeugen
 - Hauptdiagonale: Einzelne Buchstaben ($w[1], \dots, w[n]$)
 - Feld $(1, n)$: Gesamtes Wort w
- **Hauptdiagonale:** durch Regeln $A \rightarrow c$
- **Nebendiagonalen:** iterativ durch Regeln $A \rightarrow BC$
- Feld $(1, n)$ enthält S gdw. $w \in \mathcal{L}(G)$ gilt



Tadao Kasami
(1930–2007)



Daniel Younger



John Cocke
(1925–2002)

Beispiel 3.30

$w = abacba$

Regeln:

$$\begin{array}{l} S \rightarrow a \\ B \rightarrow b \\ B \rightarrow c \\ S \rightarrow SA \\ A \rightarrow BS \\ B \rightarrow BB \\ B \rightarrow BS \end{array}$$

$i \setminus j$	1	2	3	4	5	6
1	S	/	S	/	/	S
2		B	A, B	B	B	A, B
3			S	/	/	S
4				B	B	A, B
5					B	A, B
6						S
$w =$	a	b	a	c	b	a

Umgekehrte Regeln:

$$\begin{array}{l} SA \leftarrow S \\ BS \leftarrow A, B \\ BB \leftarrow B \end{array}$$

CYK-Algorithmus iterativ

Eingabe: Wort $w = c_1 \cdot \dots \cdot c_n$, Grammatik $G = (N, \Sigma, P, S)$ in CNF

Ausgabe: „ja“, wenn $w \in \mathcal{L}(G)$ gilt; sonst „nein“

```
1: for  $i := 1$  to  $n$  do
2:    $N_{i,i} := \{A \mid A \rightarrow c_i \in P\}$ 
3: end for
4: for  $d := 1$  to  $n - 1$  do
5:   for  $i := 1$  to  $n - d$  do
6:      $j := i + d$ 
7:      $N_{i,j} := \{\}$ 
8:     for  $k := i$  to  $j - 1$  do
9:        $N_{i,j} := N_{i,j} \cup \{A \mid A \rightarrow BC \in P; B \in N_{i,k}; C \in N_{k+1,j}\}$ 
10:    end for
11:   end for
12: end for
13: if  $S \in N_{1,n}$  then
14:   return „ja“
15: else
16:   return „nein“
17: end if
```

CYK-Algorithmus rekursiv

Funktion $g(w, G)$

Eingabe: Wort w , Grammatik $G = (N, \Sigma, P, S)$ in CNF

Ausgabe: Menge $R \subseteq N$, die w erzeugen können

```
1: R := {}
2: n := |w|
3: if n = 1 then
4:     for all A ∈ N do
5:         if A → w ∈ P then
6:             R := R ∪ {A}
7:         end if
8:     end for
9: else
10:    for i := 1 to n - 1 do
11:        if A → BC ∈ P and B ∈ g(w[1, i], G) and C ∈ g(w[i + 1, n], G) then
12:            R := R ∪ {A}
13:        end if
14:    end for
15: end if
16: return R
```

Übung 3.31

Gegeben sei die Grammatik $G = (N, \Sigma_{ab}, P, S)$

- $N = \{S, A, B, D, X, Y\}$

- $P = \left\{ \begin{array}{l} S \rightarrow SB|BD|YB|XY \\ B \rightarrow BD|YB|XY \\ D \rightarrow XB \\ X \rightarrow a \\ Y \rightarrow b \end{array} \right\}$

Entscheiden Sie mit Hilfe des CYK-Algorithmus, ob die folgenden Wörter von G erzeugt werden können.

- 1 $w_1 = \text{babaaab}$
- 2 $w_2 = \text{abba}$

Satz 3.32

Es ist entscheidbar, ob für ein Wort w und eine kontextfreie Grammatik G gilt: $w \in \mathcal{L}(G)$.

Beweis.

Der CYK-Algorithmus entscheidet das Wortproblem in kubischer Zeit. □

Satz 3.33

Es ist entscheidbar, ob für eine kontextfreie Grammatik G gilt: $\mathcal{L}(G) = \{\}$.

Beweis.

Sei $G = (N, \Sigma, P, S)$.

Bestimme die Menge T der terminierenden Nichtterminalsymbole von G (Satz 3.26).

Dann gilt $\mathcal{L}(G) = \{\}$ gdw $S \notin T$ gilt. □

Satz 3.34

Es ist **unentscheidbar**, ob für zwei kontextfreie Grammatiken G_1, G_2 gilt: $\mathcal{L}(G_1) = \mathcal{L}(G_2)$.

Wie kann man beweisen, dass es keinen Algorithmus gibt?

- **Formaler** Algorithmus-Begriff nötig
- Beweis in Kapitel 5.3

- Wortproblem
- beliebige Grammatik: keine Obergrenze für Länge der Ableitung \rightsquigarrow
kein direkter Entscheidungsalgorithmus
 - Grammatik in CNF: in $\mathcal{O}(n^3)$ entscheidbar mit CYK-Algorithmus

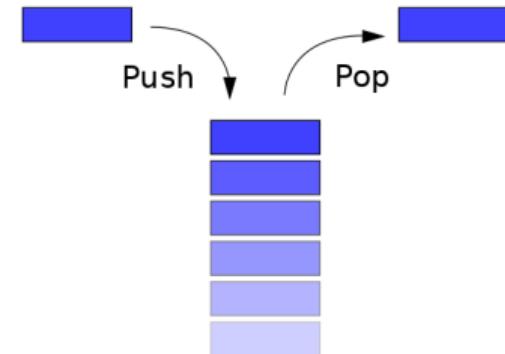
Leerheitsproblem in $\mathcal{O}(n)$ entscheidbar durch Bestimmen terminierender Symbole

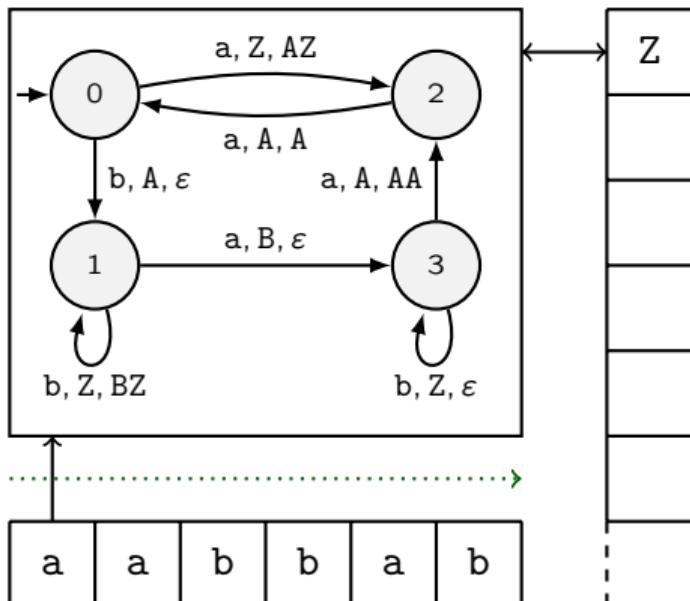
Äquivalenzproblem unentscheidbar

Inhalt

- 1. Einführung
- 2. Reguläre Sprachen und endliche Automaten
- 3. Chomsky-Grammatiken und kontextfreie Sprachen**
 - 3.1 Chomsky-Grammatiken
 - 3.2 Die Chomsky-Hierarchie
 - 3.3 Rechtslineare Grammatiken
 - 3.4 Kontextfreie Grammatiken
 - 3.5 Entscheidungsprobleme
- 3.6 Kellerautomaten**
- 3.7 Eigenschaften kontextfreier Sprachen
- 4. Turing-Maschinen
- 5. Entscheidbarkeit
- 6. Berechenbarkeit
- 7. Komplexität

- Gesucht: Maschinenmodell für kontextfreie Sprachen
- EA sind zu schwach
- Sprachen wie $a^n b^n$ benötigen unbeschränkte Speicherkomponente
- Kellerautomaten erweitern EA um Stack
 - Symbole werden „gestapelt“
 - LIFO: last in, first out
 - Höhe unbegrenzt
 - nur oberes Ende sichtbar
 - feststellbar: leer / nichtleer
 - nicht feststellbar: Anzahl und Inhalt der unteren Symbole
 - Operationen
 - push neues Element (mehrere) auf Stack legen
 - pop oberstes Element vom Stack nehmen
 - peek oberstes Element lesen, ohne es wegzunehmen
kann durch „pop; push“ simuliert werden





- EA plus unbegrenzter Stack:
 - Übergänge können Stack lesen und schreiben
 - nur oberes Ende
 - Stack-Alphabet Γ
- Übergänge:
 - gelesenes Eingabesymbol
 - gelesenes Stacksymbol
 - zu schreibende Stacksymbole
- Akzeptanzbedingung
 - leerer Stack nach Lesen der gesamten Eingabe
 - keine Endzustände nötig
- Gemeinsamkeiten mit EA:
 - Eingabe von links nach rechts lesen
 - Zustandsmenge, Startzustand
 - Eingabealphabet

$$\Delta \subseteq Q \times \Sigma \cup \{\varepsilon\} \times \Gamma \times \Gamma^* \times Q$$

- KA ist in Ausgangszustand
- kann nächstes Eingabesymbol lesen
- muss oberstes Stacksymbol lesen (und entfernen)
- kann beliebig viele Symbole auf Stack schreiben
- geht in Folgezustand

Definition 3.35 (Kellerautomat)

Ein **Kellerautomat** (KA) ist ein 6-Tupel $(Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$, wobei

- Q, Σ, q_0 wie bei NEA definiert sind,
- Γ das Stackalphabet ist,
- $Z_0 \in \Gamma$ das Stack-Startsymbol ist,
- $\Delta \subseteq Q \times \Sigma \cup \{\varepsilon\} \times \Gamma \times \Gamma^* \times Q$ die Übergangsrelation ist.

Eine **Konfiguration** eines KA ist ein Tripel (q, w, γ) , wobei

- $q \in Q$ der aktuelle Zustand ist,
- $w \in \Sigma^*$ die noch nicht gelesene Eingabe ist,
- $\gamma \in \Gamma^*$ der aktuelle Stack-Inhalt (von oben gelesen) ist.

Folgekonfiguration und **Lauf** sind wie für NEA definiert (mit Stack).

Ein Kellerautomat \mathcal{A} **akzeptiert** ein Wort $w \in \Sigma^*$, wenn es einen Lauf gibt mit

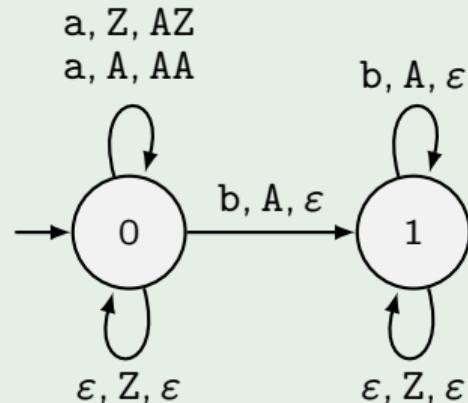
- Startkonfiguration (q_0, w, Z_0) und
- Endkonfiguration $(q, \varepsilon, \varepsilon)$ für ein beliebiges q .

Kellerautomat für $a^n b^n$

Beispiel 3.36 (Automat \mathcal{A}_{ab})

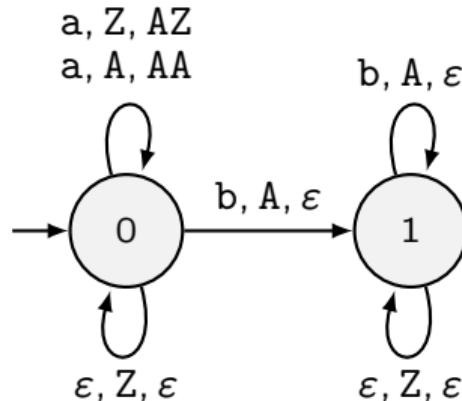
$$\mathcal{A} = (Q, \Sigma_{ab}, \Gamma, \Delta, 0, Z)$$

- $Q = \{0, 1\}$
- $\Gamma = \{A, Z\}$
- Δ : siehe Tabelle



Q	Σ	Γ	Γ^*	Q	Anmerkungen
$(0, \epsilon, Z, \epsilon, 0)$					akzeptiere leeres Wort
$(0, a, Z, AZ, 0)$	a	Z	AZ	0	Lese erstes a , push A
$(0, a, A, AA, 0)$	a	A	AA	0	Lese weitere a , push A
$(0, b, A, \epsilon, 1)$	b	A	ϵ	1	Lese erstes b , pop A
$(1, b, A, \epsilon, 1)$	b	A	ϵ	1	Lese weitere b , pop A
$(1, \epsilon, Z, \epsilon, 1)$					akzeptiere, wenn alle A gelöscht sind

Läufe von \mathcal{A}_{ab}



$$\Delta = \{(0, \quad \epsilon, \quad Z, \quad \epsilon, \quad 0), \\ (0, \quad a, \quad Z, \quad AZ, \quad 0), \\ (0, \quad a, \quad A, \quad AA, \quad 0), \\ (0, \quad b, \quad A, \quad \epsilon, \quad 1), \\ (1, \quad b, \quad A, \quad \epsilon, \quad 1), \\ (1, \quad \epsilon, \quad Z, \quad \epsilon, \quad 1)\}$$

Lauf auf aabb:

- 1 (0, aabb, Z)
- 2 (0, abb, AZ)
- 3 (0, bb, AAZ)
- 4 (1, b, AZ)
- 5 (1, ε, Z)
- 6 (1, ε, ε)

Lauf auf abb:

- 1 (0, abb, Z)
- 2 (0, bb, AZ)
- 3 (1, b, Z)
- 4 (1, b, ε)
- 5 kein Übergang möglich,
Eingabe nicht vollständig gelesen

- Γ und Σ müssen **nicht** disjunkt sein
 - $\Sigma \subseteq \Gamma$ erlaubt Einkellern von Eingabesymbolen
 - Konvention: $\Sigma = \{a, b, c, \dots\}$, $\Gamma = \{A, B, C, \dots\}$
- ε -Übergänge sind erlaubt
 - können Stack modifizieren
 - können nicht wie im NEA ersetzt werden
- KA in Definition 3.35 sind **nichtdeterministisch** (NKA)
- **Deterministischer KA:** In jeder Konfiguration höchstens eine Folgekonfiguration möglich
(Übergang entweder mit ε oder mit einem $c \in \Sigma$)
 - echt schwächer als NKA
 - beschreibt Klasse der **deterministisch kontextfreien** Sprachen
- Akzeptanz auch durch Endzustände möglich, aber...
 - Repräsentation umfangreicher (7-Tupel)
 - Beweise schwieriger

Übung 3.37

Die Sprache L der Palindrome (gerader Länge) über Σ_{ab} ist

$$L = \{w \cdot \overleftarrow{w} \mid w \in \Sigma^*\}$$

mit

$$\overleftarrow{w} = c_n \dots c_1 \text{ wenn } w = c_1 \dots c_n$$

- 1 Definieren Sie einen nichtdeterministischen Kellerautomaten, der L erkennt.
- 2 Finden Sie einen deterministischen Kellerautomaten?

Satz 3.38

Die Klasse der durch (nichtdeterministische) Kellerautomaten erkennbaren Sprachen ist genau die Klasse der Sprachen, die durch kontextfreie Grammatiken erzeugt werden können.

Beweis.

- 1 Erzeuge für eine kontextfreie Grammatik G einen KA \mathcal{A}_G mit $\mathcal{L}(\mathcal{A}_G) = \mathcal{L}(G)$.
- 2 Erzeuge für einen KA \mathcal{A} eine kontextfreie Grammatik $G_{\mathcal{A}}$ mit $\mathcal{L}(G_{\mathcal{A}}) = \mathcal{L}(\mathcal{A})$.



Definition 3.39 (Transformation einer KFG in einen Kellerautomaten)

Gegeben sei eine kontextfreie Grammatik $G = (N, \Sigma, P, S)$.

Der Kellerautomat \mathcal{A}_G ist definiert wie folgt:

$$\begin{aligned}\mathcal{A}_G &= (\{q\}, \Sigma, \Sigma \cup N, \Delta, q, S) \text{ mit} \\ \Delta &= \{(q, \varepsilon, A, \beta, q) \mid A \rightarrow \beta \in P\} \quad \cup \\ &\quad \{(q, c, c, \varepsilon, q) \mid c \in \Sigma\}\end{aligned}$$

\mathcal{A}_G simuliert Ableitungen von G :

- **NTS** A auf Stack: Regel wird auf A angewendet
- **TS** c auf Stack: wird gelöscht, wenn es dem nächsten Eingabesymbol entspricht

Satz 3.40

Es gilt: $\mathcal{L}(G) = \mathcal{L}(\mathcal{A}_G)$.

- \mathcal{A}_G hat nur einen Zustand.
- Korollar: KA brauchen keine Zustände.

Übung 3.41

Gegeben Sei die Grammatik $G = (\{S\}, \Sigma_{ab}, P, S)$ mit

$$\begin{aligned}P = \{S &\rightarrow aSa \\&S \rightarrow bSb \\&S \rightarrow \varepsilon\}\end{aligned}$$

- 1 Verwenden sie das gezeigte Verfahren, um einen äquivalenten KA \mathcal{A}_G zu erzeugen.
- 2 Geben Sie einen akzeptierenden Lauf von \mathcal{A}_G auf der Eingabe abba an.

Die Transformation eines Kellerautomaten $\mathcal{A} = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$ in eine kontextfreie Grammatik $G_{\mathcal{A}} = (N, \Sigma, P, S)$ ist aufwändiger:

1 Eliminierung der Zustandsmenge

- Transformiere KA \mathcal{A} in Automat \mathcal{A}' mit nur einem Zustand
- codiere Zustände in Stacksymbole
- exponentiell

2 Transformation eines KA mit einem Zustand in Grammatik

- ähnlich Transformation KFG \rightarrow KA
- linear

Eliminierung der Zustandsmenge

Gegeben KA $\mathcal{A} = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$

Gesucht KA $\mathcal{A}' = (\{q'\}, \Sigma, \Gamma', \Delta', q', Z')$ mit $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$

Idee Codiere Zustände von \mathcal{A} in Stacksymbole von \mathcal{A}'

- verwende Tripel $[pXq]$
- Intuition: \mathcal{A} geht von p zu q über und löscht dabei X

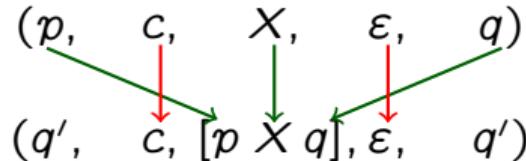
Beispiel 3.42 (Läufe in \mathcal{A} und \mathcal{A}')

Lauf in \mathcal{A}	$((0, ab, Z),$	$(0, b, AZ),$	$(1, \varepsilon, Z),$	$(1, \varepsilon, \varepsilon))$
Lauf in \mathcal{A}'	$((q', ab, Z'),$	$(q', ab, [0Z1])$	$(q', b, [0A1][1Z1]),$	$(q', \varepsilon, [1Z1]),$

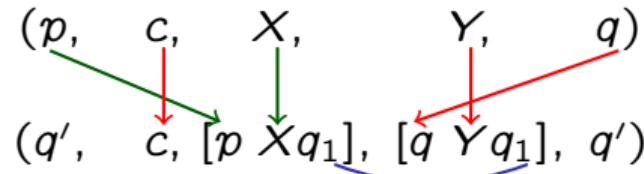
- Schwierigkeit: Zustände müssen **geraten** werden, sobald Stack-Symbol erzeugt wird
- Alle Abfolgen von Zuständen müssen möglich sein \rightsquigarrow exponentielle Konstruktion

Veranschaulichung der Transformation

Fall $n = 0$: 1 Übergang



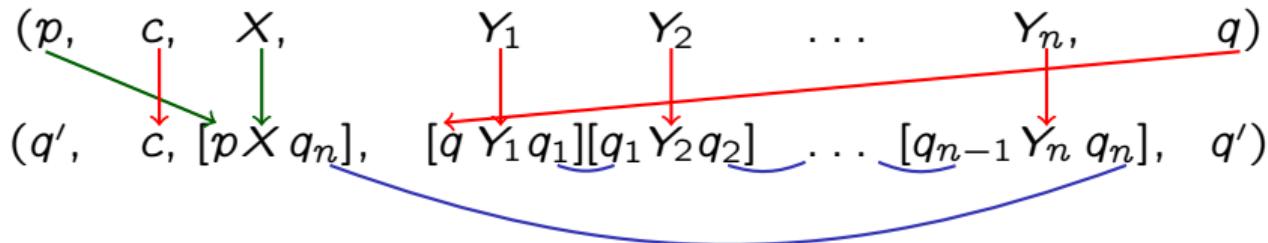
Fall $n = 1$: $|Q|$ Übergänge (alle $q_1 \in Q$)



„Um von p zu q überzugehen und dabei X zu löschen, lese ein c , gehe dann von q zu q_1 über und lösche dabei Y .“

„Um von p zu q_1 überzugehen und dabei X zu löschen, lese ein c , gehe dann von q zu q_1 über und lösche dabei Y .“

Allgemeiner Fall: $|Q|^n$ Übergänge (alle $(q_1, \dots, q_n) \in Q^n$)



„Um von p zu q_n überzugehen und dabei X zu löschen, lese ein c , finde dann von q aus Zustände q_1, \dots, q_{n-1} , über die man zu q_n übergehen und dabei Y_1, \dots, Y_n löschen kann.“

Definition 3.43 (Kellerautomat \mathcal{A}')

Gegeben sei der Kellerautomat $\mathcal{A} = (Q, \Sigma, \Gamma, \Delta, q_0, Z_0)$.

Der Automat $\mathcal{A}' = (\{q'\}, \Sigma, \Gamma', \Delta', q', Z')$ ist definiert wie folgt:

- $\Gamma' = \{Z'\} \cup \{[pXq] \mid \{p, q\} \subseteq Q \text{ und } X \in \Gamma\}$
- Für jeden Übergang $(p, c, X, Y_1Y_2 \dots Y_n, q) \in \Delta$ enthält Δ' alle Übergänge

$$(q', c, [pXq_n], [qY_1q_1][q_1Y_2q_2] \dots [q_{n-1}Y_nq_n], q')$$

für alle möglichen Kombinationen von Zuständen q_1, q_2, \dots, q_n .

- Zusätzlich enthält Δ' für jeden Zustand $q \in Q$ einen Übergang

$$(q', \varepsilon, Z', [q_0Z_0q], q')$$

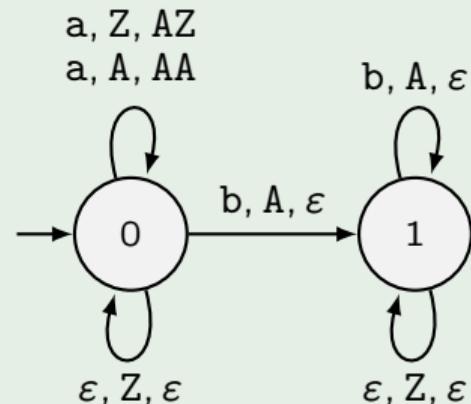
- \mathcal{A}' rät zuerst, in welchem Zustand der Stack geleert wird,
- anschließend werden auf dem Stack die Übergänge von \mathcal{A} mit Zuständen und Stacksymbolen simuliert.

Übung 3.44

Gegeben sei der Kellerautomat $\mathcal{A} = (Q, \Sigma_{ab}, \Gamma, \Delta, 0, Z)$ mit

- $Q = \{0, 1\}$
- $\Gamma = \{A, Z\}$

$$\blacksquare \quad \Delta = \left\{ \begin{array}{lllll} (0, & \varepsilon, & Z, & \varepsilon, & 0), \\ (0, & a, & Z, & AZ, & 0), \\ (0, & a, & A, & AA, & 0), \\ (0, & b, & A, & \varepsilon, & 1), \\ (1, & b, & A, & \varepsilon, & 1), \\ (1, & \varepsilon, & Z, & \varepsilon, & 1) \end{array} \right\}$$



- 1 Transformieren Sie \mathcal{A} mit dem gezeigten Verfahren in einen äquivalenten Automaten \mathcal{A}' mit nur einem Zustand.
- 2 Geben Sie akzeptierende Läufe von \mathcal{A}' auf den Wörtern ε , ab und $aabb$ an.

Sie können in der Definition und bei den Läufen von \mathcal{A}' den Zustand weglassen, also z.B.

- einen Übergang als $(a, [0Z1], [0A1][1Z1])$ und
- eine Konfiguration als $(aba, [0A1][1Z1])$ schreiben.

Definition 3.45 (Transformation eines Kellerautomaten in eine KFG)

Gegeben sei ein Kellerautomat $\mathcal{A} = (\{q\}, \Sigma, \Gamma, \Delta, q, Z_0)$.

Die kontextfreie Grammatik $G_{\mathcal{A}} = (N, \Sigma, P, S)$ ist definiert wie folgt:

- $N = \Gamma$;
 - $S = Z_0$;
 - $P = \{A \rightarrow cY_1Y_2 \dots Y_n \mid (q, c, A, Y_1Y_2 \dots Y_n, q) \in \Delta\}$.
-
- c kann auch ε sein
 - n kann auch 0 sein

Satz 3.46

Es gilt: $\mathcal{L}(\mathcal{A}) = \mathcal{L}(G_{\mathcal{A}})$.

Übung 3.47

Gegeben sei der Automat \mathcal{A}' aus Übung 3.44.

- 1 Transformieren Sie \mathcal{A}' in eine äquivalente kontextfreie Grammatik G .
- 2 Reduzieren Sie G .
- 3 Geben Sie in G Ableitungen für die Wörter ε , ab und aabb an.

Übung 3.48

- 1 a Erzeugen Sie einen KA \mathcal{A}_1 , der $L_1 = \{w \in \Sigma_{ab}^* \mid |w|_a = |w|_b\}$ erkennt.
b Geben Sie einen akzeptierenden Lauf von \mathcal{A}_1 auf abbbaa an.
- 2 a Erzeugen Sie einen KA \mathcal{A}_2 , der $L_2 = \{w \in \Sigma_{ab}^* \mid |w|_a < |w|_b\}$ erkennt.
b Geben Sie einen akzeptierenden Lauf von \mathcal{A}_2 auf bbbaaaabb an.
- 3 a Erzeugen Sie einen KA \mathcal{A}_3 , der $L_3 = \{w \in \Sigma_{abc}^* \mid |w| \text{ ist ungerade und } w[(|w| + 1)/2] = a\}$ erkennt (alle Wörter, deren mittlerer Buchstabe ein a ist).
b Geben Sie einen akzeptierenden Lauf von \mathcal{A}_3 auf cccaaabb an.
- 4 a Erzeugen Sie einen KA \mathcal{A}_4 , der $L_4 = \{a^n b^m c^o \mid n, m, o \in \mathbb{N}, n = m + o\}$ erkennt.
b Geben Sie einen akzeptierenden Lauf von \mathcal{A}_4 auf aabc an.

- **Stack:** unbeschränkte Speicherkomponente
- Übergänge können oberes Ende des Stacks lesen und schreiben
- ε -Übergänge möglich
- KA erkennt mit **leerem Stack**
- nur **nichtdeterministischer** KA ist äquivalent zu kontextfreien Grammatiken
 - KFG → KA Simulation von Regelanwendungen auf Stack
nur ein Zustand nötig
 - KA → KFG Exponentiell viele NTS, nicht praktikabel

- 1. Einführung
- 2. Reguläre Sprachen und endliche Automaten
- 3. Chomsky-Grammatiken und kontextfreie Sprachen**
 - 3.1 Chomsky-Grammatiken
 - 3.2 Die Chomsky-Hierarchie
 - 3.3 Rechtslineare Grammatiken
 - 3.4 Kontextfreie Grammatiken
 - 3.5 Entscheidungsprobleme
 - 3.6 Kellerautomaten
 - 3.7 Eigenschaften kontextfreier Sprachen**
- 4. Turing-Maschinen
- 5. Entscheidbarkeit
- 6. Berechenbarkeit
- 7. Komplexität

Wie kann man zeigen, dass eine Sprache nicht kontextfrei ist?

- Für reguläre Sprachen: Pumping-Lemma I
 - Endliche Automaten müssen auf langen Wörtern eine Schleife durchlaufen
 - Schleifendurchläufe können wiederholt werden
 - Wenn dabei Wörter entstehen, die nicht zur Sprache gehören, ist sie nicht regulär
- Für kontextfreie Sprachen: Pumping-Lemma II
 - Argument verwendet Grammatiken
 - Wenn G beliebig lange Wörter erzeugen kann, gibt es ein NTS, das sich selbst und einen nichtleeren Kontext erzeugt.
 - z.B. in Palindrom-Grammatik: $S \rightarrow aSa$
 - jede „echt“ kontextfreie Sprache ist unendlich und enthält beliebig lange Wörter
 - da G kontextfrei ist, kann dieser Teil wiederholt werden
 - das verlängerte Wort wird von G erzeugt

Satz 3.49 (Pumping-Lemma für kontextfreie Sprachen)

Sei L eine kontextfreie Sprache. Dann gibt es ein $k \in \mathbb{N}^{\geq 1}$, so dass gilt:

Für jedes $s \in L$ mit $|s| \geq k$ gibt es eine Zerlegung $s = u \cdot v \cdot w \cdot x \cdot y$ mit

- 1 $vx \neq \epsilon$
- 2 $|vwx| \leq k$
- 3 $u \cdot v^h \cdot w \cdot x^h \cdot y \in L$ für alle $h \in \mathbb{N}$.

Unterschiede zum PL I:

- Zwei Segmente werden parallel aufgepumpt
- k beschränkt nur Länge, nicht Position der aufgepumpten Teilwörter

Beispiel 3.50

Sei $G = (\{S\}, \Sigma_{ab}, \{S \rightarrow aSa|bSb|\varepsilon\}, S)$ die aus Übung 3.41 bekannte Palindrom-Grammatik.

Dann gilt:

Es gibt ein $k \in \mathbb{N}^{\geq 1}$,	$k = 6$
so dass für jedes $s \in L$ mit $ s \geq k$	$s = aabaabaa$
eine Zerlegung $s = uvwx y$ existiert	$s = \textcolor{blue}{a} \textcolor{red}{a} \textcolor{blue}{b} \textcolor{red}{a} \textcolor{blue}{b} \textcolor{red}{a} \textcolor{blue}{a}$
mit $vx \neq \varepsilon$	$\textcolor{red}{abba} \neq \varepsilon$
und $ vwx \leq k$	$ \textcolor{blue}{aba} \textcolor{red}{a} \leq 6$
und für alle h : $uv^h w x^h y \in L$	$\textcolor{red}{aabababa} \textcolor{blue}{a} \textcolor{red}{b} \textcolor{blue}{a} \textcolor{red}{b} \textcolor{blue}{a} \textcolor{red}{b} \textcolor{blue}{a} \textcolor{red}{b} \textcolor{blue}{a} \textcolor{red}{a}$

Beweis.

Sei $G = (N, \Sigma, P, S)$ eine KFG; o.B.d.A. enthalte P weder ε -Regeln noch Kettenregeln.

Sei $n = |N|$ und l die maximale Länge einer rechten Regelseiten in P .

Wir definieren $k := l^{n+1}$.

- Ein Baum der Tiefe $n + 1$ und Breite l hat maximal l^{n+1} Blätter.
- Wenn $|s| \geq k$ gilt, hat der Ableitungsbaum von s mindestens Tiefe $n + 1$, enthält also einen Pfad mit $n + 1$ Kanten und $n + 2$ Knoten.
- Von den Knoten sind $n + 1$ Nichtterminalsymbole, also kommt eins mehrfach vor:
 $A \Rightarrow^* vAx$
- Da P weder ε - noch Kettenregeln enthält, ist mindestens eins der Wörter v, x nicht leer.
- Die maximale Länge des aus vAx entstehenden Teilworts vwx ist gleich $l^{n+1} = k$.
- Da G kontextfrei ist, kann auf das erzeugte A dieselbe Ableitung erneut angewendet werden:
 $A \Rightarrow^* vAx \Rightarrow^* vvAxx \Rightarrow^* vvvAxxx \Rightarrow^* \dots$



Nachweis, dass L nicht kontextfrei ist:

- Für jedes $k \in \mathbb{N}^{\geq 1}$
- existiert ein $s \in L$ mit $|s| \geq k$
- so dass für jede Zerlegung $s = uvwxy$ mit $vx \neq \varepsilon$ und $|vwx| \leq k$
- ein $h \in \mathbb{N}$ existiert mit $uv^hwx^hy \notin L$.

Spieltheoretisch:

- Gegner wählt k
- Wähle s mit $|s| \geq k$
- Gegner wählt Zerlegung $s = uvwxy$ mit $vx \neq \varepsilon$ und $|vwx| \leq k$
- Wähle $h \in \mathbb{N}$
- Zeige $uv^hwx^hy \notin L$

Beispiel 3.51

Sei $L_D = \{ww \mid w \in \Sigma_{ab}^*\}$.

Zeige, dass L_D nicht kontextfrei ist, mit Kontraposition des PL II.

1. Versuch

- Sei $k \in \mathbb{N}^{\geq 1}$.
- Wähle $s = a^l b^l a^l b^l$ mit $l = \left\lceil \frac{k}{4} \right\rceil$.

- Z.B. mit Zerlegung $\varepsilon a^l b^l a^l b^l$:
- gehört jedes aufgepumpte Wort zu L_D :
 $\varepsilon a^l a^l a^l b^l a^l a^l a^l b^l$ (wie bei Palindromen)

2. Versuch: nutze $|vwx| \leq k$

- Sei $k \in \mathbb{N}^{\geq 1}$.
- Wähle $s = a^k b^k a^k b^k$.
- Dann gilt für jede Zerlegung $s = uvwxy$ eine der Aussagen:
 - vwx liegt in **erster Worthälfte**: Wähle $h = 2$. Dann gilt $uv^2wx^2y \notin L_D$, denn die erste Worthälfte beginnt mit a, die zweite mit b.
 - vwx liegt in **zweiter Worthälfte**: Wähle $h = 2$. Dann gilt $uv^2wx^2y \notin L_D$, denn die erste Worthälfte endet mit a, die zweite mit b.
 - vwx umfasst Mitte von s: Wähle $h = 0$. Dann gilt wegen $|vwx| \leq k$: $uwy = a^k b^{k-i} a^{k-j} b^k$, wobei mindestens eins von i, j größer als 0 ist.
Daraus folgt $uwy \notin L_D$.

Übung 3.52

Zeigen Sie unter Verwendung des Pumping-Lemma II, dass die Sprache

$$L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$$

nicht kontextfrei ist.

Satz 3.53 (Abschluss unter $\cup, \cdot, ^*$)

Die Klasse der kontextfreien Sprachen ist abgeschlossen unter Vereinigung, Konkatenation und Kleene-Stern.

Beweis.

Seien $G_1 = (N_1, \Sigma, P_1, S_1)$ und $G_2 = (N_2, \Sigma, P_2, S_2)$ kontextfreie Grammatiken mit $N_1 \cap N_2 = \{\}$, S ein neues Startsymbol.

Die folgenden zusätzlichen Regeln liefern Grammatiken für reguläre Operationen:

$S \rightarrow S_1, S \rightarrow S_2$ liefert G_{\cup} mit $\mathcal{L}(G_{\cup}) = \mathcal{L}(G_1) \cup \mathcal{L}(G_2)$

$S \rightarrow S_1 S_2$ liefert G_{\cdot} mit $\mathcal{L}(G_{\cdot}) = \mathcal{L}(G_1) \cdot \mathcal{L}(G_2)$

$S \rightarrow \epsilon, S \rightarrow S_1 S$ liefert G_{*} mit $\mathcal{L}(G_{*}) = \mathcal{L}(G_1)^*$



Satz 3.54 (Abschluss unter \cap)

Die Klasse der kontextfreien Sprachen ist **nicht** abgeschlossen unter Durchschnitt.

Beweis.

Angenommen, die Klasse der kontextfreien Sprachen wäre abgeschlossen unter Durchschnitt.
Dann wäre $L_{abc} = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ kontextfrei:

- $\{a^n b^n c^m \mid \{m, n\} \subset \mathbb{N}\}$ ist kontextfrei (Konkatenation von $\{a^n b^n\}$ und $\{c^m\}$)
- $\{a^m b^n c^n \mid \{m, n\} \subset \mathbb{N}\}$ ist kontextfrei (Konkatenation von $\{a^m\}$ und $\{b^n c^n\}$)
- $\{a^n b^n c^n \mid n \in \mathbb{N}\} = \{a^n b^n c^m\} \cap \{a^m b^n c^n\}$

Mit dem Pumping-Lemma wurde gezeigt, dass L_{abc} nicht kontextfrei ist. \square

Übung 3.55

Definieren Sie kontextfreie Grammatiken für $L_1 = \{a^n b^n c^m \mid \{n, m\} \subset \mathbb{N}\}$ und $L_2 = \{a^m b^n c^n \mid \{n, m\} \subset \mathbb{N}\}$.

Beispiel 3.56

L_D aus Beispiel 3.51 ist die Sprache der doppelten Wörter über Σ_{ab} , d.h.

- für alle $w \in L_D$ gilt: $l = |w|$ ist gerade und
- für alle i mit $1 \leq i \leq \frac{l}{2}$ gilt: $w[i] = w[i + \frac{l}{2}]$
- z.B.: abaababaab

Ihr Komplement $\overline{L_D}$ besteht aus den folgenden Wörtern:

- Wörter mit ungerader Länge,
- Wörter mit gerader Länge, für die gilt:
 - es gibt ein i mit $1 \leq i \leq \frac{l}{2}$ und $w[i] \neq w[i + \frac{l}{2}]$
 - z.B.: abaaabba
 - anders gesagt: Ein a, ein b, dazwischen $\frac{l}{2} - 1$ Zeichen
 - anders gesagt: Gleich viele Zeichen zwischen a und b wie außerhalb von a und b
 - formal: $\{\{a, b\}^m x \{a, b\}^{m+n} y \{a, b\}^n \mid \{x, y\} = \{a, b\} \text{ und } \{m, n\} \subset \mathbb{N}\}$

Kontextfreie Grammatik für $\overline{L_D}$: $G = (\{S, U, E, A, B, C\}, \Sigma_{ab}, P, S)$ mit

$$\begin{array}{llll} P = \{ & S \rightarrow E|U, & U \rightarrow CUC|C, & C \rightarrow a|b, \\ & E \rightarrow AB|BA, & A \rightarrow CAC|a, & B \rightarrow CBC|b \} \end{array}$$

Satz 3.57

Die Klasse der kontextfreien Sprachen ist nicht abgeschlossen unter Komplement.

Beweis.

Angenommen, die Klasse der kontextfreien Sprachen wäre abgeschlossen unter Komplement.
Die Sprache $\overline{L_D}$ ist kontextfrei (Beispiel 3.56).

Ihr Komplement ist $\overline{\overline{L_D}} = L_D$.

L_D ist aber nicht kontextfrei (Beispiel 3.51). \ntriangleq



Warum ist $\overline{L_D}$ einfacher als L_D ?

- Nur für ein Zeichenpaar muss Ungleichheit getestet werden.
- Bei restlichen Paaren ist nur Anzahl relevant, nicht (Un-)Gleichheit.

Übung 3.58

Führen Sie einen alternativen indirekten Beweis für Satz 3.57.

Verwenden Sie hierzu die Sprachen L_1 und L_2 aus Übung 3.55 sowie die bekannten Abschlusseigenschaften, um die Annahme, die Klasse der kontextfreien Sprachen sei abgeschlossen unter Komplement, zum Widerspruch zu führen.

- Pumping-Lemma II
 - Gleches Grundprinzip
 - Wort wird in 5 Segmente zerlegt, 2 werden parallel aufgepumpt
- Abschlusseigenschaften
 - abgeschlossen unter $\cup, ^*, \cdot$
 - nicht abgeschlossen unter $\cap, -$

- Chomsky-Hierarchie
 - Unterscheidung Sprachen und Grammatiken
 - Kontextfreie Grammatiken sind keine Unterklasse der monotonen Grammatiken
- Rechtslineare Grammatiken erzeugen reguläre Sprachen
- Kontextfreie Sprachen
 - können verschachtelte Strukturen beschreiben
 - sind relevant für die Praxis (Programmiersprachen, NLP)
- Kontextfreie Grammatiken
 - Chomsky-Normalform \rightsquigarrow Leerheitsproblem
 - CYK-Algorithmus \rightsquigarrow Wortproblem
- Kellerautomaten
 - erkennen kontextfreie Sprachen
 - nur nicht-deterministische Variante äquivalent zu KFG
- Schwieriger zu handhaben als reguläre Sprachen
 - nicht abgeschlossen unter \cap , $\bar{\cdot}$
 - Äquivalenzproblem unentscheidbar

Inhalt

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
- 4. Turing-Maschinen**
 - 4.1 Motivation
 - 4.2 Aufbau und Funktionsweise
 - 4.3 Mehrband-Turingmaschinen
 - 4.4 Unbeschränkte Grammatiken
 - 4.5 Linear beschränkte Automaten
5. Entscheidbarkeit
6. Berechenbarkeit
7. Komplexität

Inhalt

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
- 4. Turing-Maschinen**
 - 4.1 Motivation
 - 4.2 Aufbau und Funktionsweise
 - 4.3 Mehrband-Turingmaschinen
 - 4.4 Unbeschränkte Grammatiken
 - 4.5 Linear beschränkte Automaten
5. Entscheidbarkeit
6. Berechenbarkeit
7. Komplexität

Formale Sprachen

- Maschinenmodelle für Sprachen von Typ 1 und 0
- Abschlusseigenschaften
- Entscheidungsprobleme

Algorithmik

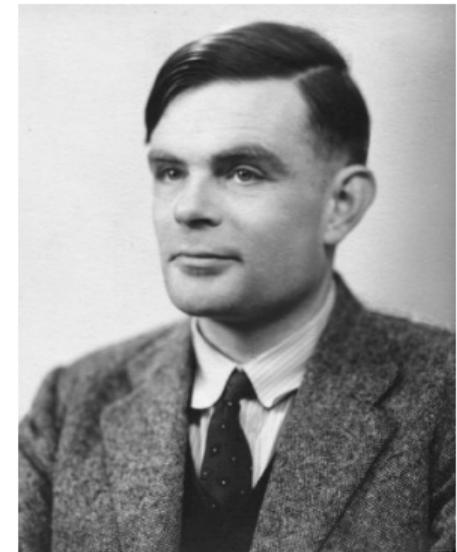
- **Präzise** Definition von „Algorithmus“
- Entscheidbarkeit und Berechenbarkeit
- **Grenzen** von Algorithmen / Computern

Vier Sprachklassen definiert durch Grammatiken und Maschinenmodelle:

Typ	Sprachklasse	Grammatik	Maschinenmodell
0	rekursiv aufzählbar	unbeschränkt	?
1	kontextsensitiv	monoton	?
2	kontextfrei	kontextfrei	Kellerautomat
3	regulär	rechtslinear	endlicher Automat

Nötig ist mächtigeres Modell als Kellerautomat

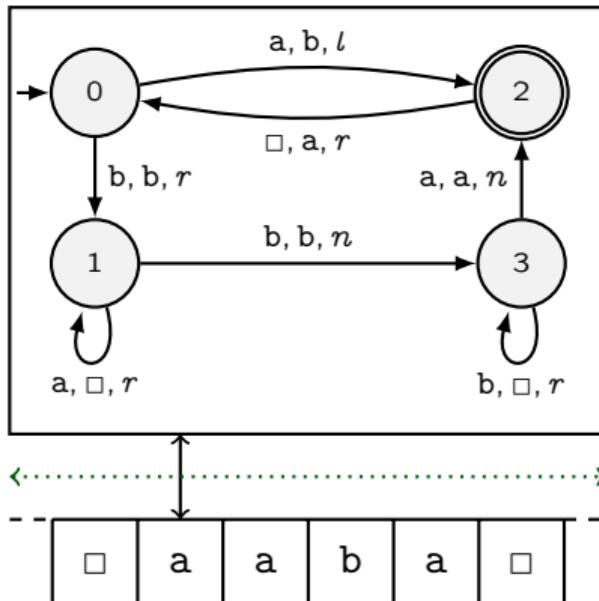
- 1936 von Alan Turing vorgestellt
 - *On computable numbers, with an application to the Entscheidungsproblem*
 - verwendet für Beweis der **Unentscheidbarkeit** des Gültigkeitsproblems der Prädikatenlogik
- Modell eines **universellen Computers**
 - sehr einfach \leadsto geeignet für Beweise
 - Unendliches Band, ein Schreib-/Lesekopf
 - Band enthält Symbole
 - Endlicher Automat kontrolliert Schreib-/Leseoperationen und Bewegungen des Kopfes
 - sehr mächtig
 - Kann alles berechnen, was ein realer Computer berechnen kann
 - Kann alles berechnen, was ein idealer Computer berechnen kann
 - Kann alles berechnen, was ein Mensch berechnen kann (?)



Alan M. Turing
(1912–1954)

Inhalt

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
- 4. Turing-Maschinen**
 - 4.1 Motivation
 - 4.2 Aufbau und Funktionsweise**
 - 4.3 Mehrband-Turingmaschinen
 - 4.4 Unbeschränkte Grammatiken
 - 4.5 Linear beschränkte Automaten
5. Entscheidbarkeit
6. Berechenbarkeit
7. Komplexität



- Medium: unendliches Band (bidirektional)
 - enthält anfangs Eingabe (und Blanks □)
 - dient auch für Ausgabe und Zwischenergebnisse
 - kann gelesen und beschrieben werden
 - Lesekopf kann beliebig bewegt werden
- Übergänge
 - Lesen auf aktueller Bandposition
 - Schreiben auf aktueller Bandposition
 - Bewegen des Schreib-/Lesekopfs (*l, r, n*)
- Akzeptanzbedingung
 - TM ist in Endzustand
 - keine weitere Bewegung möglich
- Gemeinsamkeiten mit EA
 - Kontrolleinheit (endliche Zustandsmenge),
 - Start- und Endzustände
 - Eingabealphabet

$$\Delta \subseteq Q \times \Gamma \times \Gamma \times \{l, r, n\} \times Q$$

- TM ist in Ausgangszustand
- liest Bandsymbol an aktueller Position
- schreibt Bandsymbol auf aktuelle Position
- bewegt Kopf nach links, rechts oder gar nicht
- geht in Folgezustand

Definition 4.1 (Turing-Maschine)

Eine Turing-Maschine (TM) ist ein 6-Tupel $(Q, \Sigma, \Gamma, \Delta, q_0, F)$, wobei

- Q, Σ, q_0, F definiert sind wie für endliche Automaten;
- $\Gamma \supseteq \Sigma \cup \{\square\}$ das Band-Alphabet ist und zumindest Σ und das Blank-Symbol enthält;
- $\Delta \subseteq Q \times \Gamma \times \Gamma \times \{l, n, r\} \times Q$ die Übergangsrelation ist.

Wenn Δ höchstens einen Übergang (p, c, c', d, q) für jedes Paar $(p, c) \in Q \times \Sigma$ enthält, heißt die TM deterministisch (DTM). Die Übergangs-Funktion wird dann mit δ bezeichnet.

- Γ kann weitere Zeichen enthalten.
 - Das macht die TM nicht mächtiger, aber leichter handhabbar.
- Existenz eines Übergangs ist nicht gefordert
 - sonst gäbe es keine Stopkonfigurationen

Definition 4.2 (Konfiguration)

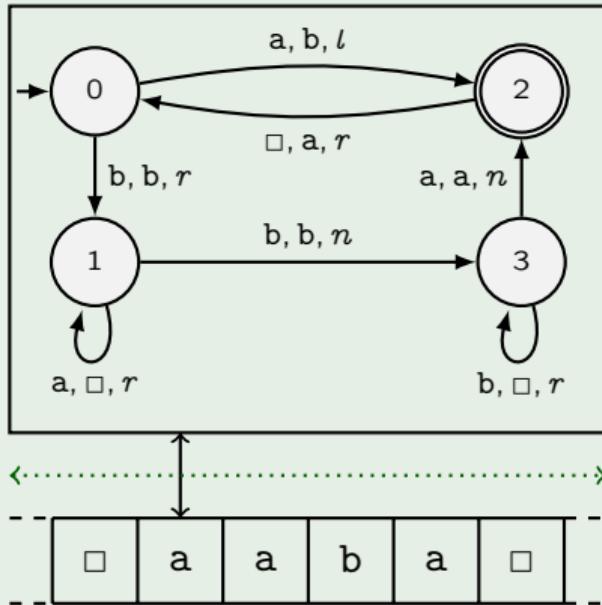
Eine **Konfiguration** $c = xqy$ einer Turing-Maschine $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_0, F)$ besteht aus

- dem aktuellen Zustand $q \in Q$;
- dem Band-Inhalt $x \in \Gamma^*$ links von der aktuellen Kopf-Position
(ohne unendliche \square -Folgen)
- dem Bandinhalt $y \in \Gamma^*$ beginnend mit der aktuellen Kopf-Position
(ohne unendliche \square -Folgen)

Eine Konfiguration $c = xqy$ heißt **akzeptierend**, wenn $q \in F$ gilt.

Eine Konfiguration c heißt **Stop-Konfiguration**, wenn es keine Übergänge von c aus gibt.

Beispiel 4.3



Konfigurationen:

- 1 0aaba
- 2 2 \square baba (akzeptierend)
- 3 a0baba
- 4 ab1aba
- 5 ab \square 1ba
- 6 ab \square 3ba
- 7 ab \square \square 3a
- 8 ab \square \square 2a (akzeptierend, Stop)

Definition 4.4 (Folge-Konfiguration)

Sei

- $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_0, F)$ eine TM und
- $c = vdpaw$ eine Konfiguration von \mathcal{M} mit
- $p \in Q, \{a, b, d\} \subseteq \Gamma, \{v, w\} \subseteq \Gamma^*$.

Die Konfiguration c' heißt **Folgekonfiguration** von c ($c \vdash c'$), wenn

- $c' = vqdbw$ gilt für einen Übergang $(p, a, b, l, q) \in \Delta$,
- $c' = vdqbw$ gilt für einen Übergang $(p, a, b, n, q) \in \Delta$, oder
- $c' = vdbqw$ gilt für einen Übergang $(p, a, b, r, q) \in \Delta$.

Für eine beliebig lange Folge von Übergängen schreibt man auch $c \vdash^* c'$.

Definition 4.5 (Berechnung, Akzeptanz, erkannte Sprache)

Eine **Berechnung** einer TM $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_0, F)$ auf einem Wort w ist eine Folge von Konfigurationen von \mathcal{M} , die mit q_0w beginnt und für die gilt, dass jede weitere Konfiguration eine Folgekonfiguration der vorherigen ist.

\mathcal{M} **akzeptiert** w , wenn es eine Berechnung von \mathcal{M} auf w gibt, die mit einer akzeptierenden Stopkonfiguration c_a endet, d.h. $q_0w \vdash^* c_a$.

\mathcal{M} **verwirft** w , wenn jede Berechnung von \mathcal{M} auf w in einer nicht akzeptierenden Stopkonfiguration endet.

Die von \mathcal{M} **erkannte Sprache** ist die Menge der von \mathcal{M} akzeptierten Wörter.

Übung 4.6

Sei $L = \{w \in \Sigma_{ab}^* \mid w[1] = w[|w|]\}$, also die Sprache aller Wörter, bei denen der erste gleich dem letzten Buchstaben ist.

- 1 Geben Sie eine TM \mathcal{M} an, die L erkennt.
- 2 Geben Sie Berechnungen von \mathcal{M} auf den Wörtern b , abb und $abaa$ an.

Sie können Ihre Lösung auf <https://turingmachinesimulator.com/> testen.

Beispiel 4.7 (Turing-Maschine für $\{a^n b^n c^n \mid n \in \mathbb{N}\}$)

$\mathcal{M}_{abc} = (Q, \Sigma_{abc}, \Gamma, \Delta, \text{start}, \{\text{accept}\})$ mit

- $Q = \{\text{start}, \text{findb}, \text{findc}, \text{check}, \text{back}, \text{test}, \text{accept}\}$
- $\Gamma = \Sigma_{abc} \cup \{\square, x, y, z\}$

Δ				
start	\square	\square	n	accept
start	a	x	r	findb
findb	a	a	r	findb
findb	y	y	r	findb
findb	b	y	r	findc
findc	b	b	r	findc
findc	z	z	r	findc
findc	c	z	r	check
check	c	c	l	back
check	\square	\square	l	test

Δ				
back	z	z	l	back
back	b	b	l	back
back	y	y	l	back
back	a	a	l	back
back	x	x	r	start
test	z	z	l	test
test	y	y	l	test
test	x	x	l	test
test	\square	\square	r	accept

Übung 4.8

- 1 Geben Sie Berechnungen von \mathcal{M}_{abc} auf den Wörtern aabbcc und aabc an.
- 2 Wie viele Schritte braucht \mathcal{M}_{abc} in Abhängigkeit von der Länge der Eingabe (\mathcal{O} -Notation)?

Übung 4.9

- 1 Entwickeln Sie eine Turing-Maschine \mathcal{M}_D , die die Sprache $L_D = \{ww \mid w \in \Sigma_{ab}^*\}$ erkennt (Beispiel 3.51).
- 2 Wie viele Schritte braucht Ihre TM in Abhängigkeit von der Länge der Eingabe?
- 3 Gibt es eine deterministische Turing-Maschine, die L_D erkennt?

- Unendliches Band
 - schreiben möglich (schlägt endliche Automaten)
 - beliebige Bewegungen (schlägt Kellerautomaten)
- Bandalphabet mit Blank
- Akzeptanz: Stopkonfiguration mit Endzustand

Inhalt

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
- 4. Turing-Maschinen**
 - 4.1 Motivation
 - 4.2 Aufbau und Funktionsweise
 - 4.3 Mehrband-Turingmaschinen**
 - 4.4 Unbeschränkte Grammatiken
 - 4.5 Linear beschränkte Automaten
5. Entscheidbarkeit
6. Berechenbarkeit
7. Komplexität

Vorteil TM gegenüber Kellerautomat: Unbegrenztes Band

- beliebig lange Zwischenergebnisse
- Zeichen können beliebig oft gelesen werden.

Praktischer Nachteil: Eingabe und Zwischenergebnisse hintereinander auf demselben Band

- Umständliches Hin-und-her-Spulen des Bandes
- Verlängern eines Teils erfordert Verschieben des kompletten Restinhalts

Abhilfe: TM mit **mehreren Bändern**

- z.B. ein Band Ein-/Ausgabe, ein Band Zwischenergebnisse
- Köpfe können **unabhängig** über Bänder bewegt werden

Definition 4.10 (k -Band-Turing-Maschine)

Eine k -Band-Turingmaschine ist ein 6-Tupel $(Q, \Sigma, \Gamma, \Delta, q_0, F)$, wobei

- Q, Σ, Γ, q_0 und F definiert sind wie für (Einband-)Turingmaschinen und
- $\Delta \subseteq Q \times \Gamma^k \times \Gamma^k \times \{r, l, n\}^k \times Q$ gilt.

Beispiel 4.11

Der Übergang $(p, \begin{pmatrix} a \\ a \\ c \end{pmatrix}, \begin{pmatrix} b \\ c \\ c \end{pmatrix}, \begin{pmatrix} l \\ r \\ n \end{pmatrix}, q)$ besagt:

- Wenn die TM im Zustand p ist
- und auf den Bändern 1–3 die Symbole a, a und c stehen,
- werden diese mit b, c bzw. c überschrieben,
- die 3 Köpfe bewegen sich nach links, rechts bzw. gar nicht,
- anschließend geht die Maschine in den Zustand q über.

Definition 4.12 (Konfiguration)

Eine **Konfiguration** $c = xqy$ einer Mehrband-TM ist definiert wie die einer Einband-TM, mit dem Unterschied, dass x und y Elemente von $(\Gamma^k)^*$ sind.

Berechnung und Akzeptanz sind definiert wie für Einband-Turingmaschinen.

Beispiel 4.13 (Konfiguration 3-Band-TM)

$$\begin{pmatrix} a \\ \square \\ c \end{pmatrix} \begin{pmatrix} b \\ \square \\ \square \end{pmatrix} \begin{pmatrix} c \\ b \\ \square \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} 5 \begin{pmatrix} c \\ \square \\ a \end{pmatrix} \begin{pmatrix} \square \\ c \\ \square \end{pmatrix}$$

Beispiel 4.14

$\mathcal{M}_2 = (Q, \Sigma_{abc}, \Gamma, \Delta, \text{start}, \{\text{accept}\})$ mit

- $Q = \{\text{start}, \text{reada}, \text{readb}, \text{readc}, \text{accept}\}$
- $\Gamma = \Sigma_{abc} \cup \{\square\}$
- Δ siehe Tabelle

Vorteile:

- nur $\mathcal{O}(|w|)$ Schritte
- einfachere Übergangsrelation
- keine zusätzlichen Band-Symbole
- kein Abschluss-Check

start				accept
start				reada
reada				reada
reada				readb
readb				readb
readb				readc
readc				readc
readc				accept

Übung 4.15

Geben Sie die Berechnungen von \mathcal{M}_2 auf den Wörtern aabbcc und aabc an.

Übung 4.16

Geben Sie eine 2-Band-TM für die Sprache $\{ww \mid w \in \Sigma_{ab}^*\}$ an, die für eine Eingabe der Länge n nur $\mathcal{O}(n)$ Berechnungsschritte benötigt.

Simulation von k -Band-TM \mathcal{M}_k durch 1-Band-TM \mathcal{M}_1

- statt k Bänder: \mathcal{M}_1 hat ein Band mit k Spuren \rightsquigarrow Alphabet Γ^k
- Kopfpositionen: weitere k Spuren: X markiert Kopfposition, alle anderen Felder: \square
- Berechnungsschritte:
 - Laufe über verwendetes Band (Anfangs-/ End-Markierungen)
 - merke in Zuständen den Inhalt von Spur $2 \cdot i$, wenn auf Spur $2 \cdot i - 1$ ein X steht
 - finde passenden Übergang in Δ
 - modifiziere Spursymbole und Kopfpositionen

Anmerkung

- explizite Darstellung von \mathcal{M}_1 sehr umfangreich
 - umständlich aufzuschreiben
 - schwer zu verstehen
- für komplexe Konstruktionen: Informelle Beschreibung einer TM
 - einzelne Schritte offensichtlich mit TM machbar

Satz 4.17

Jede Sprache, die von einer k -Band-Turingmaschine erkannt wird, wird auch von einer (1-Band-)Turing-Maschine erkannt.

Die Maschinenmodelle „Turing-Maschine“ und „ k -Band-Turing-Maschine“ sind äquivalent.

Wichtig

- Zustandsmenge wird sehr groß
 - $|\Gamma|^k$ Zustände zum Merken des Bandinhalts
- viel mehr Berechnungsschritte nötig
- funktioniert nur für festes k
 - Merken in Zuständen sonst nicht möglich

NTM \mathcal{M} akzeptiert w , wenn eine Berechnung existiert, die zu einer akzeptierenden Stopkonfiguration führt.

Satz 4.18 (Äquivalenz von DTM und NTM)

Jede NTM $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_0, F)$ kann durch eine DTM \mathcal{M}_{det} simuliert werden.
DTMs und NTMs beschreiben dieselbe Sprachklasse.

Beweis.

Verwende 2-Band-DTM für \mathcal{M}_{det} :

- Band 1: Aufzählung aller möglichen Konfigurationen von \mathcal{M} auf w
 - funktioniert als Warteschlange (queue)
 - beginnend mit $q_0 w$
 - zusätzliche Bandsymbole: q_i Zustände, „*“ als Trennsymbol, „!“ für aktuelle Konfiguration c_a
- Band 2: Zwischenspeicher für Konfigurationen
- Ablauf:
 - 1 wenn c_a akzeptierende Stopkonfiguration ist, gehe in akzeptierende Stopkonfiguration
 - 2 sonst kopiere c_a auf B2; gehe ans Ende von B1
 - 3 für k mögliche Übergänge von c_a : erzeuge k Kopien von c_a auf B1 (getrennt durch „*“)
 - 4 modifiziere Kopien auf B1 entsprechend Δ
 - 5 bewege Marker „!“ eine Konfiguration nach rechts
 - 6 weiter bei 1

□

Weitere äquivalente Varianten von Turingmaschinen

- einseitig unbeschränktes Band
- Bewegung nur r oder l (nicht n)

- k Bänder
- k Köpfe können sich unabhängig bewegen
- Übergänge und Konfigurationen: k -Tupel
- Beschreiben dieselbe Sprachklasse wie 1-Band-Turingmaschinen
 - ... ermöglichen aber einfachere Konstruktionen
- **Nichtdeterministische** Turingmaschinen
 - können durch eine deterministische 2-Band-TM simuliert werden
 - beschreiben dieselbe Sprachklasse wie 1-Band-Turingmaschinen

Inhalt

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
- 4. Turing-Maschinen**
 - 4.1 Motivation
 - 4.2 Aufbau und Funktionsweise
 - 4.3 Mehrband-Turingmaschinen
 - 4.4 Unbeschränkte Grammatiken**
 - 4.5 Linear beschränkte Automaten
5. Entscheidbarkeit
6. Berechenbarkeit
7. Komplexität

Satz 4.19 (Äquivalenz von Turing-Maschinen und Typ-0-Grammatiken)

Die Klasse der von Turing-Maschinen erkennbaren Sprachen ist identisch mit der Klasse der Sprachen, die durch unbeschränkte Grammatiken erzeugt werden

Beweis.

- 1 Simuliere Ableitung einer Typ-0-Grammatik mit Turingmaschine.
- 2 Simuliere Berechnung einer Turingmaschine mit Typ-0-Grammatik.



Gegeben: Grammatik $G = (N, \Sigma, P, S)$

Verwende nicht-deterministische 2-Band-TM:

- Band 1 speichert Eingabewort w
- Band 2 simuliert von S ausgehende Ableitungen gemäß P
- Ablauf:
 - 1 wähle (nicht-deterministisch) Position p auf Band 2
 - 2 wenn das auf p beginnende Wort zu einer Regel $\alpha \rightarrow \beta$ passt
 - verschiebe Bandinhalt wenn nötig
 - ersetze α durch β
 - 3 vergleiche Band 2 mit Band 1
 - wenn gleicher Inhalt, gehe in akzeptierende Stopkonfiguration
 - sonst weiter bei 1

Ziel: Transformation von TM \mathcal{M} in Grammatik $G_{\mathcal{M}}$

Technische Schwierigkeit:

- Berechnung von \mathcal{M} :
 - 1 \mathcal{M} erhält Eingabewort w am Anfang;
 - 2 \mathcal{M} kann w lesen und verändern;
 - 3 \mathcal{M} akzeptiert ursprüngliche Eingabe w oder nicht (Stopkonfiguration oder Endlosschleife).
- Ableitung von $G_{\mathcal{M}}$:
 - 1 $G_{\mathcal{M}}$ beginnt mit S ;
 - 2 $G_{\mathcal{M}}$ wendet Regeln auf Teilwörter an;
 - 3 $G_{\mathcal{M}}$ erzeugt möglicherweise Terminalwort w am Ende.
- Zur Simulation der Berechnung von \mathcal{M} muss $G_{\mathcal{M}}$
 - 1 ein Terminalwort w erzeugen;
 - 2 die Berechnungsschritte von \mathcal{M} simulieren;
 - 3 w wiederherstellen, falls die Berechnung akzeptierend ist.

Sei $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_0, F)$.

$G_{\mathcal{M}} = (N, \Sigma, P, S)$; $N = \{S, \square\} \cup Q$; P hat drei Gruppen von Regeln:

- 1** Erzeugen eines beliebigen $w \in \Sigma^*$ mit Blanks und Startzustand
 \rightsquigarrow Startkonfiguration $\square \dots \square q_0 \text{input} \square \dots \square$

- 2** Simulation der Berechnung von \mathcal{M} auf w

$$(p, a, b, n, q) \rightsquigarrow pa \rightarrow qb$$

$$(p, a, b, r, q) \rightsquigarrow pa \rightarrow bq$$

$$(p, a, b, l, q) \rightsquigarrow dpa \rightarrow qdb \text{ (für alle } d \in \Gamma\text{)}$$

Übung 4.20

Gegeben sei die TM $\mathcal{M} = (\{0, 1, 2, 3, 4\}, \{a\}, \{a, \square\}, \Delta, 0, \{4\})$ mit Δ in der folgenden Tabelle:

0	a	\square	r	1
0	\square	\square	n	4
1	a	a	r	1
1	\square	\square	l	2
2	a	\square	l	3
3	a	a	l	3
3	\square	\square	r	0

- 1 Geben Sie Regeln für die Gruppen 1 und 2 des beschriebenen Verfahrens an, um eine Grammatik $G_{\mathcal{M}}$ zu erzeugen, die die Turing-Maschine \mathcal{M} simuliert.
- 2 Geben Sie eine Ableitung von $G_{\mathcal{M}}$ an, die die Berechnung von \mathcal{M} auf dem Wort aaaa simuliert.
- 3 Welche Sprache erkennt \mathcal{M} ?

- 3 Wiederherstellung von w nach Erreichen einer akzeptierenden Stopkonfiguration
 - benötigt Nichtterminal-Alphabet $N \cup \Sigma \times N \cup \Sigma$ mit „Backup-Spur“
 - Startkonfiguration: $\dots (\square) (q_0) (i) (n) (p) (u) (t) (\square) \dots$
 - Simulation der Berechnung nur in erster Komponente
 $\rightsquigarrow \dots (a) (i) (d) (\square) (n) (p) (u) (t) (\square) \dots$
 - Wiederherstellung von w aus zweiter Komponente: $\rightsquigarrow \text{input}$

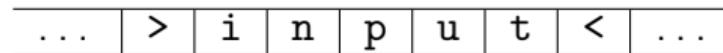
Typ	Sprachklasse	Grammatik	Maschinenmodell
0	rekursiv aufzählbar	unbeschränkt	Turing-Maschine
1	kontextsensitiv	monoton	?
2	kontextfrei	kontextfrei	Kellerautomat
3	regulär	rechtslinear	endlicher Automat

- Konstruktion Grammatik \rightsquigarrow TM: verwendet nichtdeterministische 2-Band-TM
 - ... wäre aber auch mit deterministischer 1-Band-TM möglich.
- Konstruktion TM \rightsquigarrow Grammatik: ähnlich komplex wie bei Typ-2-Grammatiken
 - Paare von Bandsymbolen als Nichtterminal-Symbole
 - Backup-Komponente zum Wiederherstellen des Eingabewortes

Inhalt

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
- 4. Turing-Maschinen**
 - 4.1 Motivation
 - 4.2 Aufbau und Funktionsweise
 - 4.3 Mehrband-Turingmaschinen
 - 4.4 Unbeschränkte Grammatiken
 - 4.5 Linear beschränkte Automaten**
5. Entscheidbarkeit
6. Berechenbarkeit
7. Komplexität

- Monotone Grammatiken: Keine kürzenden Regeln
 - Ableitung von w enthält keine Wörter, die länger als w sind
- LBA: Turingmaschine, deren Band auf die Länge von w beschränkt ist
 - Marker kennzeichnen Grenzen von w
 - Schreib-/Lesekopf darf Marker nicht passieren oder überschreiben
- Mehrband-LBA möglich



Definition 4.21 (LBA)

Ein linear beschränkter Automat \mathcal{A} ist eine Turing-Maschine $(Q, \Sigma, \Gamma, \Delta, q_0, F)$ so dass

- $\{>, <\} \subset \Gamma \setminus \Sigma$ gilt und
- Δ Übergänge mit Markern nur in der Form $(q, >, >, r, q')$ oder $(q, <, <, l, q')$ enthalten.

Die von \mathcal{A} erkannte Sprache ist die Menge aller Wörter $w \in \Sigma^*$, für die von der Startkonfiguration $>q_0w<$ aus eine akzeptierende Stopkonfiguration erreicht wird.

Satz 4.22

Monotone Grammatiken und linear beschränkte Automaten beschreiben dieselbe Sprachklasse.

Beweis.

Transformation monotone Grammatik $G \rightsquigarrow$ LBA \mathcal{A}_G :

- wie für Typ-0-Grammatik: verwende 2-Band-LBA
 - speichere Eingabewort w auf Band 1
 - simuliere Operationen von G auf Band 2
 - akzeptiere, wenn Inhalte gleich sind
- G enthält keine kürzenden Regeln
 - \mathcal{A}_G geht in nicht akzeptierende Stopkonfiguration, wenn längeres Wort als w erzeugt würde

Transformation LBA $\mathcal{A} \rightsquigarrow$ monotone Grammatik $G_{\mathcal{A}}$:

- ähnlich wie für TM:
 - erzeuge zufälliges Eingabewort w ohne Blanks
 - simuliere Berechnung von \mathcal{A} auf w
 - rekonstruiere Eingabe
- keine kürzenden Regeln vorhanden



Satz 4.23

Die Klasse der kontextsensitiven Sprachen ist abgeschlossen unter $\cup, \cdot, ^*$.

Kontext beeinflusst Regelanwendung

\rightsquigarrow Konkatenation und Kleene-Stern aufwändiger als für kontextfreie Sprachen

Beispiel 4.24

- $\mathcal{L}_1 = \{a^n b^n \mid n \in \mathbb{N}^{\geq 1}\}$ mit $P_1 = \{S_1 \rightarrow aS_1b \mid ab\}$
- $\mathcal{L}_2 = \{b^n \mid n \in \mathbb{N}^{\geq 1}\}$ mit $P_2 = \{S_2 \rightarrow S_2b \mid b, \quad bS_2 \rightarrow aS_2\}$
- Fehlerhafte Konstruktion für $\mathcal{L}_1 \cdot \mathcal{L}_2$ analog zu KFG: $P = \{S \rightarrow S_1S_2\} \cup P_1 \cup P_2$

Ableitung: $S \Rightarrow S_1S_2 \Rightarrow^2 aabbS_2 \Rightarrow aabaS_2 \Rightarrow aabab \notin \mathcal{L}_1 \cdot \mathcal{L}_2$

Abhilfe:

- ersetze TS durch NTS (wie bei CNF) $\rightsquigarrow X_a, X_b, \dots$
- benenne NTS disjunkt um $\rightsquigarrow X_{a_1}, X_{a_2}, \dots$

Beweis.

Gegeben: Monotone Grammatiken $G_1 = (N_1, \Sigma, P_1, S_1)$ und $G_2 = (N_2, \Sigma, P_2, S_2)$

$\mathcal{L}(G_1) \cup \mathcal{L}(G_2)$ Wie für KFG:

- benenne NTS disjunkt um
- $P = \{S \rightarrow S_1, S \rightarrow S_2\} \cup P_1 \cup P_2$
- $\mathcal{L}(G_1) \cdot \mathcal{L}(G_2)$
 - führe für $c \in \Sigma$ neue NTS X_c und Regeln $X_c \rightarrow c$ ein
 - ersetze c in allen Regeln durch X_c
 - benenne NTS der Grammatiken disjunkt um
 - alle Regeln haben Form $N \rightarrow c$ oder $N_1 \dots N_k \rightarrow M_1 \dots M_j$
 - $P = \{S \rightarrow S_1 S_2\} \cup P_1 \cup P_2$
- $(\mathcal{L}(G_1))^*$
 - erzeuge Kopie G'_1 von G_1
 - behandle G_1 und G'_1 wie bei Konkatenation
 - $P = \{S \rightarrow \varepsilon | S_0, S_0 \rightarrow S_1 | S_1 S'_1 | S_1 S'_1 S_0\} \cup P_1 \cup P'_1$

Zusätzliche Regeln nötig, falls $\varepsilon \in \mathcal{L}(G_1)$ oder $\varepsilon \in \mathcal{L}(G_2)$ gilt. □

Übung 4.25

Gegeben sei die Grammatik $G_1 = (N, \Sigma_{abc}, P, S_1)$ mit $\mathcal{L}(G_1) = \{a^n b^n c^n \mid n \geq 1\}$:

- $N = \{S_1, B, C\}$,

- $P = \left\{ \begin{array}{lll} S_1 & \rightarrow & aSBC \quad 1 \\ S_1 & \rightarrow & aBC \quad 2 \\ CB & \rightarrow & BC \quad 3 \\ aB & \rightarrow & ab \quad 4 \\ bB & \rightarrow & bb \quad 5 \\ bC & \rightarrow & bc \quad 6 \\ cC & \rightarrow & cc \quad 7 \end{array} \right\}$

- 1 Verwenden Sie den gezeigten Algorithmus, um eine Grammatik G für die Sprache $\{a^n b^n c^n \mid n \geq 1\}^*$ zu erzeugen.
- 2 Geben Sie in G eine Ableitung für das Wort aabbccabc an.

Satz 4.26

Die Klasse der kontextsensitiven Sprachen ist abgeschlossen unter Durchschnitt.

Beweis.

Gegeben seien LBAs \mathcal{A}_1 und \mathcal{A}_2 und ein Wort w .

Der LBA \mathcal{A} für $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ verwendet 2 Bänder:

- 1 Kopiere w von Band 1 auf Band 2.
- 2 Simuliere Berechnung von \mathcal{A}_1 auf w auf Band 1.
 - Wenn \mathcal{A}_1 verwirft: verwaffe
 - Wenn \mathcal{A}_1 akzeptiert: Weiter mit Schritt 3
- 3 Simuliere Berechnung von \mathcal{A}_2 auf w auf Band 2.
 - Wenn \mathcal{A}_2 verwirft: verwaffe
 - Wenn \mathcal{A}_2 akzeptiert: akzeptiere

Wenn eine der beiden Simulationen nicht terminiert, terminiert auch \mathcal{A} nicht, d.h. die Eingabe wird nicht akzeptiert. □

Satz 4.27 (Immerman, Szelepcsényi 1988)

Die Klasse der kontextsensitiven Sprachen ist abgeschlossen unter Komplement.

Schwierigkeit: Berücksichtigung aller (evtl. exponentiell vielen) nichtdeterministischen Berechnungen mit **linearem Platz**

Ansatz: Speichere nicht alle Ableitungen, sondern nur ihre **Anzahl**

Grobe Beweisskizze.

Gegeben seien ein LBA \mathcal{A} und ein Wort w .

Der LBA $\overline{\mathcal{A}}$ für $\mathcal{L}(\mathcal{A})$ geht wie folgt vor:

- 1 Zähle a : alle möglichen Konfigurationen von \mathcal{A} mit Eingabe w
 - a ist beschränkt durch $2^{|w|}$
 - a kann binär codiert werden \rightsquigarrow Platzbedarf linear in $|w|$
- 2 Zähle b : alle **verwerfenden** Konfigurationen von \mathcal{A} mit Eingabe w
- 3 Wenn $a = b$: akzeptiere; sonst: verwerfe



Neil Immerman
(*1953)



Róbert Szelepcsényi
(*1966)

Satz 4.28

Das Wortproblem für kontextsensitive Sprachen ist entscheidbar.

Beweis.

Gegeben seien eine monotone Grammatik $G = (N, \Sigma, P, S)$ und ein Wort $w \in \Sigma$.

Der Test $\overset{?}{w \in \mathcal{L}(G)}$ funktioniert wie folgt:

- Berechne $M = \{v \in (N \cup \Sigma)^* \mid S \Rightarrow^* v \text{ und } |v| \leq |w|\}$
- $w \in \mathcal{L}(G)$ gilt genau dann, wenn $w \in M$ gilt.

M ist endlich und kann in endlicher Zeit berechnet werden, denn:

- N , Σ und P sind endlich.
 - Regeln von P sind nicht-kürzend.
-
- Algorithmus ist nicht auf linearen Platz beschränkt
 - Beweis ist einfacher als für Abgeschlossenheit unter Komplement



Satz 4.29

Das Leerheitsproblem für kontextsensitive Sprachen ist unentscheidbar.

Beweis.

Folgt aus Unentscheidbarkeit des Postschen Korrespondenzproblems (Kapitel 5.3). □

Satz 4.30

Das Äquivalenzproblem für kontextsensitive Sprachen ist unentscheidbar.

Beweis.

Wäre das Äquivalenzproblem für kontextsensitive Sprachen entscheidbar, dann auch für kontextfreie Sprachen (da jede kontextfreie Sprache auch kontextsensitiv ist). □

Zusammenfassung: Linear beschränkte Automaten

- Turingmaschine mit Start- und Endmarker auf Band
- Platz beschränkt auf Länge der Eingabe
- äquivalent zu monotonen Grammatiken
- abgeschlossen unter $\cup, \cap, \cdot, ^*, \overline{\quad}$
 - manche Konstruktionen sehr aufwändig
- Wortproblem entscheidbar
- Leerheits- und Äquivalenzproblem unentscheidbar

Typ	Sprachklasse	Grammatik	Maschinenmodell
0	rekursiv aufzählbar	unbeschränkt	Turing-Maschine
1	kontextsensitiv	monoton	linear beschränkter Automat
2	kontextfrei	kontextfrei	Kellerautomat
3	regulär	rechtslinear	endlicher Automat

Rückblick:

- Jede KFG kann in Chomsky-Normalform transformiert werden
- Nur Regeln der Form $A \rightarrow BC$ oder $A \rightarrow c$ nötig
- Funktioniert auch für rechtslineare Grammatiken: $A \rightarrow bC$ oder $A \rightarrow c$

Normalformen für Typ 0 und 1?

Typ 3 rechtslinear	Typ 2 kontextfrei	Typ 1 monoton	Typ 0 unbeschränkt
„Chomsky-NF“	Chomsky-NF	Kuroda-NF	„Kuroda-NF“
$A \rightarrow c$	$A \rightarrow c$	$A \rightarrow c$	$A \rightarrow c$
$A \rightarrow bC$	$A \rightarrow BC$	$A \rightarrow BC$ $AB \rightarrow CD$	$A \rightarrow BC$ $AB \rightarrow CD$ $A \rightarrow \epsilon$



Sige-Yuki Kuroda
(1934–2009)

- Turingmaschine: Endlicher Automat plus **unendliches Band**
 - lesen und schreiben
 - beliebige Bewegungen
- Unterschiedliche Varianten:
 - **Mehrband-TM**
 - deterministische TM
 - deterministische **Einband-TM** kann nichtdeterministische Mehrband-TM simulieren
- äquivalent zu **unbeschränkten Grammatiken**
- Linear beschränkter Automat
 - Bandlänge **beschränkt auf Eingabe**
 - äquivalent zu **monotonen Grammatiken**
 - **abgeschlossen** unter allen (betrachteten) Operationen
 - nur **Wortproblem** entscheidbar

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
4. Turing-Maschinen
- 5. Entscheidbarkeit**
 - 5.1 Unentscheidbarkeit des speziellen Wortproblems
 - 5.2 Reduktionsbeweise
 - 5.3 Das PKP und weitere unentscheidbare Probleme
 - 5.4 Semi-Entscheidbarkeit
 - 5.5 Die universelle Turing-Maschine
 - 5.6 Abschlusseigenschaften
6. Berechenbarkeit
7. Komplexität

Bisher: Informelle Vorstellung „Es gibt einen Algorithmus“

- Abfolge einzelner Schritte
- eindeutig verständlich
- jeder Schritt „offensichtlich“ machbar

Jetzt: Formale Definition mit Hilfe von TM

Definition 5.1 (Entscheidbarkeit)

Eine Menge S heißt entscheidbar, wenn eine deterministische Turing-Maschine \mathcal{M} existiert, die eine Eingabe w

- akzeptiert, wenn $w \in S$ gilt,
 - verwirft, wenn $w \notin S$ gilt.
-
- akzeptiert: Berechnung führt zu akzeptierender Stopkonfiguration
 - verwirft: Berechnung führt zu nicht akzeptierender Stopkonfiguration
 - S wird oft als Problem bezeichnet

erkennbar es gibt NTM mit

$w \in L$ eine Berechnung führt zu akzeptierender Stopkonfiguration

$w \notin L$ jede Berechnung führt zu nicht akzeptierender Stopkonfiguration
oder Endlosschleife

entscheidbar es gibt DTM mit

$w \in L$ die Berechnung führt zu akzeptierender Stopkonfiguration

$w \notin L$ die Berechnung führt zu nicht akzeptierender Stopkonfiguration

- Entscheidbar: TM muss immer terminieren
- Entscheidbare Sprachen sind erkennbar, aber nicht umgekehrt
- Erkennbare Sprachen sind **semi-entscheidbar** (Kapitel 5.4)

Beispiel 5.2 ($S_{abc} = \{a^n b^n c^n \mid n \in \mathbb{N}\}$)

Die Menge S_{abc} ist entscheidbar, denn die TM \mathcal{M}_{abc} (Beispiel 4.14) terminiert auf jeder Eingabe, und es gilt $\mathcal{L}(\mathcal{M}_{abc}) = S_{abc}$.

Beispiel 5.3 ($S_{\mathbb{P}} = \{a^p \mid p \text{ ist prim}\}$)

Die Menge $S_{\mathbb{P}}$ ist entscheidbar.

Die 3-Band-DTM $\mathcal{M}_{\mathbb{P}}$ geht wie folgt vor:

- Band 1 enthält Eingabe w
- Band 2 zählt mögliche Teiler t von w auf ($1 < t \leq w$)
- Band 3 zählt Vielfache v von t auf ($t < v \leq w$)

Notation: b_i steht für Inhalt von Band i

- 1 Wenn $|b_1| < 2$: verwirfe
- 2 Schreibe auf Band 2 ein a
- 3 Hänge an Band 2 ein a an
- 4 Wenn $|b_1| = |b_2|$: akzeptiere
- 5 Kopiere Band 2 auf Band 3
- 6 Hänge b_2 an b_3 an
- 7 Wenn $|b_1| = |b_3|$: verwirfe
- 8 Wenn $|b_1| < |b_3|$: weiter bei 3
- 9 Weiter bei 6

Übung 5.4

Der Vergleich von b_1 mit b_2 oder b_3 kann wie bei \mathcal{M}_{abc} durchgeführt werden. Es fehlt eine Methode zum Anhängen eines Bandinhalts an einen anderen Bandinhalt.

Schreiben Sie eine 2-Band-Turing-Maschine mit Alphabet $\{a\}$, die

- den Anfang von b_1 sucht,
- das Ende von b_2 sucht,
- b_1 an b_2 anhängt.

Sie können davon ausgehen, dass die Bänder jeweils nur ein Wort enthalten, und dass der jeweilige Schreib-/Lesekopf zu Anfang innerhalb des Wortes steht.

Satz 5.5

Sei Σ ein Alphabet und $L \subseteq \Sigma^*$ eine Sprache.

Ist L entscheidbar, dann auch das Komplement $\overline{L} = \Sigma^* \setminus L$.

Beweis.

Sei $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$ eine DTM, die L entscheidet.

Wir definieren $\overline{\mathcal{M}}$ als $(Q, \Sigma, \Gamma, \delta, q_0, Q \setminus F)$. $\overline{\mathcal{M}}$ entscheidet \overline{L} , denn

- $\overline{\mathcal{M}}$ ist deterministisch;
- $\overline{\mathcal{M}}$ hält auf jeder Eingabe;
- $\overline{\mathcal{M}}$ akzeptiert genau die Wörter, die \mathcal{M} verwirft.



1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
4. Turing-Maschinen
- 5. Entscheidbarkeit**
 - 5.1 Unentscheidbarkeit des speziellen Wortproblems
 - 5.2 Reduktionsbeweise
 - 5.3 Das PKP und weitere unentscheidbare Probleme
 - 5.4 Semi-Entscheidbarkeit
 - 5.5 Die universelle Turing-Maschine
 - 5.6 Abschlusseigenschaften
6. Berechenbarkeit
7. Komplexität

Wozu Unentscheidbarkeit untersuchen?

Turing-Maschinen

- komplexer als bisherige Berechnungsmodelle (EA, KA)
- gleich mächtig wie Computer
- nur rudimentäre Funktionen

Nachteil: umständlich in der Anwendung

Vorteil: ermöglicht Beweise



(Scott Adams hat Wirtschaftswissenschaften und Management studiert...)

Es ist möglich, zu beweisen, dass eine **Turing-Maschine** (oder ein Computer) etwas nicht kann!

Wie kann Unentscheidbarkeit eines Problems P gezeigt werden?

- ohne formalen Algorithmen-Begriff
 - gar nicht
 - Asok: „It's not logically possible to prove something can't be done.“
- mit Turing-Maschine:
 - Schwierigkeit ähnlich wie bei Pumping-Lemma
 - Zeige für **unendlich** viele Turingmaschinen \mathcal{A} , dass \mathcal{A} das Problem P **nicht** entscheidet
 - indirekter Beweis: Annahme „ \mathcal{A} entscheidet P “ führt zu Widerspruch
 - Dan: „It's impossible for most people, but I'm a trained scientist.“

Nachweis der Existenz unentscheidbarer Probleme:

- 1 Codierung von Turingmaschinen
- 2 Definition des speziellen Wortproblems
- 3 Nachweis der Unentscheidbarkeit

- Turing-Maschinen können als Eingabe für eine TM \mathcal{M} codiert werden
- Schwierigkeit:
 - \mathcal{M} hat **fixes** Alphabet und **fixe** Zustandsmenge
 - \mathcal{M} muss als Eingabe beliebig große Alphabete und Zustandsmengen verarbeiten können
- Ziel: \mathcal{M} simuliert die Berechnungen beliebiger TM (später)

Gödelisierung g mit Alphabet Σ_{ab} :

- Zustände $\{q_1, q_2, q_3, \dots\}$: a, aa, aaa, ...
- Alphabet $\{c_1, c_2, c_3, \dots\}$: a, aa, aaa, ...
- Kopfbewegung $\{l, n, r\}$: a, aa, aaa

Beispiel 5.6 (Gödelisierung einer TM)

Übergang $(q_3, c_1, c_3, r, q_1) \mapsto \text{aaababaaaabaaaba}$
Endzustandsmenge $\{q_1, q_2, q_4\} \mapsto \text{abaabaaaa}$
Konfiguration $c_2c_4q_3c_1c_3 \mapsto \text{aabaaaabbbaaabbaaaa}$



Kurt Gödel
(1906–1978)

Definition 5.7 (Spezielles Wortproblem)

Sei T die Menge aller Codierungen von Turing-Maschinen und \mathcal{M}_w für $w \in T$ die TM mit Codierung w .

Das **spezielle Wortproblem** W ist die Menge aller Codierungen von Turing-Maschinen, die ihre eigene Codierung akzeptieren:

$$W = \{w \mid w \in \mathcal{L}(\mathcal{M}_w)\}$$

Aus Satz 5.5 folgt: Ist W entscheidbar, dann auch $\overline{W} = \{w \mid w \notin \mathcal{L}(\mathcal{M}_w)\}$.

Satz 5.8 (Unentscheidbarkeit des speziellen Wortproblems)

Das spezielle Wortproblem ist unentscheidbar.

Beweis.

Wir zeigen indirekt die Unentscheidbarkeit von \overline{W} .

Angenommen, es gibt eine TM $\mathcal{M}_{\overline{W}}$, die \overline{W} entscheidet, und die die Codierung $w_{\overline{W}}$ hat.

Dann gilt:

$$w_{\overline{W}} \in \mathcal{L}(\mathcal{M}_{\overline{W}}) \quad \text{gdw} \quad \mathcal{M}_{\overline{W}} \text{ akzeptiert } w_{\overline{W}} \quad \text{gdw} \quad w_{\overline{W}} \notin \mathcal{L}(\mathcal{M}_{\overline{W}}) \quad \square$$



Analogie mit Computer-Programmen

Turing-Maschine M ausgeführtes Programm

Gödelisierung $g(M)$ Programm als Binärdatei

akzeptieren terminieren mit Rückgabewert 0

\overline{W} Menge aller Programme, die sich selbst als Eingabe **nicht** akzeptieren

$M_{\overline{W}}$ Programm *Opposite*, das genau die Programme akzeptiert, die sich selbst als Eingabe **nicht** akzeptieren

Angenommen, *Opposite* existiert.

Was macht *Opposite*, wenn es sich selbst als Eingabe erhält?

- 1 *Opposite* akzeptiert. Da *Opposite* genau die Programme akzeptiert, die sich selbst nicht akzeptieren, folgt, dass *Opposite* sich selbst nicht akzeptiert. ↴
- 2 *Opposite* akzeptiert **nicht**. Da *Opposite* genau die Programme akzeptiert, die sich selbst nicht akzeptieren, folgt, dass *Opposite* sich selbst akzeptiert. ↴

Also existiert *Opposite* **nicht**.

Schwierigkeit des Beweises:

Zeige für alle unendlich vielen Turing-Maschinen, dass sie das spezielle WP nicht entscheiden.

TM	Eingabe	$g(\mathcal{M}_1)$	$g(\mathcal{M}_2)$	$g(\mathcal{M}_3)$	$g(\mathcal{M}_4)$	$g(\mathcal{M}_5)$...
\mathcal{M}_1		X					
\mathcal{M}_2			X				
\mathcal{M}_3				X			
\mathcal{M}_4					X		
\mathcal{M}_5						X	
:							..

Diagonale: TM mit eigener Codierung als Eingabe \rightsquigarrow Widerspruch

Satz 5.9 (Cantor-Diagonalisierung, 1891)

Die Menge der reellen Zahlen ist nicht abzählbar.

Satz 5.10 (Paradoxon des Epimenides, 6. Jh. v. Chr.)

Epimenides [der Kreter] sagt: „Alle Kreter lügen [immer].“

Satz 5.11 (Russell'sche Antinomie, 1903)

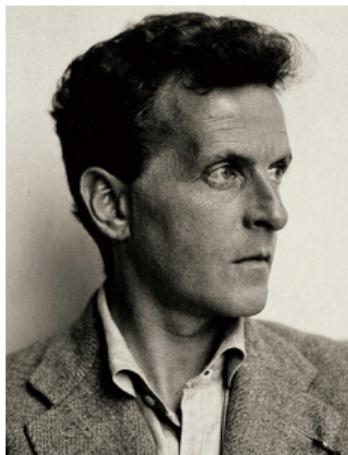
$R := \{T \mid T \notin T\}$. Gilt $R \in R$?

Satz 5.12 (Gödels Unvollständigkeits-Satz, 1931)

Konstruktion eines Satzes der Prädikatenlogik 2. Stufe, der aussagt, dass er selbst nicht bewiesen werden kann.

Ist das wichtig?

- Was ist schlimm daran, dass man nicht entscheiden kann, ob eine Turingmaschine ihre eigene Codierung akzeptiert?
- Ist dies ein rein theoretisches Problem?



Ludwig Wittgenstein
1889–1951

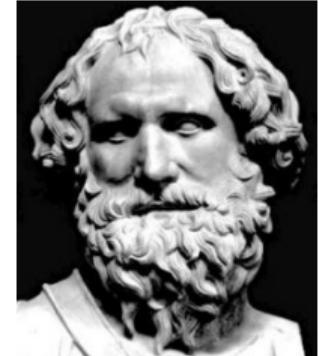
It is very queer that this should have puzzled anyone. [...] If a man says ‘I am lying’ we say that it follows that he is not lying, from which it follows that he is lying and so on. Well, so what? You can go on like that until you are black in the face. Why not? It doesn't matter.

Ludwig Wittgenstein, Lectures on the Foundations of Mathematics, Cambridge 1939

Or does it?

Gebt mir einen Hebel, und ich hebe die Welt aus den Angeln.

Archimedes



Archimedes
(287 v.–212 v.)

- Spezielles Wortproblem W als Ansatzpunkt
- Indirekter Beweis der Unentscheidbarkeit anderer Probleme P
- Idee: „Könnte ich P entscheiden, dann auch W .“



- Formaler Algorithmusbegriff (TM) ermöglicht Untersuchung von Unentscheidbarkeit
- Spezielles Wortproblem für Turingmaschinen ist unentscheidbar
 - indirekter Beweis
 - Gödelisierung
 - Diagonalisierung
- Ermöglicht weitere Unentscheidbarkeitsresultate

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
4. Turing-Maschinen
- 5. Entscheidbarkeit**
 - 5.1 Unentscheidbarkeit des speziellen Wortproblems
 - 5.2 Reduktionsbeweise**
 - 5.3 Das PKP und weitere unentscheidbare Probleme
 - 5.4 Semi-Entscheidbarkeit
 - 5.5 Die universelle Turing-Maschine
 - 5.6 Abschlusseigenschaften
6. Berechenbarkeit
7. Komplexität

Reduktion

„Zurückführung“ eines Problems auf ein allgemeineres

Definition 5.13

Eine **Reduktion** von $L_1 \subseteq T_1$ auf $L_2 \subseteq T_2$ ist eine mit einer TM berechenbare Funktion $f : T_1 \rightarrow T_2$, so dass für alle $w \in T_1$ gilt:

$$w \in L_1 \quad \text{gdw} \quad f(w) \in L_2$$

L_1 ist auf L_2 **reduzierbar** ($L_1 \leq L_2$), wenn es eine Reduktion von L_1 auf L_2 gibt.

Lemma 5.14

- 1 Wenn $L_1 \leq L_2$ gilt und L_2 entscheidbar ist, ist L_1 entscheidbar.
- 2 Wenn $L_1 \leq L_2$ gilt und L_1 unentscheidbar ist, ist L_2 unentscheidbar.

Beispiel 5.15 (Reduktion des Leerheitsproblems auf das Äquivalenzproblem für reguläre Sprachen)

- L_1 : Menge aller DEAs \mathcal{A} , die die leere Sprache erkennen
- L_2 : Menge aller Paare $(\mathcal{A}_1, \mathcal{A}_2)$ von DEAs, die dieselbe Sprache erkennen
- $\mathcal{A}_{\{\}}$: Trivialer DEA für leere Sprache
- $f = \mathcal{A} \mapsto (\mathcal{A}, \mathcal{A}_{\{\}})$

Ergebnis: Da ein Algorithmus existiert, der entscheidet, ob \mathcal{A} und $\mathcal{A}_{\{\}}$ äquivalent sind, haben wir jetzt einen Algorithmus, der entscheidet, ob \mathcal{A} die leere Sprache erkennt.

- Alltagsgeschäft des Informatikers
- Zurückführung eines neuen Problems
- auf ein altes, das bekanntermaßen lösbar ist

Übung 5.16

Reduzieren Sie das Problem „Akzeptiert der DEA \mathcal{A} das leere Wort?“ auf das Wortproblem für DEAs.

- T_1 : Trägermenge: Menge aller DEAs \mathcal{A}
- L_1 : Menge aller DEAs \mathcal{A} , die das leere Wort akzeptieren
- T_2 : Menge aller Paare (\mathcal{A}, w) von DEAs \mathcal{A} und Wörtern w
- L_2 : Menge aller Paare (\mathcal{A}, w) mit $w \in \mathcal{L}(\mathcal{A})$
- gesucht: $f : T_1 \rightarrow T_2$ mit $x \in L_1$ gdw $f(x) \in L_2$

Beispiel 5.17 (Reduktion des speziellen Wortproblems auf das (allgemeine) Wortproblem für TM)

- L_1 : Menge aller TM \mathcal{M} , die ihre eigene Codierung $g(\mathcal{M})$ akzeptieren
- L_2 : Menge aller Paare (\mathcal{M}, w) von TM \mathcal{M} und Wörtern w mit $w \in \mathcal{L}(\mathcal{M})$
- $f = \mathcal{M} \mapsto (\mathcal{M}, g(\mathcal{M}))$

Zu zeigen: $x \in L_1$ gdw $f(x) \in L_2$.

Beweis:

$$\mathcal{M} \in L_1 \quad \text{gdw.} \quad \mathcal{M} \text{ akzeptiert } g(\mathcal{M}) \quad \text{gdw.} \quad (\mathcal{M}, g(\mathcal{M})) \in L_2$$

Satz 5.18 (Wortproblem für Turingmaschinen)

Das Wortproblem für Turingmaschinen ist unentscheidbar.

Beweis.

- 1 Das spezielle WP ist auf das allgemeine WP reduzierbar.
- 2 Wäre das allgemeine WP entscheidbar, dann auch das spezielle.
- 3 Wir haben aber durch Diagonalisierung gezeigt, dass das spezielle WP unentscheidbar ist.
- 4 Also kann auch das allgemeine WP nicht entscheidbar sein. □

Definition 5.19 (Halteproblem)

Das Halteproblem ist die Menge aller Paare (\mathcal{M}, w) von Turingmaschinen \mathcal{M} und Wörtern w , so dass \mathcal{M} mit Eingabe w hält.

Satz 5.20

Das Halteproblem ist unentscheidbar.

Beweis.

Reduktion des Wortproblems auf das Halteproblem:

- $f = (\mathcal{M}, w) \mapsto (\mathcal{M}', w)$
- \mathcal{M}' verhält sich zunächst wie \mathcal{M} .
- Erreicht \mathcal{M} eine akzeptierende Stopkonfiguration, stoppt auch \mathcal{M}' .
- Erreicht \mathcal{M} eine nicht akzeptierende Stopkonfiguration, geht \mathcal{M}' in eine Endlosschleife.

Dann gilt:

- (mathcal{M}, w) ∈ WP
gdw \mathcal{M} akzeptiert w
gdw \mathcal{M} erreicht mit Eingabe w eine akzeptierende Stopkonfiguration
gdw \mathcal{M}' hält mit Eingabe w
gdw $(\mathcal{M}', w) ∈ HP$



Definition 5.21 (Leerheitsproblem)

Das Leerheitsproblem ist die Menge aller Turingmaschinen \mathcal{M} , die die leere Sprache erkennen.

Satz 5.22

Das Leerheitsproblem ist unentscheidbar.

Der Beweis verwendet das Komplement des Halteproblems:

Definition 5.23 (Endlosschleifen-Problem)

Das Endlosschleifen-Problem ist die Menge aller Paare (\mathcal{M}, w) von Turingmaschinen \mathcal{M} und Wörtern w , so dass \mathcal{M} mit Eingabe w nicht hält.

Satz 5.24

Das Endlosschleifen-Problem ist unentscheidbar.

Beweis.

Angenommen, das Endlosschleifen-Problem ist entscheidbar.

Dann folgt aus Satz 5.5, dass auch das Halteproblem entscheidbar ist. $\frac{1}{2}$



Beweis.

Reduktion des Endlosschleifen-Problems auf das Leerheitsproblem:

- $f = (\mathcal{M}, w) \mapsto \mathcal{M}'$
- \mathcal{M}' ersetzt ihre eigene Eingabe durch w
- Danach verhält sich \mathcal{M}' wie \mathcal{M} .
 - Berechnung hängt nur von w ab, nicht von ursprünglicher Eingabe
 - \mathcal{M}' erzeugt gleiches Resultat für alle Eingaben
- Wenn eine Stopkonfiguration erreicht wird, akzeptiert \mathcal{M}' .

Dann gilt:

- \mathcal{M} hält nicht auf w
- gdw \mathcal{M}' geht mit jeder Eingabe in Endlosschleife
- gdw \mathcal{M}' erkennt die leere Sprache



Definition 5.25 (Äquivalenzproblem)

Das Äquivalenzproblem ist die Menge aller Paare $(\mathcal{M}, \mathcal{N})$ von Turingmaschinen, so dass \mathcal{M} und \mathcal{N} dieselbe Sprache erkennen.

Satz 5.26

Das Äquivalenzproblem für Turingmaschinen ist unentscheidbar.

Übung 5.27

Zeigen Sie die Unentscheidbarkeit des Äquivalenzproblems für Turingmaschinen, indem Sie das Leerheitsproblem auf das Äquivalenzproblem reduzieren.

D.h.: Geben Sie eine Abbildung f von der Menge aller TM in die Menge aller Paare von TM, so dass gilt:

$$\mathcal{M} \in \text{LP} \quad \text{gdw} \quad f(\mathcal{M}) \in \text{ÄP}$$

- Alle bisher betrachteten Entscheidungsprobleme sind unentscheidbar.
- Ebenso das Halteproblem.
- Halten ist eine fundamentale Eigenschaft.
- Wenn man das Halteproblem nicht entscheiden kann, was dann?

Satz 5.28 (Henry Gordon Rice (1920–2003), 1953)

Jede nicht-triviale semantische Eigenschaft von Turingmaschinen ist unentscheidbar.

nicht-trivial trifft auf einige TM zu, auf andere nicht

semantisch Eigenschaft bezieht sich nur auf erkannte Sprache, nicht auf Maschine selbst

Beispiel 5.29 (Entscheidbare Eigenschaften)

trivial Die von der TM erkannte Sprache besteht aus Wörtern über dem Eingabealphabet.

syntaktisch Die TM hat 42 Zustände.

Beispiel 5.30 (Eigenschaft E : TM erkennt eine unendliche Sprache)

Behauptung: Wenn E entscheidbar ist, dann auch das Halteproblem

Ansatz: Reduziere das Halteproblem auf E .

- $L_1 = \{(\mathcal{M}_H, w_H) \mid \mathcal{M}_H \text{ hält auf } w_H\}$ (Halteproblem)
- $L_2 = \{\mathcal{M}_E \mid \mathcal{M}_E \text{ hat Eigenschaft } E\}$
- $f = (\mathcal{M}_H, w_H) \mapsto \mathcal{M}'_E$ wie folgt

- Sei \mathcal{M}_E eine TM mit Eigenschaft E und w_E eine Eingabe für \mathcal{M}_E
- Definiere TM \mathcal{M}'_E mit Eingabe w_E :
 - 1 führe Berechnung von \mathcal{M}_H auf w_H durch
 - endet nach endlich vielen Schritten, wenn \mathcal{M}_H auf w_H hält
 - Endlosschleife, wenn \mathcal{M}_H auf w_H nicht hält
 - 2 führe Berechnung von \mathcal{M}_E auf w_E durch
 - akzeptiert, wenn $w_E \in \mathcal{L}(\mathcal{M}_E)$ gilt
 - akzeptiert nicht, sonst
- Teste, ob \mathcal{M}'_E die Eigenschaft E hat, also eine unendliche Sprache erkennt (ist laut Annahme entscheidbar)
 - ja $\rightsquigarrow \mathcal{M}_H$ hält mit Eingabe w_H
 - nein $\rightsquigarrow \mathcal{M}_H$ hält nicht mit Eingabe w_H

- **Reduktion:** Berechenbare Funktion von Problem L_1 in Problem L_2
- Aus $L_1 \leq L_2$ und Unentscheidbarkeit von L_1 folgt **Unentscheidbarkeit von L_2**
- Spezielles Wortproblem kann auf andere Probleme von TM reduziert werden
 - allgemeines Wortproblem
 - Halteproblem
 - Leerheitsproblem
 - Äquivalenzproblem
- Rice: **Jedes interessante Problem** für Turingmaschinen ist unentscheidbar.

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
4. Turing-Maschinen
- 5. Entscheidbarkeit**
 - 5.1 Unentscheidbarkeit des speziellen Wortproblems
 - 5.2 Reduktionsbeweise
 - 5.3 Das PKP und weitere unentscheidbare Probleme**
 - 5.4 Semi-Entscheidbarkeit
 - 5.5 Die universelle Turing-Maschine
 - 5.6 Abschlusseigenschaften
6. Berechenbarkeit
7. Komplexität

- Bisher: Entscheidungsprobleme für Turing-Maschinen
 - Ergebnis: Kein interessantes Problem ist entscheidbar
 - Praktische Relevanz: Hauptsächlich für Programmierer
- Jetzt: Unentscheidbare Probleme aus anderen Bereichen
 - Postsches Korrespondenzproblem
 - Entscheidungsprobleme für kontextfreie und -sensitive Sprachen
 - Gültigkeitsproblem für Formeln der Prädikatenlogik
(Das „Entscheidungsproblem“, Hilbert und Ackermann, 1928)
 - Hilbertsche Probleme (1900)
 - 2 Widerspruchsfreiheit von Axiomensystemen
 - 10 Existenz ganzzahliger Lösungen für Polynome mit mehreren Variablen



David Hilbert
(1862–1943)



Wilhelm Ackermann
(1896–1962)

Definition 5.31 (Postsches Korrespondenzproblem)

Eine Instanz des Postschen Korrespondenzproblems (PKP) ist eine endliche Folge von Paaren von nichtleeren Wörtern über einem endlichen Alphabet Σ :

$$P = ((l_1, r_1), (l_2, r_2), \dots, (l_n, r_n))$$

Eine Lösung von P ist eine Indexfolge i_1, i_2, \dots, i_m mit $m > 0$ und $i_j \in \{1, \dots, n\}$, so dass gilt:

$$l_{i_1} \cdot l_{i_2} \cdot \dots \cdot l_{i_m} = r_{i_1} \cdot r_{i_2} \cdot \dots \cdot r_{i_m}$$



Emil Leon Post
(1897–1954)

Beispiel 5.32

Sei $\Sigma = \Sigma_{ab}$.

1 $P_1 = ((a, aaa), (abaa, ab), (aab, b))$

Lösung 1: (2, 1)

- abaaa
- abaaa

2 $P_2 = (ab, aba), (baa, aa), (aba, baa))$

Keine Lösung:

- Lösung muss mit Paar 1 beginnen
- Danach nur Paar 3 möglich (beliebig oft)
- Erstes Wort bleibt immer kürzer als zweites

Lösung 2: (1, 3)

- aaab
- aaab

- ababaabaaba...
- ababaabaabaa...

Nachweis, dass es keine Lösung gibt, kann sehr schwierig sein.

Übung 5.33

Bestimmen Sie, falls vorhanden, die Lösungen der folgenden Instanzen des PKP.

- 1 $P_1 = ((abb, ba), (a, aabba), (aa, b), (ab, a))$
- 2 $P_2 = ((b, bb), (aa, aba), (a, ab))$
- 3 $P_3 = ((ab, aba), (a, b), (abb, bab))$

- Ziel: Beweis der Unentscheidbarkeit des PKP
- Methode: Reduktion $\text{HP} \leq \text{PKP}$
- Zwischenschritt: [Modifiziertes PKP](#)
- Reduktion $\text{HP} \leq \text{MPKP} \leq \text{PKP}$

Definition 5.34 (Modifiziertes PKP (MPKP))

Eine [Instanz](#) des MPKP ist definiert wie eine Instanz des PKP.

Eine [Lösung](#) des MPKP ist eine Indexfolge, die mit 1 beginnt.

Es zählen nur Lösungen, die mit dem ersten Wortpaar beginnen.

Lemma 5.35

Das MPKP kann auf das PKP reduziert werden.

- Zu zeigen: Man kann eine Instanz M des MPKP in eine Instanz P des PKP übersetzen, so dass M eine Lösung hat gdw. P eine Lösung hat.
- Schwierigkeit: Folgen, die nicht mit 1 beginnen, aber gleiche Wörter erzeugen
- Sei $M = ((l_1, r_1), \dots, (l_n, r_n))$ Instanz des MPKP mit Alphabet Σ .
- Seien | und \$ Symbole, die nicht in Σ vorkommen.
- Definiere $P = ((l'_1, r'_1), \dots, (l'_n, r'_n), (l'_{n+1}, r'_{n+1}), (l'_{n+2}, r'_{n+2}))$ mit
 - für $1 \leq i \leq n$:
füge in l'_i nach jedem Symbol ein | ein
füge in r'_i vor jedem Symbol ein | ein
z.B. $(aba, ab) \mapsto (a|b|a|, |a|b)$
 - $l'_{n+1} = |l'_1, r'_{n+1} = r'_1$; z.B. $(aba, ab) \mapsto (|a|b|a|, |a|b)$
 - $l'_{n+2} = $, r'_{n+2} = |$$

Beispiel 5.36

- $M = ((a, aba), (bab, b))$
- $P = ((a|, |a|b|a), (b|a|b|, |b), (|a|, |a|b|a), (\$, |\$))$
- Die Lösung (1, 2) für M entspricht der Lösung (3, 2, 4) von P :
 - abab
 - abab
 - |a|b|a|b|\$
 - |a|b|a|b|\$
- Die Folge (2, 1) erzeugt identische Wörter, beginnt aber nicht mit 1 und ist deshalb keine Lösung für M .
Sie erzeugt auch keine Lösung für P , da l'_2 mit b beginnt und r_2 mit |.

Behauptung: M hat eine Lösung gdw. P eine Lösung hat.

Beweis.

\Rightarrow Sei (i_1, \dots, i_m) eine Lösung von M .

- Nach Definition des MPKP gilt $i_1 = 1$.
- Dann folgt: $(k + 1, i_2, \dots, i_m, k + 2)$ ist Lösung von P .

\Leftarrow Sei (i_1, \dots, i_m) eine Lösung von P .

- Es gilt $i_1 = k + 1$, da alle anderen Paare mit unterschiedlichen Symbolen anfangen.
- Es gibt ein $t \leq m$ mit $i_t = k + 2$, da alle anderen Paare mit unterschiedlichen Symbolen enden.
- Für das kleinste solche t gilt: $(1, i_2, \dots, i_{t-1})$ ist Lösung von M .



Lemma 5.37

Das Halteproblem kann auf das MPKP reduziert werden.

Ansatz: zu einer terminierenden Berechnung von \mathcal{M} auf w (HP)

$$c_0 \vdash c_1 \vdash \dots \vdash c_n$$

existiert ein Lösungswort für $P_{\mathcal{M}, w}$ (MPKP)

$$!c_0!c_1!\dots!c_n!\dots!c_{n+k}!$$

Wortpaare von $P_{\mathcal{M}, w}$

- Sei $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_0, F)$ und $w \in \Sigma^*$
- Alphabet für $P_{\mathcal{M}, w}$: $Q \cup \Gamma \cup \{!, \otimes\}$ (mit $\{!, \otimes\} \cap (Q \cup \Gamma) = \{\}$)

Anfangspaar	$(!, !!q_0w)$		
Kopierpaare	(c, c)	für	$c \in \Gamma \cup \{!\}$
Übergangspaare	(pa, qb)	für	$(p, a, b, \textcolor{blue}{n}, q) \in \Delta$
	(pa, bq)	für	$(p, a, b, \textcolor{blue}{r}, q) \in \Delta$
	(dpa, qdb)	für	$(p, a, b, \textcolor{blue}{l}, q) \in \Delta$ und $d \in \Gamma$
	$(!pa, !q\Box b)$	für	$(p, a, b, \textcolor{blue}{l}, q) \in \Delta$
	$(p!, qb!)$	für	$(p, \Box, b, \textcolor{blue}{n}, q) \in \Delta$
	$(p!, bq!)$	für	$(p, \Box, b, \textcolor{blue}{r}, q) \in \Delta$
	$(dp!, qdb!)$	für	$(p, \Box, b, \textcolor{blue}{l}, q) \in \Delta$ und $d \in \Gamma$
	$(!p!, !q\Box b!)$	für	$(p, \Box, b, \textcolor{blue}{l}, q) \in \Delta$
Löschpaare	(pa, \otimes)	falls es	keinen Übergang (p, a, \dots) gibt
	$(p!, \otimes!)$	falls es	keinen Übergang (p, \Box, \dots) gibt
	$(c\otimes, \otimes)$	für	$c \in \Gamma$
	$(\otimes c, \otimes)$	für	$c \in \Gamma$
Abschlusspaar	$(\otimes!!, !)$	für	Stopzustand $q \in Q$

- Linkes Wort endet mit Ausgangskonfiguration, rechtes Wort endet mit Folgekonfiguration
- Vom Übergang nicht betroffene Wortteile werden durch Kopierpaare übertragen
- Änderungen gemäß Δ werden durch Übergangspaare vorgenommen
 - Sonderfälle mit \square bei Erreichen des Bandendes
- Anfangspaar muss erstes Paar der MPKP-Instanz sein
 - für PKP gäbe es wegen Kopierpaaren immer eine Lösung
- Löschpaare sind notwendig, damit nach Erreichen einer Stopkonfiguration die Wörter gleich werden

Übung 5.38

- 1 Erzeugen Sie eine Turing-Maschine \mathcal{M} mit dem Alphabet Σ_{ab} , die
 - auf Wörtern der Sprache a^*b^* hält,
 - auf anderen Wörtern nicht hält.
- 2 Erzeugen Sie aus \mathcal{M} die MPKP-Instanz $P_{\mathcal{M},aab}$.
- 3 Verdeutlichen Sie, wie sich aus der (terminierenden) Berechnung von \mathcal{M} auf dem Wort aab eine Lösung für $P_{\mathcal{M},aab}$ ergibt.

Beweis.

Für eine TM \mathcal{M} existiert eine terminierende Berechnung auf w gdw. eine Lösung für $P_{\mathcal{M},w}$ existiert.

⇒ Sei C eine terminierende Berechnung von \mathcal{M} auf w

- Kopier- und Übergangspaare erzeugen aus Startkonfiguration jede Folgekonfiguration
- Stopkonfiguration wird erreicht, da C terminiert
- Löschpaare Löschen alle Bandsymbole
- Mit Abschlusspaar wird Gleichheit der Wörter hergestellt

⇐ Sei $I = (i_1, \dots, i_m)$ eine Lösung von $P_{\mathcal{M},w}$

- Wegen $i_1 = 1$ ist rechtes Wort anfangs länger als linkes Wort
- Nur Lösch- und Abschlusspaare können Längenunterschied verringern
- Da Gleichheit der Wörter erreicht wird (I ist Lösung), muss eine Stopkonfiguration erreicht worden sein.



Korollar 5.39

Das MPKP und das PKP sind unentscheidbar.

- Das PKP selbst ist kein Problem aus der Praxis des Informatikers ...
- ... aber es kann auf viele praktische Probleme reduziert werden.
 - Entscheidungsprobleme für kontextsensitive und kontextfreie Sprachen
 - Gültigkeit prädikatenlogischer Formeln

Definition 5.40 (Disjunktionsproblem)

Das **Disjunktionsproblem** für kontextfreie Grammatiken (DP) ist die Menge aller Paare (G_1, G_2) kontextfreier Grammatiken, für die gilt: $\mathcal{L}(G_1) \cap \mathcal{L}(G_2) = \{\}$.

Lemma 5.41

Das Disjunktionsproblem ist unentscheidbar.

Ansatz: Reduktion des PKP auf das Komplement des DP

- Bilde Instanz $P = ((l_1, r_1), \dots, (l_n, r_n))$ des PKP mit Alphabet Σ ab auf Instanz (G_l, G_r) des DP:
 - $G_l = (\{S_l\}, \Sigma_G, P_l, S_l)$
 - $G_r = (\{S_r\}, \Sigma_G, P_r, S_r)$
 - $\Sigma_G = \Sigma \cup \{1, \dots, n\}$
 - $P_l = \{S_l \rightarrow i S_l l_i | i l_i \mid 1 \leq i \leq n\}$
 - $P_r = \{S_r \rightarrow i S_r r_i | i r_i \mid 1 \leq i \leq n\}$

Dann gilt:

- $\mathcal{L}(G_l) = \{i_m \dots i_1 \cdot l_{i_1}, \dots, l_{i_m} \mid i_j \in \{1, \dots, n\}\}$
- $\mathcal{L}(G_r) = \{i_m \dots i_1 \cdot r_{i_1}, \dots, r_{i_m} \mid i_j \in \{1, \dots, n\}\}$

Beispiel 5.42

- $P = ((ab, a), (bab, ab), (b, bbb))$
 - Lösung: (1, 3, 2)
 - abbbab
 - abbbab
- $G_l = (\{S_l\}, \{a, b, 1, 2, 3\}, P_l, S_l)$
 - $P_l = \{S_l \rightarrow 1S_lab|1ab|2S_lbab|2bab|3S_lb|3b\}$
 - $S_l \Rightarrow 2S_lbab \Rightarrow 23S_lbab \Rightarrow 231abbab$
- $G_r = (\{S_r\}, \{a, b, 1, 2, 3\}, P_r, S_r)$
 - $P_r = \{S_r \rightarrow 1S_ra|1a|2S_rab|2ab|3S_rbbb|3bbb\}$
 - $S_r \Rightarrow 2S_rab \Rightarrow 23S_rbbb \Rightarrow 231abbbab$

Behauptung: P hat eine Lösung gdw $\mathcal{L}(G_l)$ und $\mathcal{L}(G_r)$ ein gemeinsames Element haben.

Beweis.

- P hat Lösung (i_1, \dots, i_m)
- gdw. P hat Lösungswort $l_{i_1} \dots l_{i_m} = r_{i_1} \dots r_{i_m}$
- gdw. G_l erzeugt $i_m \dots i_1 \cdot l_{i_1} \dots l_{i_m}$ und G_r erzeugt $i_m \dots i_1 \cdot r_{i_1} \dots r_{i_m}$
- gdw. $\mathcal{L}(G_l)$ und $\mathcal{L}(G_r)$ haben ein gemeinsames Element.



Es folgt die Unentscheidbarkeit des Disjunkttheitsproblems:

- Das PKP ist unentscheidbar.
- $\text{PKP} \leq \overline{\text{DP}}$
- DP ist entscheidbar gdw. $\overline{\text{DP}}$ entscheidbar ist.

Satz 5.43

Das Leerheitsproblem für kontextsensitive Sprachen ist unentscheidbar.

Beweis.

Reduktion des DP auf das LP

- Gegeben seien kontextfreie Grammatiken G_1, G_2
- Konstruiere LBA \mathcal{A} mit $\mathcal{L}(\mathcal{A}) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$
 - kontextfreie Sprachen sind auch kontextsensitiv
 - kontextsensitive Sprachen sind abgeschlossen unter Durchschnitt
- Dann gilt: $\mathcal{L}(\mathcal{A}) = \{\}$ gdw $(G_1, G_2) \in \text{DP}$

□

Satz 5.44

Das Äquivalenzproblem für kontextsensitive Sprachen ist unentscheidbar.

Beweis.

Reduktion des LP auf das ÄP

- Gegeben sei eine Instanz G des LP
- Sei $G_{\{\}}$ eine monotone Grammatik, die die leere Sprache erzeugt.
- Dann gilt: $\mathcal{L}(G) = \{\}$ gdw $\mathcal{L}(G) = \mathcal{L}(G_{\{\}})$



Satz 5.45

Das Äquivalenzproblem für kontextfreie Sprachen ist unentscheidbar.

- $\mathcal{L}(G_l)$ und $\mathcal{L}(G_r)$ des DP sind **deterministisch** kontextfrei
 - können durch einen deterministischen KA erkannt werden
- Deterministisch kontextfreie Sprachen sind abgeschlossen unter Komplement (ohne Beweis)

Beweis.

Reduktion des DP auf das ÄP

$$\begin{array}{lcl} \mathcal{L}(G_l) \cap \mathcal{L}(G_r) & = & \{\} \\ \text{gdw} & \mathcal{L}(G_l) & \subseteq \overline{\mathcal{L}(G_r)} \quad \text{Mengenlehre} \\ \text{gdw} & \mathcal{L}(G_l) & \subseteq \mathcal{L}(\overline{G_r}) \quad \overline{G_r} \text{ sei KFG für } \overline{\mathcal{L}(G_r)} \\ \text{gdw} & \mathcal{L}(G_l) \cup \mathcal{L}(\overline{G_r}) & = \mathcal{L}(\overline{G_r}) \quad \text{Mengenlehre} \\ \text{gdw} & \mathcal{L}(G_{\cup}) & = \mathcal{L}(\overline{G_r}) \quad G_{\cup} \text{ sei KFG für } \mathcal{L}(G_l) \cup \mathcal{L}(\overline{G_r}) \end{array}$$



Ja, diese Probleme treten in der Praxis auf...
(stackexchange.com)



An efficient way to determine if two context free grammars are equivalent?



16



I'm wondering if there's an efficient way of checking to see if two context free grammars are equivalent, besides working out "test cases" by hand (ie, just trying to see if both grammars can generate the same things, and only the same things, by trial and error).



2

Thanks!



2

computer-science context-free-grammar

[share](#) [cite](#) [improve this question](#)

asked Nov 6 '12 at 4:23



pauliwago

330 5 ▲ 12

[add a comment](#)

2 Answers

[active](#) [oldest](#) [votes](#)



21



There is not. In fact, there isn't even an inefficient way!

That is, the problem of determining whether two given CFGs represent the same language is undecidable. In fact, an even stronger statement is true: the problem of determining whether a given CFG accepts all strings on its alphabet is undecidable.

Prädikatenlogische Formeln

- Variablen x, y
- Konstantensymbole c, d
- Funktionssymbole $f(x), g(c, d)$
- Prädikate $P(x), R(c, y)$
- Quantoren $\forall x\varphi(x), \exists y\psi(y)$
- Junktoren $\varphi \rightarrow \psi, \neg(\varphi \wedge \psi)$

Eine Formel φ ist **gültig**, wenn φ in jeder Interpretation wahr ist.

Definition 5.46 (Gültigkeitsproblem)

Das **Gültigkeitsproblem** der Prädikatenlogik ist die Menge aller gültigen prädikatenlogischen Formeln.

Reduktion des PKP auf das Gültigkeitsproblem (GP):

- Übersetzung einer Instanz P des PKP in eine prädikatenlogische Formel φ_P , so dass gilt:
- P hat eine Lösung gdw. φ_P gültig ist.
- Dann folgt: Das Gültigkeitsproblem ist unentscheidbar.

- Gegeben: $P = ((l_1, r_1), \dots, (l_n, r_n))$ mit Alphabet $\Sigma = \{c_1, \dots, c_k\}$
- Signatur für φ_P :
 - Konstantensymbol ε
 - Funktionssymbole $c_1^{(1)}, \dots, c_k^{(1)}$
 - Prädikat $R^{(2)}$
- Intuition:
 - Domäne: Wörter über Σ
 - ε : Leeres Wort
 - $c_i(x)$: Konkatenation von x mit c_i
 - $R(x, y)$: Paar (x, y) ist durch Paare von P erzeugbar

Beispiel 5.47

- $P = (\textcolor{red}{(ab, a)}, \textcolor{blue}{(bab, ab)}, \textcolor{green}{(b, bbb)})$
- Wort $abb \mapsto$ Term $b(b(a(\varepsilon)))$
- Paar $(ab, a) \mapsto$ Atom $\textcolor{red}{R}(b(a(\varepsilon)), a(\varepsilon))$

Formel für P

- $P = ((l_1, r_1), \dots, (l_n, r_n))$ mit Alphabet $\Sigma = \{c_1, \dots, c_k\}$
- Für $w = c_1 \cdot \dots \cdot c_m \in \Sigma^*$ sei $f_w(x)$ der Term $c_m(\dots(c_1(x)))$

$$\begin{aligned}\varphi_P &= \varphi_1 \wedge \varphi_2 \rightarrow \varphi_3 \\ \varphi_1 &= \bigwedge_{i=1, \dots, n} R(f_{l_i}(\varepsilon), f_{r_i}(\varepsilon)) && \text{Start-Paare} \\ \varphi_2 &= \forall x \forall y (R(x, y) \rightarrow \bigwedge_{i=1, \dots, n} R(f_{l_i}(x), f_{r_i}(y))) && \text{Folge-Paare} \\ \varphi_3 &= \exists x R(x, x) && \text{Es gibt eine Lösung}\end{aligned}$$

Beispiel 5.48

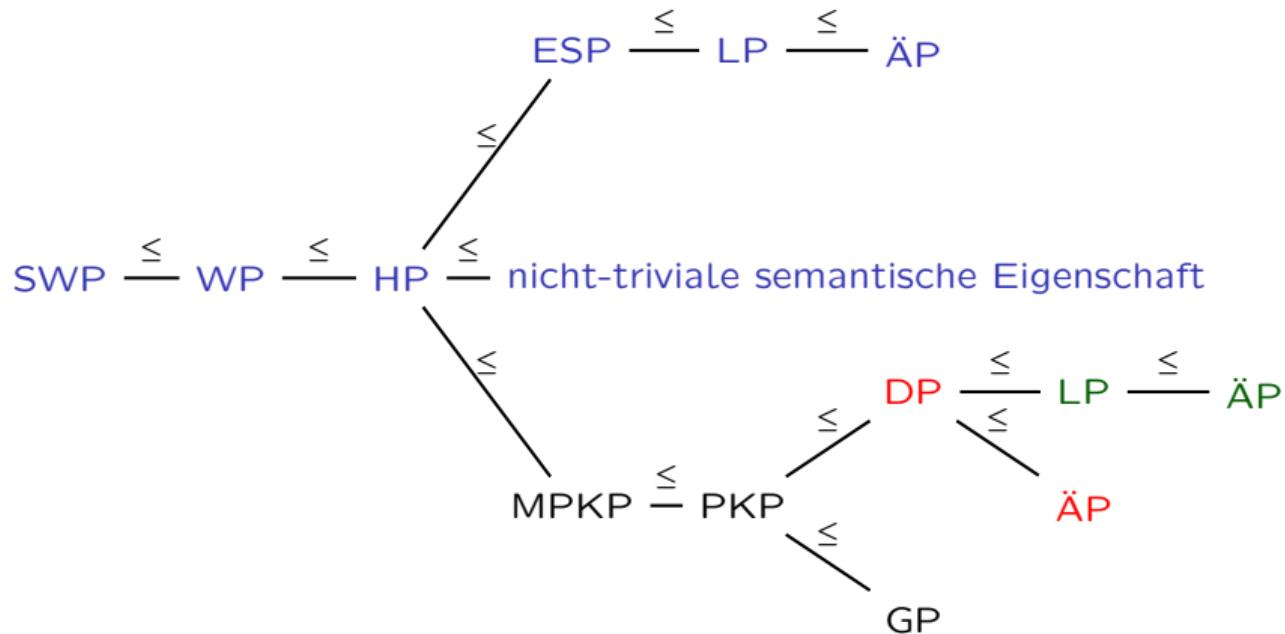
$$\begin{aligned}P &= (\textcolor{red}{(ab, a)}, \textcolor{blue}{(bab, ab)}, \textcolor{green}{(b, bbb)}) \\ \varphi_1 &= R(\textcolor{red}{b(a(\varepsilon))}, \textcolor{red}{a(\varepsilon)}) \wedge R(\textcolor{blue}{b(a(b(\varepsilon)))), b(a(\varepsilon))) \wedge R(\textcolor{green}{b(\varepsilon)}, \textcolor{green}{b(b(b(\varepsilon))))}) \\ \varphi_2 &= \forall x \forall y (R(x, y) \rightarrow R(\textcolor{red}{b(a(x))}, \textcolor{red}{a(y)}) \wedge R(\textcolor{blue}{b(a(b(x)))), b(a(y))) \wedge R(\textcolor{green}{b(x)}, \textcolor{green}{b(b(b(y))))})) \\ \varphi_3 &= \exists x R(x, x)\end{aligned}$$

$P = ((ab, a), (bab, ab), (b, bbb))$; Lösung: (1, 3, 2); Lösungswort: abbbab

$$\begin{array}{ll} R(b(a(\varepsilon)), a(\varepsilon)) & \text{wegen } \varphi_1 \\ R(b(b(a(\varepsilon))), b(b(b(a(\varepsilon))))) & \text{wegen } \varphi_2 \\ R(b(a(b(b(b(a(\varepsilon)))))), b(a(b(b(b(a(\varepsilon))))))) & \text{wegen } \varphi_2 \\ \\ \rightsquigarrow \varphi_3 \text{ wird wahr} & \end{array}$$

- Viele Interpretationen machen φ_P wahr, z.B. wenn $R^I = \Delta^I \times \Delta^I$
- Nur wenn eine Lösung für P existiert, ist φ_P in allen Interpretationen wahr (**gültig**)

Zusammenfassung: Unentscheidbare Probleme



Turing-Maschinen / rekursiv aufzählbare Sprachen
kontextsensitive Sprachen
kontextfreie Sprachen

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
4. Turing-Maschinen
- 5. Entscheidbarkeit**
 - 5.1 Unentscheidbarkeit des speziellen Wortproblems
 - 5.2 Reduktionsbeweise
 - 5.3 Das PKP und weitere unentscheidbare Probleme
- 5.4 Semi-Entscheidbarkeit**
 - 5.5 Die universelle Turing-Maschine
 - 5.6 Abschlusseigenschaften
6. Berechenbarkeit
7. Komplexität

- Gültigkeit in der Prädikatenlogik ist unentscheidbar
- Aber: Tableau- und Resolutionsalgorithmus sind
 - korrekt: leere Klausel/Clash ableitbar $\rightsquigarrow \neg\varphi$ unerfüllbar, φ gültig
 - widerlegungsvollständig: φ gültig, $\neg\varphi$ unerfüllbar \rightsquigarrow leere Klausel/Clash ableitbar
- φ gültig \rightsquigarrow Algorithmus gibt nach endlicher Zeit richtige Antwort
- φ nicht gültig \rightsquigarrow Algorithmus terminiert nicht immer, gibt aber keine falsche Antwort
- Weniger als Entscheidbarkeit, aber „mehr als nichts“...

Definition 5.49 (semi-entscheidbar)

Sei Σ ein Alphabet. Eine Sprache $L \subseteq \Sigma^*$ heißt **semi-entscheidbar**, wenn es eine DTM gibt, die mit Eingabe $w \in \Sigma^*$

- terminiert, wenn $w \in L$ gilt,
- nicht terminiert sonst.

Definition 5.50 (rekursiv aufzählbar, Aufzähl-TM)

Eine Sprache $L \subseteq \Sigma^*$ heißt **rekursiv aufzählbar**, wenn eine Aufzähl-Turingmaschine existiert, die L aufzählt.

Eine **Aufzähl-TM** ist eine DTM $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_0, \{\})$ mit folgenden Eigenschaften:

- Es existiert ein **Ausgabezustand** $q_{\text{output}} \in Q$,
- Wann immer der Ausgabezustand erreicht wird, gilt der Bandinhalt von der Kopfposition bis zum ersten Symbol aus $\Gamma \setminus \Sigma$ als **Ausgabe**.

Die **von \mathcal{M} aufgezählte Sprache** ist die Menge aller Ausgaben, die \mathcal{M} ausgehend von q_0 bei leerem Band erzeugt.

Endzustandsmenge ist irrelevant:

- Für Semi-Entscheidbarkeit wird nur Terminierung gefordert
 - Verwerfen entspricht Endlosschleife
- Aufzähl-TM testet i.d.R. alle Elemente von Σ^* und terminiert nicht

Beispiel 5.51

$\mathcal{M}_{\text{odd}} = (\{0, 1, \text{output}\}, \{a\}, \{a, \square\}, \Delta, 0, \{\})$ zählt die Sprache $\{a, aaa, aaaaa, \dots\}$ auf.

	Ausgangszustand	lesen	schreiben	bewegen	Folgezustand
$\Delta =$	0	<input type="checkbox"/>	a	n	output
	output	a	a	l	1
	1	<input type="checkbox"/>	a	l	0

Übung 5.52

Geben Sie die Konfigurationsfolge von \mathcal{M}_{odd} bis zur dritten Ausgabe an.

Satz 5.53

Eine Sprache $L \in \Sigma^*$ ist rekursiv aufzählbar gdw. sie semi-entscheidbar ist.

Beweis „ \Rightarrow “.

Sei \mathcal{M}_a eine Aufzähl-TM für eine Sprache L und $\Sigma^* = \{w_1, w_2, w_3, \dots\}$.

Wir definieren die TM \mathcal{M}_s wie folgt:

- Eingabe w bleibt auf Eingabeband und wird nicht verändert.
- Auf zweitem Band startet \mathcal{M}_s das Aufzählverfahren für L und verhält sich wie \mathcal{M}_a .
- Wann immer der Ausgabezustand erreicht wird, wird die Ausgabe mit w verglichen.
 - Bei Gleichheit stoppt \mathcal{M}_s .
 - Sonst fährt sie mit dem Aufzählverfahren fort.
- Wenn das Aufzählverfahren terminiert, ohne dass w erzeugt wird, geht \mathcal{M}_s in Endlosschleife.



Beweis „ \Leftarrow “.

Sei \mathcal{M}_s ein Semi-Entscheidungsverfahren für L und $\Sigma^* = \{w_1, w_2, w_3, \dots\}$.

Wir definieren eine Aufzähl-TM \mathcal{M}_a wie folgt:

- 1 Führe einen Schritt der Berechnung von \mathcal{M}_s auf w_1 aus.
- 2 Führe zwei Schritte der Berechnung von \mathcal{M}_s auf w_1 und w_2 aus.
- ...
- n Führe n Schritte der Berechnung von \mathcal{M}_s auf w_1, \dots, w_n aus.
- ...

Wann immer \mathcal{M}_s auf einer Eingabe terminiert, gibt \mathcal{M}_a diese Eingabe aus und fährt dann fort.



Übung 5.54

Warum funktioniert der naive Ansatz nicht, alle Berechnungen von \mathcal{M}_s nacheinander zu simulieren?

Die im letzten Beweis verwendete Technik heißt **Dovetailing**.

- **verzahnt** verschiedene Berechnungen
- vermeidet Sackgasse, wenn eine Berechnung nicht terminiert



Satz 5.55

Eine Sprache $L \subseteq \Sigma^*$ ist entscheidbar gdw sowohl L als auch \overline{L} semi-entscheidbar sind.

Beweis.

⇒ Sei \mathcal{M} eine TM, die L entscheidet, d.h. bei jeder Eingabe terminiert.

Semi-Entscheidungsverfahren \mathcal{M}^+ und \mathcal{M}^- für L bzw. \overline{L} verhalten sich zunächst wie \mathcal{M} .

- Wenn \mathcal{M} die Eingabe **verwirft**, geht \mathcal{M}^+ in Endlosschleife
- Wenn \mathcal{M} die Eingabe **akzeptiert**, geht \mathcal{M}^- in Endlosschleife

⇐ Seien \mathcal{M}^+ und \mathcal{M}^- Semi-Entscheidungsverfahren für L bzw. \overline{L} .

Eine TM \mathcal{M} , die L entscheidet, simuliert auf zwei Bändern abwechselnd die Berechnungen von \mathcal{M}^+ und \mathcal{M}^- .

- Wenn \mathcal{M}^+ terminiert, akzeptiert \mathcal{M} .
- Wenn \mathcal{M}^- terminiert, verwirft \mathcal{M} .

□

- **Semi-Entscheidungsverfahren** für L :
 - terminiert mit Eingabe $w \in L$
 - terminiert nicht mit Eingabe $w \notin L$
- **Aufzähl**-Turingmaschine für L erzeugt sukzessive alle Elemente von L
- L ist semi-entscheidbar gdw. L rekursiv aufzählbar ist.
- L ist entscheidbar, wenn L und \overline{L} semi-entscheidbar sind.

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
4. Turing-Maschinen
- 5. Entscheidbarkeit**
 - 5.1 Unentscheidbarkeit des speziellen Wortproblems
 - 5.2 Reduktionsbeweise
 - 5.3 Das PKP und weitere unentscheidbare Probleme
 - 5.4 Semi-Entscheidbarkeit
- 5.5 Die universelle Turing-Maschine**
- 5.6 Abschlusseigenschaften
6. Berechenbarkeit
7. Komplexität

- \mathcal{U} ist eine DTM, die andere TM simuliert („TM-Emulator“)
 - Nicht möglich bei bisherigen Maschinenmodellen!
- da TM endliche Alphabete und Zustandsmengen haben, können sie in einem (Binär-) Alphabet mit einer Funktion $g()$ codiert werden (Gödelisierung wie in Kapitel 5.1)
- Eingabe:
 - Codierung $g(\mathcal{M})$ einer TM \mathcal{M} ,
 - Codierung $g(w)$ eines Eingabeworts w für \mathcal{M} .
- Mit Eingabe $g(\mathcal{M})$ und $g(w)$, verhält sich \mathcal{U} genau wie \mathcal{M} auf w :
 - \mathcal{U} akzeptiert gdw \mathcal{M} akzeptiert
 - \mathcal{U} hält gdw \mathcal{M} hält
 - \mathcal{U} läuft unendlich lange gdw \mathcal{M} unendlich lange läuft

Korollar 5.56

Jedes (mit Turingmaschinen) lösbarer Problem kann in Software gelöst werden.

Codierungsfunktion $g()$ mit Alphabet Σ_{ab} für $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_0, F)$:

Zustand	$g(q_i)$	=	a^i	$g(Q) = g(q_1)bg(q_2)bg(\dots)bg(q_n)$
				$g(F) = g(q_{f_1})bg(q_{f_2})bg(\dots)bg(q_{f_n})$
Symbol	$g(c_i)$	=	a^i	$g(\Sigma) = g(c_1)bg(c_2)bg(\dots)bg(c_m)$
				$g(\Gamma) = g(c_1)bg(c_2)bg(\dots)bg(c_l)$
Kopfbewegung	$g(l)$	=	a	
	$g(n)$	=	aa	
	$g(r)$	=	aaa	
Übergang	$g((q_i, c_j, c_k, m, q_l))$	=		$g(q_i)bg(c_j)bg(c_k)bg(m)bg(q_l)$
				$g(\Delta) = g(\Delta_1)bbg(\Delta_2)bb\dots bbg(\Delta_k)$
TM	$g(\mathcal{M})$	=		$g(Q)bbbgb(\Sigma)bbbgb(\Gamma)bbbgb(\Delta)bbbgb(q_0)bbbgb(F)$
Wort	$g(w)$	=		$g(w _1)bg(w _2)bg(\dots)bg(w _j)$
Konfiguration	$g(vqw)$	=		$g(v)bbg(q)bbg(w)$

Eingabe: $g(\mathcal{M})\textcolor{blue}{bbb}g(w)$

- 1 codiere Startkonfiguration auf zweitem Band
 - Bandinhalt links von Kopfposition (\square)
 - Startzustand q_0
 - Bandinhalt beginnend mit Kopfposition ($g(w)$)
- 2 finde Nachfolgekonfiguration mit Hilfe von $g(\mathcal{M})$
 - suche Übergang (q, s, \dots) in $g(\mathcal{M})$:
 - vergleiche aktuellen Zustand q von \mathcal{M} mit Übergängen in $g(\mathcal{M})$
 - vergleiche aktuelles Bandsymbol s mit Übergängen in $g(\mathcal{M})$
- 3 wenn kein Übergang möglich:
 - akzeptiere, wenn q Endzustand ist
 - stoppe
- 4 sonst: modifiziere aktuelle Konfiguration entsprechend gefundenem Übergang
- 5 weiter bei Schritt 2

Satz 5.57

Das Wortproblem für Turingmaschinen ist semi-entscheidbar.

Beweis.

Semi-Entscheidungsverfahren für das Wortproblem (\mathcal{M}, w) :

- Benutze \mathcal{U} zur Simulation der Berechnung von \mathcal{M} auf w .
- Wenn akzeptierende Stopkonfiguration erreicht wird, stoppe.
- Wenn nicht akzeptierende Stopkonfiguration erreicht wird, gehe in Endlosschleife.



Korollar 5.58

Das Komplement des Wortproblems ist nicht semi-entscheidbar.

Beweis.

Wäre es semi-entscheidbar, wäre das Wortproblem entscheidbar.



Satz 5.59

- 1 Das Leerheitsproblem ist **nicht** semi-entscheidbar.
- 2 Sein Komplement ist semi-entscheidbar.

Beweis.

2 Sei \mathcal{M} eine TM mit Alphabet Σ und (w_1, w_2, \dots) eine Aufzählung aller Wörter über Σ .
Semi-Entscheidungsverfahren für Nicht-Leerheit von $\mathcal{L}(\mathcal{M})$:

- 1 Führe **einen** Schritt der Berechnung von \mathcal{M} auf w_1 aus.
- 2 Führe **zwei** Schritte der Berechnung von \mathcal{M} auf w_1 und w_2 aus.

...

- n** Führe **n** Schritte der Berechnung von \mathcal{M} auf w_1, \dots, w_n aus.

...

Stoppe, wenn \mathcal{M} eine Eingabe akzeptiert.

- 1 folgt aus Unentscheidbarkeit des Leerheitsproblems.



Reduktionen funktionieren auch für Semi-Entscheidbarkeit:

Lemma 5.60

- 1 Wenn $L_1 \leq L_2$ gilt und L_2 semi-entscheidbar ist, ist L_1 semi-entscheidbar.
- 2 Wenn $L_1 \leq L_2$ gilt und L_1 nicht semi-entscheidbar ist, ist L_2 nicht semi-entscheidbar.

Beweis.

- 1 Semi-Entscheidungsverfahren für L_1 :
 - 1 Übersetze Instanz I für L_1 in $f(I)$
 - 2 Starte Semi-Entscheidungsverfahren \mathcal{M}_2 für L_2 mit $f(I)$
 - 3 Terminiere, falls \mathcal{M}_2 terminiert
- 2 Angenommen, L_2 wäre semi-entscheidbar. Dann könnte man die für Teil 1 skizzierte Methode verwenden, um ein Semi-Entscheidungsverfahren für L_1 zu erhalten. ↴ □

Satz 5.61

Weder das Äquivalenzproblem für TM noch sein Komplement ist semi-entscheidbar.

Beweis.

Reduktion des Komplements des Wortproblems ($\overline{\text{WP}}$) auf das Äquivalenzproblem ÄP (und sein Komplement $\overline{\text{ÄP}}$):

Sei (\mathcal{M}, w) Instanz von WP. Die DTM \mathcal{M}' ist wie folgt definiert:

- \mathcal{M}' ersetzt seine Eingabe durch w , verhält sich danach wie \mathcal{M} .
- Akzeptiert \mathcal{M} , stoppt \mathcal{M}' ; verwirft \mathcal{M} , geht \mathcal{M}' in Endlosschleife.

D.h. \mathcal{M}' akzeptiert jede Eingabe, wenn $w \in \mathcal{L}(\mathcal{M})$ gilt, sonst die leere Sprache.

Reduktion auf ÄP: $w \notin \mathcal{L}(\mathcal{M})$ gdw. $\mathcal{L}(\mathcal{M}') = \{\}$ gdw. $\mathcal{L}(\mathcal{M}') = \mathcal{L}(\mathcal{M}_{\{\}})$

Reduktion auf $\overline{\text{ÄP}}$: $w \notin \mathcal{L}(\mathcal{M})$ gdw. $\mathcal{L}(\mathcal{M}') \neq \Sigma^*$ gdw. $\mathcal{L}(\mathcal{M}') \neq \mathcal{L}(\mathcal{M}_{\Sigma^*})$

$\mathcal{M}_{\{\}}$: DTM für leere Sprache; \mathcal{M}_{Σ^*} : DTM für Σ^* .



Übung 5.62

Finden Sie den Fehler im folgenden „Beweis“ für die Semi-Entscheidbarkeit von $\overline{\text{ÄP}}$ (Dovetailing, analog zum Beweis für $\overline{\text{LP}}$):

Pseudo-Beweis.

Gegeben seien DTM \mathcal{M}_1 und \mathcal{M}_2 mit Eingabealphabet Σ .

Sei (w_1, w_2, w_3, \dots) eine Aufzählung aller Wörter von Σ^* .

- 1 Führe je **einen** Schritt der Berechnungen von \mathcal{M}_1 und \mathcal{M}_2 auf w_1 aus.
- 2 Führe je **zwei** Schritte der Berechnungen von \mathcal{M}_1 und \mathcal{M}_2 auf w_1 und w_2 aus.
...
- n Führe je **n** Schritte der Berechnungen von \mathcal{M}_1 und \mathcal{M}_2 auf w_1, \dots, w_n aus.
...

Wenn \mathcal{M}_1 und \mathcal{M}_2 zu unterschiedlichen Antworten kommen, stoppe.
Sonst fahre fort.

Satz 5.63

Das Komplement des Äquivalenzproblems für kontextsensitive Sprachen ist semi-entscheidbar.

Beweis.

Seien G_1 und G_2 monotone Grammatiken.

Semi-Entscheidungsverfahren für Ungleichheit von $\mathcal{L}(G_1)$ und $\mathcal{L}(G_2)$:

- 1 Sei $n = 0$.
- 2 Vergleiche Wörter der Länge n in $\mathcal{L}(G_1)$ und $\mathcal{L}(G_2)$.
- 3 Wenn sie unterschiedlich sind: Stoppe.
- 4 Wenn sie gleich sind: Erhöhe n , weiter bei 2

Da G_1 und G_2 keine kürzenden Regeln enthalten, gibt es nur endlich viele Ableitungen für Wörter der Länge n . □

Aus der Inklusion der Sprachklassen der Chomsky-Hierarchie folgt:

Korollar 5.64

Die folgenden Probleme sind semi-entscheidbar:

- 1 Komplement des Leerheitsproblems für kontextsensitive Sprachen
- 2 Komplement des Äquivalenzproblems für kontextfreie Sprachen

Aus Korollar 5.64 und Satz 5.55 folgt:

Korollar 5.65

Die folgenden Probleme sind **nicht** semi-entscheidbar:

- 1 Leerheitsproblem für kontextsensitive Sprachen
- 2 Äquivalenzproblem für kontextfreie Sprachen

- \mathcal{U} kann jede TM \mathcal{M} simulieren
 - Eingabe: Codierung von \mathcal{M} und Wort w
 - Codierung von Konfigurationen
 - Suche nach Folgekonfigurationen
 - Test auf Akzeptanz
- WP, LP und ÄP sind unentscheidbar.
- Aber: WP und $\overline{\text{LP}}$ sind semi-entscheidbar.
 - Algorithmus terminiert zumindest für eine der möglichen Antworten.
 - Für die andere läuft er unendlich lange.
- Weder ÄP noch $\overline{\text{ÄP}}$ ist semi-entscheidbar.
 - Algorithmus liefert nie verlässlich eine Antwort.
 - Unentscheidbarkeit von ÄP ist schwerwiegender als die von WP und LP.

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
4. Turing-Maschinen
- 5. Entscheidbarkeit**
 - 5.1 Unentscheidbarkeit des speziellen Wortproblems
 - 5.2 Reduktionsbeweise
 - 5.3 Das PKP und weitere unentscheidbare Probleme
 - 5.4 Semi-Entscheidbarkeit
 - 5.5 Die universelle Turing-Maschine
 - 5.6 Abschlusseigenschaften**
6. Berechenbarkeit
7. Komplexität

Lemma 5.66

Eine Sprache L ist semi-entscheidbar gdw sie Turing-erkennbar ist.

Unterschied in Akzeptanzbedingung:

semi-entscheidbar TM hält

Turing-erkennbar TM erreicht akzeptierende Stopkonfiguration

Beweis.

- ⇒ Sei $\mathcal{M}_S = (Q, \Sigma, \Gamma, \Delta, q_0, F)$ ein Semi-Entscheidungsverfahren für L . Dann erkennt $\mathcal{M}_E = (Q, \Sigma, \Gamma, \Delta, q_0, Q)$ die Sprache L , denn jede Stopkonfiguration ist akzeptierend.
- ⇐ Sei \mathcal{M}_E eine TM, die L erkennt, und \mathcal{M}_{det} die entsprechende deterministische TM für L . Das Semi-Entscheidungsverfahren \mathcal{M}_S unterscheidet sich von \mathcal{M}_E dadurch, dass es von jeder nicht akzeptierenden Stopkonfiguration in eine Endlosschleife übergeht (z.B. mit Junk-Zustand). □

Satz 5.67 (Abschluss unter \neg)

Die Menge der rekursiv aufzählbaren Sprachen ist nicht abgeschlossen unter Komplement.

Beweis.

- WP ist semi-entscheidbar, damit Turing-erkennbar.
- Wären die Turing-erkennbaren Sprachen abgeschlossen unter Komplement, wäre auch \overline{WP} Turing-erkennbar und damit semi-entscheidbar.
- Wären WP und \overline{WP} semi-entscheidbar, wäre WP entscheidbar. $\frac{\perp}{\square}$

Satz 5.68 (Abschluss unter $\cup, \cdot, ^*, \cap$)

Die Menge der rekursiv aufzählbaren Sprachen ist abgeschlossen unter den Operationen Vereinigung, Konkatenation, Kleene-Stern und Durchschnitt.

Beweis.

Analog zu kontext-sensitiven Sprachen.

Z.B. Durchschnitt: Seien \mathcal{M}_1 und \mathcal{M}_2 (deterministische) Turingmaschinen für L_1 und L_2 . Dann entscheidet die TM \mathcal{M}_\cap die Sprache $L_1 \cap L_2$:

- \mathcal{M}_\cap kopiert die Eingabe w auf ein zusätzliches Band.
- \mathcal{M}_\cap verhält sich auf dem Eingabeband wie \mathcal{M}_1 .
- Wenn \mathcal{M}_1 akzeptiert, wechselt \mathcal{M}_\cap auf das zusätzliche Band und verhält sich wie \mathcal{M}_2 .

Offensichtlich akzeptiert \mathcal{M}_\cap eine Eingabe w gdw. sowohl \mathcal{M}_1 als auch \mathcal{M}_2 w akzeptieren.



Satz 5.69

Es gibt Turing-erkennbare Sprachen, die nicht kontextsensitiv sind.

Beweis.

Sei Σ ein Alphabet und

- (G_0, G_1, G_2, \dots) eine Aufzählung aller monotonen Grammatiken über Σ ,
- (w_0, w_1, w_2, \dots) eine Aufzählung aller Wörter über Σ .

(Diese Aufzählungen sind mit TM realisierbar.)

$$L := \{w_i \mid w_i \notin \mathcal{L}(G_i)\} \quad (\text{Diagonalisierung!})$$

L ist entscheidbar (Typ 0):

- Zähle G_0, G_1, \dots, G_i auf
- Zähle w_0, w_1, \dots, w_i auf
- Entscheide Wortproblem $(\mathcal{L}(G_i), w_i)$

L ist nicht kontextsensitiv (Typ 1):

- Sonst gibt es ein k mit $L = \mathcal{L}(G_k)$
- $w_k \in \mathcal{L}(G_k)$ gdw. $w_k \in L$ ($L = \mathcal{L}(G_k)$)
gdw. $w_k \notin \mathcal{L}(G_k)$ (Definition von L) \square

- Gödelisierung: Codierung einer TM als Eingabe für TM
- Turing: Unentscheidbarkeit von WP und HP (Diagonalisierung)
- Unentscheidbarkeitsbeweise durch Reduktion
- Rice: Alle interessanten Probleme für TM sind unentscheidbar
- Unentscheidbare Problem aus anderen Bereichen
 - PKP
 - kontextfreie und kontextsensitive Sprachen
 - Gültigkeit in der Prädikatenlogik
- Semi-Entscheidbarkeit und rekursive Aufzählbarkeit
 - Aufzähl-TM
 - WP und LP zumindest semi-entscheidbar
- Simulation von TM durch universelle TM \mathcal{U}
- Abschlusseigenschaften: Alles außer Komplement

Überblick Spracheigenschaften

Eigenschaft	regulär (Typ 3)	kontextfrei (Typ 2)	kontextsensitiv (Typ 1)	rekursiv aufzählbar (Typ 0)
Abschluss \cup , \cdot , $*$ \cap	● ● ●	● ● ●	● ● ●	● ● ●
Entscheidbarkeit				
Wortproblem	●	●	●	●
Leerheitsproblem	●	●	●	●
Äquivalenzproblem	●	●	●	●
Äquivalenz deterministisches und nicht-deterministisches Maschinenmodell	●	●	?	●

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
4. Turing-Maschinen
5. Entscheidbarkeit
- 6. Berechenbarkeit**
 - 6.1 Turing-Berechenbarkeit
 - 6.2 WHILE-Programme
 - 6.3 Die Church-Turing-These
7. Komplexität

Inhalt

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
4. Turing-Maschinen
5. Entscheidbarkeit
- 6. Berechenbarkeit**
 - 6.1 Turing-Berechenbarkeit
 - 6.2 WHILE-Programme
 - 6.3 Die Church-Turing-These
7. Komplexität

Bisher: TM erkennt Sprache

- Eingabe wird akzeptiert oder nicht akzeptiert (abhängig von F)
 - akzeptierende Stopkonfiguration
 - nicht akzeptierende Stopkonfiguration oder unendliche Berechnung
- Nur zwei mögliche Antworten

Jetzt: TM berechnet Funktion

- TM kann Band beschreiben \rightsquigarrow Ausgabe erzeugen
- nicht möglich / sinnvoll bei EA, KA und LBA
- Analog zu Funktionen in Programmiersprachen

- Berechnende TM ist immer deterministisch
 - Funktionswert sonst nicht eindeutig
- Eingabe: Auf (erstem) Band, Parameter getrennt durch \square
- Ausgabe: Inhalt des (ersten) Bandes, wenn TM stoppt (wie bei Aufzähl-TM)
 - von Kopfposition
 - bis zum ersten Symbol aus $\Gamma \setminus \Sigma$
- Endzustandsmenge irrelevant
- Unendliche Berechnung entspricht **nicht definiertem** Funktionswert
 - erlaubt **partielle** Funktionen

Definition 6.1 (Turing-berechenbar)

Eine partielle Funktion $f : (\Sigma^*)^n \rightarrow \Sigma^*$ heißt **Turing-berechenbar**, wenn es eine DTM \mathcal{M} gibt, so dass für jedes Tupel $(w_1, \dots, w_n) \in \text{dom}(f)$ mit $f(w_1, \dots, w_n) = x$ gilt:

- \mathcal{M} terminiert ausgehend von Startkonfiguration $q_0 w_1 \square w_2 \square \dots \square w_n$.
- Die Stopkonfiguration hat die Form $uqxvy$ mit
 - $u, y \in \Gamma^*$
 - $v \in \Gamma \setminus \Sigma$

Beispiel: Addition

- Mathematische Funktionen werden oft unär mit $\Sigma = \{a\}$ dargestellt
- a^k entspricht k

Beispiel 6.2

Die TM M_{add} berechnet die Summe zweier Zahlen

$$M_{add} = (\{0, 1, 2, 3\}, \{a\}, \{a, \square\}, \Delta, 0, \{\})$$

	Ausgangszustand	lesen	schreiben	bewegen	Folgezustand
$\Delta =$	0	a	a	r	0
	0	\square	a	l	1
	1	a	a	l	1
	1	\square	\square	r	2
	2	a	\square	r	3

Übung 6.3

Welche Berechnungen führt M_{add} mit den Eingaben aaa \square aa, \square aa, $\square\square$ durch?
Welche Ausgaben werden erzeugt?

Übung 6.4

Geben Sie eine TM \mathcal{M}_{sub} an, die die modifizierte Differenz ($\dot{-}$) berechnet.

$$\begin{array}{rcl} \dot{-} & : & \mathbb{N}^2 \rightarrow \mathbb{N} \\ x \dot{-} y & := & \begin{cases} x - y & \text{wenn } y \leq x \\ 0 & \text{sonst} \end{cases} \end{array}$$

Definition 6.5 (Charakteristische Funktion)

Für eine Trägermenge T und eine Teilmenge $S \subseteq T$ ist die **charakteristische Funktion** von S die Funktion $\chi_S : T \rightarrow \mathbb{B}$ mit

$$\chi_S(x) = \begin{cases} 1 & \text{wenn } x \in S \\ 0 & \text{sonst} \end{cases}$$

Satz 6.6

Eine Menge S ist entscheidbar gdw. χ_S berechenbar ist.

Beweis.

- ⇒ Wenn Eingabe akzeptiert wurde, schreibe a, sonst schreibe □, stoppe.
- ⇐ Wenn Stopkonfiguration erreicht ist, lies aktuellen Bandinhalt.
Falls Inhalt gleich a ist: akzeptiere, sonst verwirfe.
Da χ_S total ist, terminiert die Berechnung für jede Eingabe.



- Erweitert Turing-Maschine um Ausgabe
- Ermöglicht Berechnung von Funktionen
 - Eingabe-Tupel wird von Band gelesen
 - Ausgabe wird auf Band geschrieben
- nicht terminierende Berechnungen \rightsquigarrow partielle Funktionen
- Entscheidbarkeit als Spezialfall von Berechenbarkeit
 - charakteristische Funktion

Inhalt

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
4. Turing-Maschinen
5. Entscheidbarkeit
- 6. Berechenbarkeit**
 - 6.1 Turing-Berechenbarkeit
 - 6.2 WHILE-Programme**
 - 6.3 Die Church-Turing-These
7. Komplexität

- an imperative Programmiersprachen angelehntes Berechnungsmodell
 - gleich mächtig
 - auf notwendige Anweisungen reduziert
- gleich mächtig wie Turingmaschinen

Syntax-Elemente:

Variablen x_0, x_1, x_2, \dots

Konstanten 0, 1, 2, ...

Zuweisungsoperator :=

Kompositionsoperator ;

Arithmetische Operatoren +, −

Schlüsselwörter **while**, **do**, **end**

Definition 6.7 (Syntax von WHILE-Programmen)

Die Syntax von WHILE-Programmen ist wie folgt induktiv definiert:

- 1 **Wertzuweisung** Für $i, j, c \in \mathbb{N}$ sind

$$x_i := x_j + c \text{ und } x_i := x_j - c$$

WHILE-Programme.

- 2 **Komposition:** Sind P_1 und P_2 WHILE-Programme, dann auch

$$P_1; P_2$$

- 3 **While-Schleife:** Ist P ein WHILE-Programm und $i \in \mathbb{N}$, dann auch

while x_i **do** P **end**

Vorsicht bei arithmetischen Operationen / Zuweisung

Immer **eine Variable** und **eine Konstante!**

Übung 6.8

Geben Sie eine kontextfreie Grammatik für WHILE-Programme an.

Definition 6.9 (Semantik von WHILE-Programmen)

Bei einem WHILE-Programm mit n Eingabewerten

- enthalten die Variablen x_1, \dots, x_n die Eingabewerte;
- haben alle anderen Variablen den Wert 0;
- ist die Ausgabe der Inhalt der Variable x_0 nach Programmende.

Die Programmkonstrukte haben folgende Bedeutung:

1 $x_i := x_j + c \quad / \quad x_i := x_j - c$

Der neue Wert von x_i ist die Summe / die modifizierte Differenz des alten Werts von x_j und der Konstanten c .

2 $P_1; P_2$

Es wird zuerst P_1 , danach P_2 ausgeführt.

3 **while** x_i **do** P **end**

Es wird so lange P ausgeführt, bis x_i den Wert 0 erreicht.

Vorsicht bei While-Schleifen

Keine anderen Bedingungen ($x_1 \leq 3$ oder $x_1 > x_2$) möglich!

Definition 6.10 (WHILE-berechenbar)

Eine (partielle) Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ heißt WHILE-berechenbar, falls es ein WHILE-Programm P gibt, für das gilt:

Wenn P mit den Eingabewerten n_1, \dots, n_k in den Variablen x_1, \dots, x_k gestartet wird,

- terminiert P , falls $f(n_1, \dots, n_k)$ definiert ist, mit dem Wert von $f(n_1, \dots, n_k)$ in der Variablen x_0 ;
- sonst terminiert P nicht.

Alle Kontrollstrukturen aus anderen Programmiersprachen können mit WHILE-Programmen realisiert werden.

Beispiel 6.11 (LOOP-Schleife)

Loop x_i **do** P **end**: Schleife mit fester Anzahl x_i von Durchläufen

- 1: $x_j := x_i + 0;$
- 2: **while** x_j **do**
- 3: $x_j := x_j - 1;$
- 4: P
- 5: **end**

x_j wird sonst nicht benutzt

Beispiel 6.12 (IF-THEN-ELSE-Konstrukt)

if x_i **then** P_1 **else** P_2 **end**: Wenn $x_i \neq 0$ gilt, wird P_1 ausgeführt, sonst P_2

```
1: loop  $x_i$  do
2:      $x_j := x_l + 1$     //  $x_i \neq 0 \rightsquigarrow x_j = 1$ 
3: end;
4:  $x_k := x_l + 1$ ;
5: loop  $x_j$  do
6:      $x_k := x_l + 0$     //  $x_j \neq 0 \rightsquigarrow x_k = 0$ 
7: end;
8: loop  $x_j$  do
9:      $P_1$       //  $x_j \neq 0 \rightsquigarrow P_1$  wird ausgeführt
10: end;
11: loop  $x_k$  do
12:      $P_2$       //  $x_k \neq 0 \rightsquigarrow P_2$  wird ausgeführt
13: end
```

x_j, x_k, x_l werden sonst nicht benutzt

Beispiel 6.13 (Variablen-Addition)

Eingabe: x_1, x_2 : Summanden

Ausgabe: x_0 : Summe $x_1 + x_2$

```
1:  $x_0 := x_1 + 0;$ 
2: loop  $x_2$  do
3:    $x_0 := x_0 + 1$ 
4: end
```

Beispiel 6.14 (Multiplikation)

Eingabe: x_1, x_2 : Faktoren

Ausgabe: x_0 : Produkt $x_1 \cdot x_2$

```
1: loop  $x_1$  do
2:   loop  $x_2$  do
3:      $x_0 := x_0 + 1$ 
4:   end
5: end
```

Übung 6.15

Schreiben Sie WHILE-Programme für

- 1 die modifizierte Differenz \div
- 2 die Ganzzahl-Division DIV,
- 3 den Modulo-Operator MOD.

Es gilt z.B.

- $5 \div 3 = 2$,
- $3 \div 5 = 0$,
- $14 \text{ DIV } 3 = 4$,
- $14 \text{ MOD } 3 = 2$.

Sie können hierzu die bisher verwendeten Konstruktoren (Variablen-Addition, **loop**, **if**, ...) verwenden.

Beispiel 6.16 (Variablen-Subtraktion)

$$s : \mathbb{N}^2 \rightarrow \mathbb{N} \text{ mit } s(x_1, x_2) = \begin{cases} x_1 - x_2 & \text{wenn } x_1 \geq x_2 \\ \text{undefiniert} & \text{sonst} \end{cases}$$

Die Funktion s ist WHILE-berechenbar:

```
1: while  $x_2$  do // Solange  $x_2 > 0$  gilt
2:   if  $x_1$  then // Wenn  $x_1 > 0$  gilt
3:      $x_1 := x_1 - 1;$ 
4:      $x_2 := x_2 - 1$  // dekrementiere  $x_1$  und  $x_2$  parallel
5:   end
6: end; // terminiert nicht, wenn  $x_2 > x_1$  gilt
7:  $x_0 := x_1 + 0$ 
```

WHILE bietet

- Kontrollstruktur **while**
- Addition und Subtraktion für natürliche Zahlen

Damit sind simulierbar:

- Kontrollstrukturen FOR, IF, ...
- Gleitkommazahlen
- komplexe mathematische Operationen
- Operationen auf anderen Datentypen (Buchstaben, Wörter, Objekte,...)

Korollar 6.17

Alle Funktionen, die mit existierenden Programmiersprachen (Java, C, Python, Prolog, Lisp, ...) berechenbar sind, sind auch WHILE-berechenbar.

Satz 6.18

Jede WHILE-berechenbare Funktion ist auch Turing-berechenbar.

Beweis.

Simuliere WHILE-Programm P mit k EingabevARIABLEn und $l \geq k$ verwendeten Variablen durch $(l + 1)$ -Band-TM \mathcal{M}_P :

- 1 Eingabevektor $x_1 \square x_2 \square \dots \square x_k$ auf Band 1
- 2 Kopiere x_i auf Band $i + 1$ für $1 \leq i \leq k$; lösche Band 1
- 3 Simuliere einzelne Schritte:
 - $x_i := x_j + / \div c$: Führe Operation auf Band $i + 1$ aus
 - $P_1; P_2$: Simuliere erst P_1 , dann P_2
 - **while** x_i **do** P_1 **end**:
 - Wenn Band $i + 1$ leer ist, terminiere
 - Sonst simuliere P_1 , beginne von vorn
- 4 Spule Band 1 an den Anfang zurück (Ausgabevariable x_0)
(Wird nur ausgeführt, wenn P terminiert)



Satz 6.19

Jede Turing-berechenbare Funktion ist auch WHILE-berechenbar.

Beweis.

Simuliere TM $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_0, F)$ durch WHILE-Programm $P_{\mathcal{M}}$:

Sei $Q = \{q_0, \dots, q_n\}$ und $\Gamma = \{c_1, \dots, c_k\}$.

Codierung einer Konfiguration xq_iy durch 3 Variablen x_1, x_2, x_3 :

- $x_2 = i$ codiert Zustand
- x_1 und x_3 codieren x und y :
 - Symbol c_i codiert als i
 - Wort $w \in \Gamma^l$ codiert als Zahl zur Basis $k + 1$

- $x_1 = c_{i_1} c_{i_2} \dots c_{i_l} \mapsto i_1 \cdot k^{l-1} + i_2 \cdot k^{l-2} + \dots + i_{l-1} \cdot k^1 + i_l \cdot k^0$

(von links nach rechts)

- $x_3 = c_{i_1} c_{i_2} \dots c_{i_l} \mapsto i_l \cdot k^{l-1} + i_{l-1} \cdot k^{l-2} + \dots + i_2 \cdot k^1 + i_1 \cdot k^0$

(von rechts nach Links)

- z.B. $k = 9$, Konfiguration $c_3c_5c_4c_1q_5c_3c_9c_6$ ergibt $x_1 = 3541, x_2 = 5, x_3 = 693$

- Lese- und Schreiboperationen sowie Kopfbewegungen möglich mit DIV und MOD

Beweis (Fortsetzung).

Arbeitsweise von P_M :

- 1 Codiere Eingabe in Startkonfiguration
- 2 WHILE-Schleife:
 - 1 Decodiere aktuelles Symbol c (aus $x_3 \text{ MOD } k + 1$) und q aus x_2
 - 2 Prüfe, ob es eine Folgekonfiguration mit Zustand q und Symbol c gibt
 - ja: ändere x_1, x_2, x_3 entsprechend
 - nein: beende WHILE-Schleife
- 3 decodiere Ausgabe aus x_3 und schreibe Wert in x_0
(wird nur ausgeführt, wenn M terminiert)



- Sprachumfang: Variablen, $+$, \cdot , While-Schleife
- Andere Sprachkonstrukte simulierbar
- Gleich mächtig wie andere Programmiersprachen
- Gleich mächtig wie Turingmaschine

Inhalt

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
4. Turing-Maschinen
5. Entscheidbarkeit
- 6. Berechenbarkeit**
 - 6.1 Turing-Berechenbarkeit
 - 6.2 WHILE-Programme
 - 6.3 Die Church-Turing-These**
7. Komplexität

Wir haben gezeigt:

- Äquivalenz zwischen Turingmaschinen und WHILE-Programmen
- Äquivalenz zwischen While-Programmen und Java/C/Lisp/Prolog-Programmen

Man kann außerdem Äquivalenz zeigen zu

- μ -rekursiven Funktionen
- λ -Kalkül (Church)
- von-Neumann-Rechner
- realen Prozessoren/Computern



Alonzo Church
(1903–1995)

Ergebnis:

- Alle existierenden Berechnungsmodelle und Computer sind äquivalent zur TM (oder schwächer).

- A function is *effectively calculable* if its values can be found by some purely mechanical process.
- A *computable function* is a function calculable by a machine.
- The classes of effectively calculable functions and computable functions are identical.

Alan Turing, 1939

effectively calculable durch einen Algorithmus berechenbar
ohne Rückgriff auf Intuition oder Verständnis des Problems
computable durch eine Turingmaschine berechenbar

Turing: Das Halteproblem für TM ist unentscheidbar.

Rice: Jede nicht-triviale semantische Eigenschaft von TM ist unentscheidbar.

Church-Turing: Jeder existierende Computer und jedes Berechnungsmodell ist äquivalent zur TM (oder schwächer).

Konsequenzen

- Keine interessante Eigenschaft von Programmen in irgendeiner mächtigen Programmiersprache ist entscheidbar.
- Zahlreiche wichtige Probleme außerhalb der Informatik sind unentscheidbar.

Innerhalb der Informatik:

SW-Engineering Erfüllt das Programm eine formale Spezifikation?

Debugging Hat das Programm ein Speicherleck?

Malware Richtet das Programm Schaden an?

Studium Berechnet das Programm des Studenten dieselbe Funktion wie die Musterlösung?

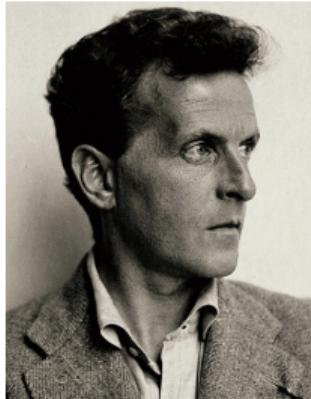
Außerhalb der Informatik:

Formale Sprachen Erzeugen zwei kontextfreie Grammatiken dieselbe Sprache?

Mathematik Hilberts zehntes Problem: Ganzzahlige Lösungen für Polynome mit mehreren Variablen

Logik Automatisierung von Beweisen

Nochmal: Ist das wichtig?



*It is very queer that this should have puzzled anyone. [...]
It doesn't matter.*

Wittgenstein

Yes, it does matter!

Church, Turing



Was bleibt möglich?

Es ist möglich

ein Programm P aus einer Sprache in eine andere zu übersetzen

festzustellen, ob ein Programm einen Befehl enthält, auf die Festplatte zu schreiben

zur Laufzeit zu prüfen, ob ein Programm auf die Festplatte zugreift

ein Programm zu schreiben, das in allen „wichtigen“ Fällen die richtige Antwort gibt

denn

ein spezielles Programm wird für P erzeugt.

dies ist eine syntaktische Eigenschaft. Unentscheidbar ist, ob dieser Befehl irgendwann ausgeführt wird.

dies entspricht einer Simulation durch \mathcal{U} . Es ist unentscheidbar, ob der Zugriff nie ausgeführt wird.

es wird immer Fälle geben, in denen gar keine oder eine falsche Antwort gegeben wird.

Was kann man tun?

Kann man die Turingmaschine „reparieren“?

- Der Unentscheidbarkeits-Beweis verwendet keine spezifischen Charakteristika der TM.
- Einzige Voraussetzung: Existenz der universellen TM \mathcal{U}
- Andere Maschinenmodelle haben dasselbe Problem (oder sind schwächer).
- Die TM ist nicht zu schwach, sondern **zu stark**.

Auswege:

- Wenn möglich: Schwächere Formalismen verwenden
 - für formale Sprachen: Reguläre Ausdrücke, kontextfreie Grammatiken
 - in der Logik: Modallogik oder Dynamische Logik statt Prädikatenlogik
- Heuristiken verwenden, die in vielen Fällen funktionieren;
verbleibende Fälle manuell lösen
 - prädikatenlogische Resolution mit Timeout

- TM kann Ausgaben erzeugen und Funktionen berechnen
- WHILE: Minimale Programmiersprache
 - äquivalent zu TM
 - äquivalent zu realen Programmiersprachen / Computern
- Church-Turing-These: TM ist äquivalent zu jedem mächtigen Berechnungsformalismus und Maschinenmodell
- Keine interessante Eigenschaft für irgendeine Programmiersprache ist entscheidbar
- Keine existierende oder zukünftige Programmiersprache kann die für TM unentscheidbaren Probleme entscheiden

Konsequenzen:

- Computer können Informatikern nicht alle Arbeiten abnehmen. ☹
- Computer werden Informatiker nie überflüssig machen. ☺

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
4. Turing-Maschinen
5. Entscheidbarkeit
6. Berechenbarkeit
- 7. Komplexität**
 - 7.1 Komplexitätsklassen
 - 7.2 NP-Vollständigkeit

Bisher: Problem ist entscheidbar / unentscheidbar / semi-entscheidbar

Jetzt: Feinere Analyse entscheidbarer Probleme

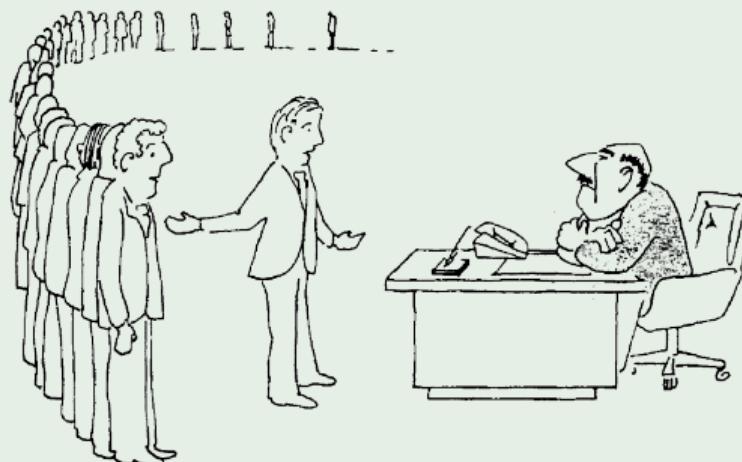
- Wieviel Zeit benötigt die Lösung des Problems?
konkret: Aus wievielen Schritten besteht die Berechnung?
- in Abhängigkeit von der Größe der Eingabe
konkret: Anzahl der durch das Eingabewort belegten Felder

Unterschiede zur Vorlesung „Algorithmen“

- Konkretes Berechnungsmodell Turing-Maschine
- Gesucht ist Komplexität des Problems
 - nicht konkreter Algorithmus
 - sondern bester Algorithmus zur Lösung des Problems
- auch untere Schranken werden untersucht

Beispiel 7.1 (aus M. Garey, D. Johnson: Computers and Intractability)

- Aufgabe: Entwickle effizienten Algorithmus für neues Produkt
- Problem: Sie finden keine effiziente Lösung, Entwicklung stockt
- Realistische Strategie:



„Ich finde keinen effizienten Algorithmus, genau wie diese berühmten Wissenschaftler.“

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
4. Turing-Maschinen
5. Entscheidbarkeit
6. Berechenbarkeit
- 7. Komplexität**
 - 7.1 Komplexitätsklassen
 - 7.2 NP-Vollständigkeit

Definition 7.2 (f -zeitbeschränkt)

Sei $f : \mathbb{N} \rightarrow \mathbb{N}$ eine Funktion. Eine Turingmaschine \mathcal{M} mit dem Alphabet Σ heißt **f -zeitbeschränkt**, wenn für jede Eingabe $w \in \Sigma^*$ jede Berechnung von \mathcal{M} höchstens $f(|w|)$ Schritte hat.

Wichtig

Beschreibt Komplexität einer speziellen Turingmaschine bzw. eines Algorithmus.

Beispiel 7.3

Die Turingmaschine \mathcal{M}_{abc} aus Beispiel 4.7 ist $(\frac{4}{9}n^2 + 1)$ -zeitbeschränkt.

Definition 7.4 (TIME und NTIME)

Sei $f : \mathbb{N} \rightarrow \mathbb{N}$ eine Funktion. Wir definieren:

$$\text{TIME}(f) = \{L \mid \text{es gibt } f\text{-zeitbeschränkte DTM } \mathcal{M} \text{ mit } \mathcal{L}(\mathcal{M}) = L\}$$

$$\text{NTIME}(f) = \{L \mid \text{es gibt } f\text{-zeitbeschränkte NTM } \mathcal{M} \text{ mit } \mathcal{L}(\mathcal{M}) = L\}$$

Wichtig

Beschreibt nicht die Komplexität eines speziellen Algorithmus, sondern des **Problems**, d.h. des besten Algorithmus.

Beispiel 7.5 (\mathcal{M}_{abc})

Die Sprache $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, die von der Turingmaschine \mathcal{M}_{abc} erkannt wird, ist in $\text{TIME}(\frac{4}{9}n^2 + 1)$.

Satz 7.6

Für jede Funktion f gilt: $\text{TIME}(f) \subseteq \text{NTIME}(f)$.

Definition 7.7 (P und NP)

$$\begin{aligned} P &:= \bigcup_{p \text{ ist Polynom in } n} \text{TIME}(p(n)) \\ NP &:= \bigcup_{p \text{ ist Polynom in } n} \text{NTIME}(p(n)) \end{aligned}$$

Beispiel 7.8

$\{a^n b^n c^n \mid n \in \mathbb{N}\} \in P$

Wichtig

- NP steht für nichtdeterministisch polynomiell, **nicht** für nicht polynomiell!
- Viele Probleme sind weder in P noch in NP:
 - das Äquivalenzproblem für reguläre Sprachen
 - das Wortproblem für kontextsensitive Sprachen

Aus Satz 7.6 folgt:

Korollar 7.9

$$P \subseteq NP$$

- P ist Klasse der effizient Lösbaren Probleme ([tractable](#))
- Probleme, die nicht in P sind: [intractable](#)
 - Viele wichtige Probleme sind in NP, aber wahrscheinlich nicht in P
 - Gilt P = NP, sind diese Probleme effizient lösbar
- Reduktionen zwischen Berechnungsmodellen sind polynomiell
 - Gleiche Komplexitätsklassen für TM, WHILE, Java, ...

Definition 7.10 (SAT)

Gegeben sei eine Codierung von AL-Formeln mit endlichem Alphabet Σ .
SAT $\subseteq \Sigma^*$ ist die Menge aller erfüllbaren aussagenlogischen Formeln.

Beispiel 7.11

$$(A \vee \neg B) \wedge (C \vee B) \in \text{SAT}; \quad (A \vee \neg B) \wedge \neg A \wedge B \notin \text{SAT}$$

Satz 7.12

SAT $\in \text{NP}$.

Beweis.

- 1 Erzeuge nichtdeterministisch eine Interpretation \mathcal{I} , d.h. eine Belegung der Aussagevariablen von φ mit Wahrheitswerten.
- 2 Werte $\varphi^{\mathcal{I}}$ (deterministisch und polynomiell) aus.
- 3 Akzeptiere genau dann, wenn $\varphi^{\mathcal{I}}$ wahr ist.



Cliquenproblem

Gegeben:

- Ungerichteter Graph $G = (V, E)$

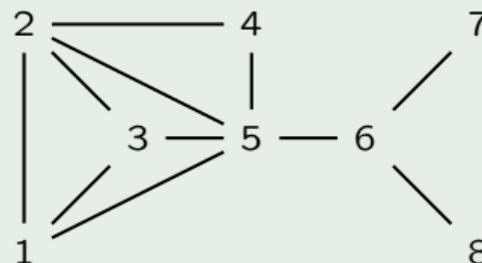
- $V = \{v_0, v_1, \dots, v_n\}$
- $E \subseteq V \times V$

- Natürliche Zahl k

Frage: Gibt es eine k -Clique in G , d.h. eine Teilmenge $V' \subseteq V$ mit

- $|V'| = k$ und
- für alle $(v_i, v_j) \in V' \times V'$ mit $v_i \neq v_j$ gilt $(v_i, v_j) \in E$.

Beispiel 7.13



Der Graph hat die 4-Clique $\{1, 2, 3, 5\}$, aber keine 5-Clique.

Definition 7.14 (CLIQUE)

Gegeben sei eine Codierung von Graphen mit endlichem Alphabet Σ .

CLIQUE ist die Menge aller Paare (G, k) , so dass $G = (V, E)$ ein Graph ist, der eine Clique der Größe k enthält.

Satz 7.15

CLIQUE $\in \text{NP}$.

Beweis.

- 1 Erzeuge nichtdeterministisch eine Menge $V' \subseteq V$ mit $|V'| = k$.
- 2 Teste (deterministisch und polynomiell), ob V' eine Clique ist.
- 3 Akzeptiere genau dann, wenn die Antwort „ja“ ist.



Kürzere Beschreibung des Algorithmus:

- 1 Rate eine Teilmenge V' .
- 2 Verifizierte, dass V' eine Clique ist.

Alle TM für Probleme in NP lassen sich beschreiben als

- 1 Nichtdeterministisch-polynomielles Raten einer Lösung L ;
- 2 Deterministisch-polynomielles Verifizieren, dass L eine Lösung ist.

Satz 7.16

Alle bekannten Berechnungsmodelle sind in polynomieller Zeit aufeinander reduzierbar.

- Komplexitätsklassen gelten auch für Java/C/...-Programme
- Komplexität der Reduktion zwischen Berechnungsmodellen ist maximal n^6 [HopcroftUllman]
 - Z.B. n^2 -zeitbeschränktes Java-Programm $\rightsquigarrow n^{12}$ -zeitbeschränkte TM

Korollar 7.17 (Erweiterte Church-Turing-These)

Für jedes realistische („effectively calculable“) Berechnungsmodell B gibt es ein Polynom $p : \mathbb{N} \rightarrow \mathbb{N}$ so dass jeder f -zeitbeschränkte Algorithmus in B übersetzt werden kann in eine $p(f)$ -zeitbeschränkte („computable“) Turing-Maschine und umgekehrt.

Mögliche Ausnahme in Zukunft: Quantencomputer

f -zeitbeschränkt Zeitbedarf einer Turingmaschine

~> Komplexität eines Algorithmus

(N)TIME(f) Probleme, für die von f -zeitbeschränkter (D)TM entschieden werden

~> Komplexität eines Problems

P, NP Probleme, die von polynomiell zeitbeschränkter (D)TM entschieden werden

Übertragbar von TM auf reale Computer / Programmiersprachen

Probleme in P

- Wortproblem für reguläre Sprachen
- Leerheitsproblem für reguläre Sprachen
- Wortproblem für kontextfreie Sprachen

Probleme in NP

- SAT
- CLIQUE

Inhalt

1. Einführung
2. Reguläre Sprachen und endliche Automaten
3. Chomsky-Grammatiken und kontextfreie Sprachen
4. Turing-Maschinen
5. Entscheidbarkeit
6. Berechenbarkeit
- 7. Komplexität**
 - 7.1 Komplexitätsklassen
 - 7.2 NP-Vollständigkeit

- NP enthält
 - Probleme, für die es keinen bekannten effizienten Algorithmus gibt wie SAT
 - einfachere Probleme wie $\{w \mid |w| \text{ ist gerade}\}$
- Welche sind die „echten“ NP-Probleme?
 - NP \ P? Nicht sinnvoll:
 - Für kein Problem $L \in NP$ konnte gezeigt werden, dass $L \notin P$ gilt.
 - Möglicherweise gilt $P = NP$.

NP-vollständig L ist mindestens so schwer wie jedes Problem in NP.

- Jedes Problem in NP lässt sich mit **polynomiellem Zeitaufwand** auf L reduzieren.

Definition 7.18 (Polynomialzeit-Reduktion)

Eine Reduktion $f : L_1 \rightarrow L_2$ heißt **Polynomialzeit-Reduktion**, wenn es ein Polynom p und eine p -zeitbeschränkte DTM gibt, die f berechnet.

L_1 heißt dann **polynomialzeit-reduzierbar** auf L_2 ($L_1 \leq_p L_2$).

Definition 7.19 (NP-hart, NP-vollständig)

Ein Problem L heißt **NP-hart** (oder NP-schwer), wenn für alle $L' \in \text{NP}$ gilt: $L' \leq_p L$.

Ein Problem L heißt **NP-vollständig**, wenn es in NP und NP-hart ist.

Lemma 7.20

Für jedes NP-vollständige Problem L gilt: Aus $L \in \text{P}$ folgt $\text{P} = \text{NP}$.

Beweis.

Sei $L \in \text{P}$ NP-vollständig. Zu zeigen: Für alle $L' \in \text{NP}$ gilt $L' \in \text{P}$.

Da L NP-vollständig ist, gilt $L' \leq_p L$. Es gibt also ein Polynom p , so dass eine Instanz w' von L' in Zeit $p(|w'|)$ in eine entsprechende Instanz w von L übersetzt werden kann.

Zeitbedarf: $p(|w'|)$.

Da $L \in \text{P}$ gilt, gibt es ein Polynom q , so dass für w in Zeit $q(|w|)$ entschieden werden kann, ob $w \in L$ gilt. Die maximale Länge von w ist $p(|w'|)$. Zeitbedarf: $q(p(|w'|))$.

Der Gesamt-Zeitbedarf $p(|w'|) + q(p(|w'|))$ ist ein Polynom. □



Stephen Cook
(*1939)

Satz 7.21 (Cook (1971) / Levin (1973))

SAT ist NP-vollständig.

Beweis.

SAT \in NP: ✓

Zu zeigen: Jedes Problem L in NP ist polynomiell auf SAT reduzierbar.

$L \in$ NP: Es gibt p -zeitbeschränkte NTM \mathcal{M} , die L entscheidet. (Polynom p)

Ansatz: Formuliere \mathcal{M} und Eingabe w als AL-Formel φ_w mit

- φ_w ist erfüllbar gdw. \mathcal{M} akzeptiert w ;
- φ_w kann in polynomieller Zeit berechnet werden.



Leonid Levin
(*1948)

Beweis (Fortsetzung).

- Gegeben: Polynom p ; p -zeitbeschränkte NTM $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_0, F)$; Eingabe w für \mathcal{M}
- Gesucht: AL-Formel φ_w , die erfüllbar ist gdw. \mathcal{M} akzeptiert w .
- Aussagenvariablen:

$B_{t,i,a}$ Zur Zeit t steht auf Feld i Symbol a

$K_{t,i}$ Zur Zeit t steht der Kopf auf Position i

$Q_{t,q}$ Zur Zeit t ist \mathcal{M} im Zustand q

Werte für q und a sind konstant; Werte für t und i polynomiell.

$$\varphi_w = \varphi_{\text{ini}} \wedge \varphi_{\text{trans}} \wedge \varphi_{\text{gleich}} \wedge \varphi_{\text{akz}} \wedge \varphi_{\text{hilf}}$$

Formel	Beispiel	Beispielformel
φ_{ini}	$w = ab$	$Q_{0,0} \wedge K_{0,0} \wedge B_{0,0,a} \wedge B_{0,1,b} \wedge B_{0,2,\square} \wedge \dots$
φ_{trans}	$(0, a, b, r, 3)$ $(0, a, a, n, 1)$	$Q_{0,0} \wedge K_{0,0} \wedge B_{0,0,a} \rightarrow (Q_{1,3} \wedge K_{1,1} \wedge B_{1,0,b}) \vee (Q_{1,1} \wedge K_{1,0} \wedge B_{1,0,a})$
φ_{gleich}		$\neg K_{0,1} \wedge B_{0,1,b} \rightarrow B_{1,1,b} \wedge \dots$
φ_{akz}	$F = \{3, 4\}$	$Q_{p(w),3} \vee Q_{p(w),4}$
φ_{hilf}		$\neg(Q_{0,0} \wedge Q_{0,1}) \wedge \dots \wedge \neg(B_{0,0,a} \wedge B_{0,0,b}) \wedge \dots \wedge \neg(K_{0,0} \wedge K_{0,1}) \wedge \dots$



Lemma 7.22

Gilt $L_1 \leq_p L_2$ und $L_2 \leq_p L_3$, dann auch $L_1 \leq_p L_3$.

Beweis.

Sei p das Polynom der Reduktion von L_1 auf L_2 und q dasjenige der Reduktion von L_2 auf L_3 . Dann ist die Verkettung von q und p eine Polynomialzeitreduktion von L_1 auf L_3 . \square

Lemma 7.23

- 1 Wenn $L_1 \leq_p L_2$ gilt und $L_2 \in \text{NP}$ ist, gilt auch $L_1 \in \text{NP}$.
- 2 Wenn $L_1 \leq_p L_2$ gilt und L_1 NP-hart ist, ist auch L_2 NP-hart.

Beweis.

- 1 Eine TM \mathcal{M}_1 für L_1 ergibt sich aus der Verkettung einer TM für die Reduktionsfunktion mit der TM \mathcal{M}_2 für L_2 .
- 2 Da L_1 NP-hart ist, gilt für jedes $L \in \text{NP}$: $L \leq_p L_1$.
Mit Lemma 7.22 folgt aus $L \leq_p L_1 \leq_p L_2$ dann $L \leq_p L_2$. \square

Warum ist es wichtig, dass die Reduktionen nur polynomielle Zeit benötigen?

Wir wissen:

- L_1 ist NP-hart.
- $L_1 \leq L_2$ mit Reduktionsfunktion f .

Wir wollen zeigen: L_2 ist NP-hart.

Wenn f polynomiell ist:

- Man kann jedes Problem $L \in \text{NP}$ polynomiell auf L_1 reduzieren.
 - Man kann L_1 polynomiell auf L_2 reduzieren
- ⇒ Man kann jedes $L \in \text{NP}$ polynomiell auf L_2 reduzieren (Lemma 7.23). ✓

Wenn f nicht polynomiell ist:

- Man kann jedes Problem $L \in \text{NP}$ polynomiell auf L_1 reduzieren.
 - Man kann L_1 irgendwie (z.B. exponentiell) auf L_2 reduzieren
- ⇒ Man kann nicht jedes $L \in \text{NP}$ polynomiell auf L_2 reduzieren. ✗

Transformation in Disjunktive Normalform

Definition 7.24 (Disjunktive Normalform)

Eine aussagenlogische Formel ist in [Disjunktiver Normalform](#) (DNF), wenn sie eine Disjunktion von Konjunktionen von Literalen ist.

[DNF-SAT](#) ist das Erfüllbarkeitsproblem für DNF-Formeln, also die Menge aller erfüllbaren Formeln in DNF.

Beispiel 7.25

Die Formel $(A \wedge \neg B \wedge \neg C) \vee (\neg A \wedge A \wedge C) \vee B \vee (\neg B \wedge C \wedge \neg C)$ ist in DNF.

Satz 7.26

Jede aussagenlogische Formel kann in eine äquivalente Formel in DNF übersetzt werden.

Beweis.

Transformation ähnlich wie für KNF.

Unterschied: Das Distributivgesetz wird nur angewendet, wenn [Disjunktionen innerhalb von Konjunktionen](#) vorkommen.



Algorithmus zur Transformation in DNF

1	a	Elimination von \leftrightarrow	$\varphi \leftrightarrow \psi$	$\rightsquigarrow (\varphi \wedge \psi) \vee (\neg\varphi \wedge \neg\psi)$
	b	Elimination von \rightarrow	$\varphi \rightarrow \psi$	$\rightsquigarrow \neg\varphi \vee \psi$
2		De-Morgan-Regeln	$\neg(\varphi \vee \psi)$	$\rightsquigarrow \neg\varphi \wedge \neg\psi$
			$\neg(\varphi \wedge \psi)$	$\rightsquigarrow \neg\varphi \vee \neg\psi$
3		Elimination doppelter Negation	$\neg\neg\varphi$	$\rightsquigarrow \varphi$
4		Distributivgesetz	$\varphi \wedge (\psi \vee \chi)$	$\rightsquigarrow (\varphi \wedge \psi) \vee (\varphi \wedge \chi)$
5		Assoziativgesetz	$\varphi \wedge (\psi \wedge \chi)$	$\rightsquigarrow \varphi \wedge \psi \wedge \chi$
			$\varphi \vee (\psi \vee \chi)$	$\rightsquigarrow \varphi \vee \psi \vee \chi$

Beispiel 7.27

$$\begin{aligned} & ((A \vee \neg B) \rightarrow C) \wedge (A \leftrightarrow \neg C) \\ \equiv & (\neg(A \vee \neg B) \vee C) \wedge ((A \wedge \neg C) \vee (\neg A \wedge \neg \neg C)) \quad 1 \\ \equiv & ((\neg A \wedge \neg \neg B) \vee C) \wedge ((A \wedge \neg C) \vee (\neg A \wedge \neg \neg C)) \quad 2 \\ \equiv & ((\neg A \wedge B) \vee C) \wedge ((A \wedge \neg C) \vee (\neg A \wedge C)) \quad 3 \\ \equiv & ((\neg A \wedge B) \wedge (A \wedge \neg C)) \vee ((\neg A \wedge B) \wedge (\neg A \wedge C)) \vee \\ & \quad (C \wedge (A \wedge \neg C)) \vee (C \wedge (\neg A \wedge C)) \quad 4 \\ \equiv & (\neg A \wedge B \wedge A \wedge \neg C) \vee (\neg A \wedge B \wedge \neg A \wedge C) \vee \\ & \quad (C \wedge A \wedge \neg C) \vee (C \wedge \neg A \wedge C) \quad 5 \end{aligned}$$

Komplexität der DNF-Transformation

- Elimination von \leftrightarrow und Distributivgesetz **verdoppeln** Teilformeln
 - $A \leftrightarrow B \rightsquigarrow (A \wedge B) \vee (\neg A \wedge \neg B)$
 - $A \wedge (B \vee C) \rightsquigarrow (A \wedge B) \vee (A \wedge C)$
- Worst case: **exponentiell**
 - DNF von $A \leftrightarrow (B \leftrightarrow (C \leftrightarrow (D \wedge (E \vee F))))$ enthält 76 Literale
(16 Konjunktionen mit je 4–5 Konjunkten)
- Konsequenzen:
 - die Reduktion SAT \leq DNF-SAT ist **nicht** polynomiell.
 - DNF-SAT ist **nicht** NP-hart.
- DNF-SAT ist sogar in **linearer** Zeit entscheidbar, also in P.
- Die eigentliche Arbeit leistet die Reduktionsfunktion, nicht der Entscheidungsalgorithmus für DNF-SAT.

Übung 7.28

Finden Sie einen polynomiellen Entscheidungsalgorithmus für DNF-SAT.

Testen Sie damit die Erfüllbarkeit der Formel aus Beispiel 7.27:

$$(\neg A \wedge B \wedge A \wedge \neg C) \vee (\neg A \wedge B \wedge \neg A \wedge C) \vee (C \wedge A \wedge \neg C) \vee (C \wedge \neg A \wedge C)$$

Definition 7.29 (3-SAT)

- 1 Eine **3-Klausel** ist eine aussagenlogische Disjunktion mit höchstens 3 Literalen.
- 2 Eine Formel ist in **3-KNF**, wenn sie eine endliche Konjunktion von 3-Klauseln ist.
- 3 **3-SAT** ist die Menge aller erfüllbaren Formeln in 3-KNF.

Beispiel 7.30

- 1 $A \vee \neg B \vee C$ und $\neg X_1 \vee \neg X_3 \vee \neg X_{17}$ sind 3-Klauseln.
- 2 $\varphi = (A \vee B \vee C) \wedge (\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee \neg B \vee C) \wedge (A \vee B \vee \neg C)$ ist in 3-KNF.
- 3 $\varphi \in \text{3-SAT}$, denn die Belegung $\{A \mapsto 1, B \mapsto 0, C \mapsto 0\}$ macht φ wahr.

Satz 7.31

3-SAT ist NP-vollständig.

Beweis.

- 1 3-SAT \in NP: Folgt aus SAT \in NP. ✓
- 2 3-SAT ist NP-hart: Reduktion von SAT auf 3-SAT.

Verwendet Funktion f , die aus einer beliebigen AL-Formel φ eine 3-Formel $f(\varphi)$ erzeugt mit

- φ und $f(\varphi)$ sind erfüllbarkeitsäquivalent und
- f ist in polynomieller Zeit berechenbar.

□

Schwierigkeit: Die aus der Logik-Vorlesung bekannte Transformation in KNF hat wie die DNF-Transformation worst-case-exponentielle Laufzeit, kann also nicht verwendet werden.

Algorithmus zur Transformation in 3-KNF

- 1 Führe für jede Teilformel eine **neue Aussagenvariable** als Abkürzung ein
- 2 Ersetze Teilformeln durch Abkürzungen
- 3 Transformiere Konjunkte nach dem bekannten Verfahren in KNF
- 4 Bilde Konjunktion der abgekürzten Formeln und der Variable für die Gesamtformel
 - Teilformeln durch Abkürzungen ersetzt \rightsquigarrow nur Aussagenvariable wird verdoppelt
 - kein exponentieller Blow-up
 - polynomielle Laufzeit

Beispiel 7.32

$$((A \vee \neg B) \stackrel{3}{\rightarrow} C) \stackrel{2}{\wedge} \neg(A \stackrel{1}{\leftrightarrow} \neg C)$$

- $X_3 \leftrightarrow A \vee \neg B$
- $X_2 \leftrightarrow (X_3 \rightarrow C)$
- $X_1 \leftrightarrow (A \leftrightarrow \neg C)$
- $X_0 \leftrightarrow X_2 \wedge \neg X_1$

$$(X_3 \leftrightarrow A \vee \neg B) \wedge (X_2 \leftrightarrow (X_3 \rightarrow C)) \wedge (X_1 \leftrightarrow (A \leftrightarrow \neg C)) \wedge (X_0 \leftrightarrow X_2 \wedge \neg X_1) \wedge X_0$$

Übung 7.33

Transformieren Sie die Formel

$$(X_3 \leftrightarrow A \vee \neg B) \wedge (X_2 \leftrightarrow (X_3 \rightarrow C)) \wedge (X_1 \leftrightarrow (A \leftrightarrow \neg C)) \wedge (X_0 \leftrightarrow X_2 \wedge \neg X_1) \wedge X_0$$

in 3-KNF. Sie können hierbei jedes Konjunkt separat transformieren.

Algorithmus:

1	a	Elimination von \leftrightarrow	$\varphi \leftrightarrow \psi$	$\rightsquigarrow (\neg\varphi \vee \psi) \wedge (\varphi \vee \neg\psi)$
	b	Elimination von \rightarrow	$\varphi \rightarrow \psi$	$\rightsquigarrow \neg\varphi \vee \psi$
2		De-Morgan-Regeln	$\neg(\varphi \vee \psi)$	$\rightsquigarrow \neg\varphi \wedge \neg\psi$
			$\neg(\varphi \wedge \psi)$	$\rightsquigarrow \neg\varphi \vee \neg\psi$
3		Elimination doppelter Negation	$\neg\neg\varphi$	$\rightsquigarrow \varphi$
4		Distributivgesetz	$\varphi \vee (\psi \wedge \chi)$	$\rightsquigarrow (\varphi \vee \psi) \wedge (\varphi \vee \chi)$
5		Assoziativgesetz	$\varphi \wedge (\psi \wedge \chi)$	$\rightsquigarrow \varphi \wedge \psi \wedge \chi$
			$\varphi \vee (\psi \vee \chi)$	$\rightsquigarrow \varphi \vee \psi \vee \chi$

Satz 7.34

CLIQUE ist NP-vollständig.

Beweis.

Um NP-Härte zu zeigen, reduzieren wir 3-SAT in polynomieller Zeit auf CLIQUE.

Sei φ eine 3-Formel mit m 3-Klauseln, also

$$\varphi = (L_{1,1} \vee L_{1,2} \vee L_{1,3}) \wedge (L_{2,1} \vee L_{2,2} \vee L_{2,3}) \wedge \dots \wedge (L_{m,1} \vee L_{m,2} \vee L_{m,3})$$

Wir definieren die zugehörige Instanz (G, k) von CLIQUE wie folgt:

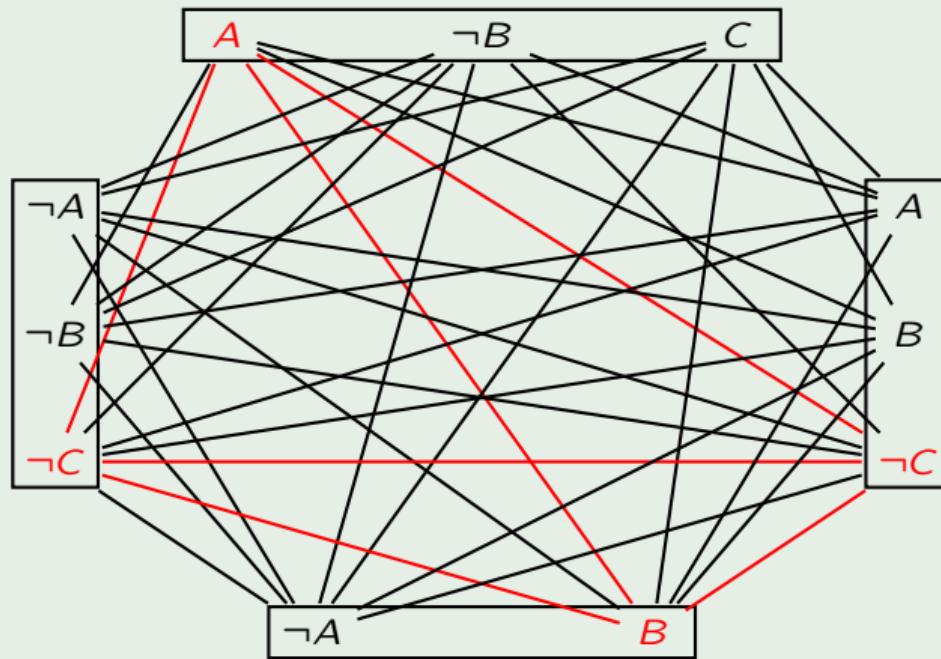
- $G = (V, E)$ mit
 - $V = \{L_{i,j} \mid L_{i,j} \text{ kommt in } \varphi \text{ vor}\}$, d.h. ein Knoten für jedes Literal in jeder Klausel;
 - $E = \{(L_{i,j}, L_{i',j'}) \mid i \neq i' \text{ und } L_{i,j} \neq \overline{L_{i',j'}}\}$,
d.h. alle Paare von Literalen aus unterschiedlichen Klauseln, die nicht komplementär sind.
- $k = m$, d.h. die Größe der Clique ist gleich der Anzahl der Klauseln.
 G hat eine Clique der Größe k
 - gdw. es gibt in G k Knoten, die paarweise miteinander verbunden sind
 - gdw. es gibt k Literale (eins aus jeder Klausel), die nicht komplementär sind
 - gdw. es gibt eine Interpretation für φ , die alle Klauseln wahr macht.



Beispiel: Reduktion 3-SAT auf CLIQUE

Beispiel 7.35

Sei $\varphi = (A \vee \neg B \vee C) \wedge (\neg A \vee \neg B \vee \neg C) \wedge (A \vee B \vee \neg C) \wedge (\neg A \vee B)$



Gibt es eine 4-Clique?

Modell für φ :
 $\{A \mapsto 1, B \mapsto 1, C \mapsto 0\}$

Übung 7.36

Gegeben sei die 3-Formel

$$\varphi = (A \vee \neg B \vee C) \wedge (\neg A \vee B \vee D) \wedge (\neg A \vee \neg C \vee D)$$

- 1 Erzeugen Sie wie in Beispiel 7.35 aus φ einen Graphen G .
- 2 Stellen Sie fest, ob G eine 3-Clique enthält, und geben Sie, wenn möglich, ein Modell für φ an.

Hamilton-Kreis

- Gegeben: Graph G
- Frage: Gibt es einen zusammenhängenden Pfad in G , der **jeden Knoten genau einmal** besucht?

aber: Euler-Kreis

- Gegeben: Graph G
- Frage: Gibt es einen geschlossenen Pfad in G , der **jede Kante genau einmal** durchläuft?
- In **linearer Zeit** entscheidbar: Teste, ob G zusammenhängend ist und jeder Knoten geraden Grad hat.

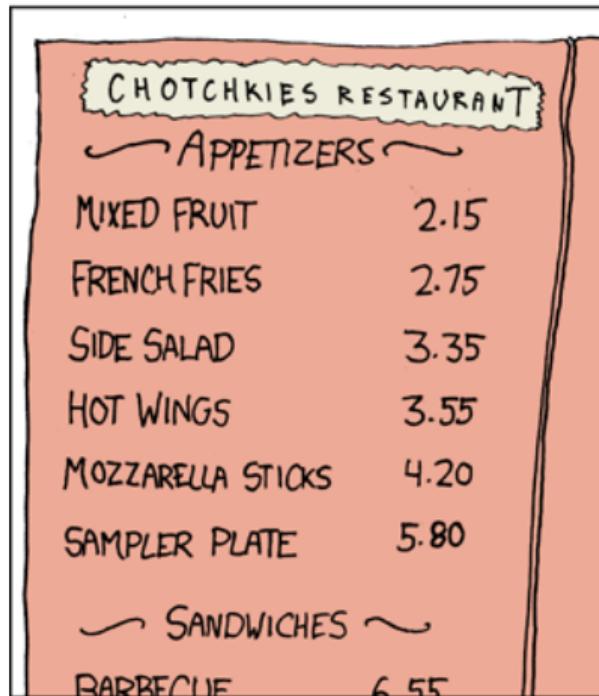
TSP

- Gegeben: Gewichteter Graph G , Maximalwert k
- Frage: Ist es möglich, alle Knoten in G zu besuchen, so dass das Gesamtgewicht der verwendeten Kanten unter k bleibt?

KNAPSACK

- Gegeben: n Objekte mit Größen g_1, \dots, g_n und Werten w_1, \dots, w_n ; Rucksack der Kapazität k ; Gesamtwert m
- Frage: Ist es möglich, im Rucksack Objekte im Wert von mindestens m unterzubringen, so dass die Gesamtgröße k nicht überschreitet?

MY HOBBY: EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS



- Probleme, die **in NP** und **mindestens so schwer** wie jedes Problem in NP sind
- Polynomialzeit-Reduktionen
- Cook/Levin: **SAT** ist NP-vollständig
 - elementar: Übersetzung von TM-Berechnungen in Aussagenlogik
- Durch Reduktionen: 3-SAT, CLIQUE, HAMILTON, TSP, KNAPSACK, ...
- Gibt es ein NP-vollständiges Problem, das zu P gehört, folgt **P = NP**.

Komplexität der behandelten Entscheidungsprobleme

Sprachklasse	Wortproblem	Leerheitsproblem	Äquivalenzproblem
regulär (Typ 3)	Linear $\text{DTIME}(n)$	Linear $\text{DTIME}(n)$	PSPACE -vollständig
kontextfrei (Typ 2)	polynomiell $\text{DTIME}(n^3)$ für CNF ¹	polynomiell $\text{DTIME}(n)$ für CNF	Komplement semi-entscheidbar
kontextsensitiv (Typ 1)	PSPACE -vollständig	Komplement semi-entscheidbar	Komplement semi-entscheidbar
rekursiv aufzählbar (Typ 0)	semi-entscheidbar	Komplement semi-entscheidbar	nicht semi-entscheidbar, Komplement nicht semi-entscheidbar

¹Transformation in CNF: $\text{DSPACE}(n^2)$ [HopcroftUllman]

Weitere Komplexitätsklassen

Es gibt viele weitere Komplexitätsklassen...

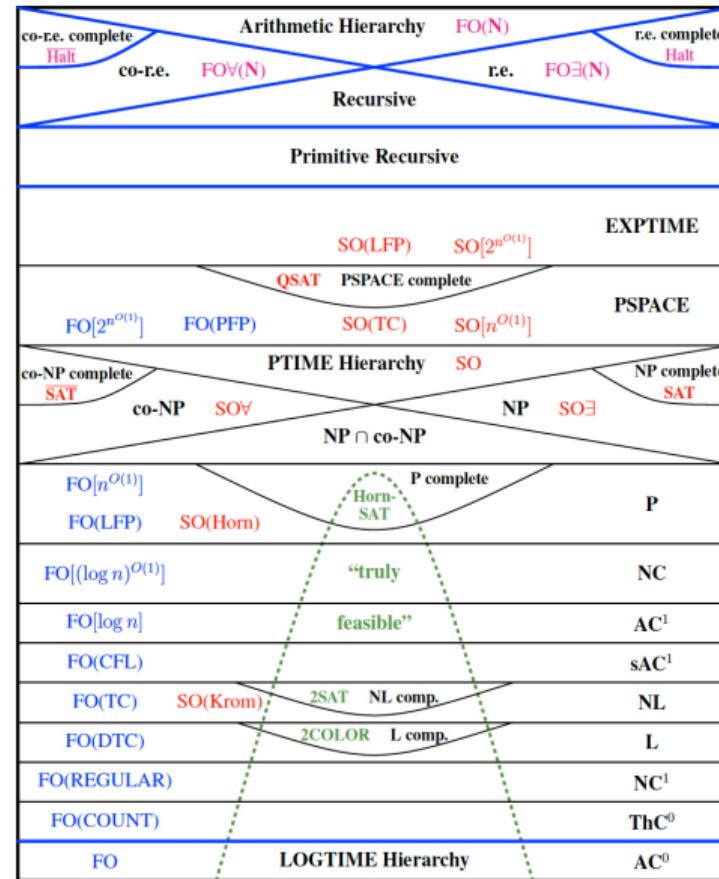


Abbildung:
Neil Immerman

■ Reguläre Sprachen und endliche Automaten

- abgeschlossen unter $\cup, \cap, ., ^*$,
- alle Probleme entscheidbar; Wort-, Leerheits- und Endlichkeitsproblem polynomiell
- effizient nutzbar
- begrenzte Ausdrucksstärke

■ kontextfreie Sprachen und Kellerautomaten

- verschachtelte Strukturen
- relevant für natürliche Sprachen und Programmiersprachen
- Wortproblem (CYK) und Leerheitsproblem effizient entscheidbar
- Äquivalenzproblem unentscheidbar
- nicht abgeschlossen unter $\cap,$

■ kontextsensitive Sprachen und monotone Grammatiken

- Grammatik: keine kürzenden Regeln
- LBA: Bandlänge auf Eingabe beschränkt
- schwer zu handhaben
- nur Wortproblem entscheidbar (PSPACE-vollständig)
- Leerheits- und Äquivalenzproblem unentscheidbar
- aber: abgeschlossen unter $\cup, \cap, ., ^*,$

- Turing-Maschinen
 - Varianten: mehrband, nichtdeterministisch, Aufzähl-TM, ...
 - gleich mächtig wie Computer
 - Church/Turing: gleich mächtig wie alle denkbaren Berechnungsmodelle
 - nicht abgeschlossen unter Komplement
 - Universelle Turingmaschine \mathcal{U}
- Entscheidbarkeit und Berechenbarkeit
 - Diagonalisierung
 - Reduktionsbeweise
 - Rice: alle interessanten Probleme für TM sind unentscheidbar
 - PKP und Gültigkeitsproblem der Prädikatenlogik sind unentscheidbar
 - Semi-Entscheidbarkeit und rekursive Aufzählbarkeit
 - WHILE-Programme und TM mit Ausgabe
- Komplexität
 - Komplexitätsklassen P, NP
 - „echt schwierig“: NP-vollständig
 - SAT (Erfüllbarkeitsproblem der Aussagenlogik)
 - CLIQUE
 - TSP, KNAPSACK, HAMILTON
 - Die Eine-Million-Dollar-Frage: $P \stackrel{?}{=} NP$