

Einführung in die Theoretische Informatik

Tobias Nipkow + Helmut Seidl + Javier Esparza

Fakultät für Informatik
TU München

Sommersemester 2022

© T. Nipkow / H. Seidl / J. Esparza / J. Křetínský

Version vom 11. Juli 2022

Kapitel I Organisatorisches

Siehe <https://www.in.tum.de/i07/lehre/ss22/theo/>

Vorkenntnisse und weiterführende Vorlesungen

- Vorkenntnisse:
 - Einführung in die Informatik 1
 - Diskrete Strukturen
- Weiterführende Vorlesungen:
 - Automaten und formale Sprachen
 - Komplexitätstheorie
 - Logik
 - Model Checking
 - Compilerbau
 - ...

Inhalt und Gliederung der Vorlesung

- Endliche Automaten und reguläre Ausdrücke
 - Grundlagen der Textanalyse, der lexikalischen Analyse von Programmiersprachen, und der Spezifikation und Analyse von Kommunikationsprotokollen.
- Kontextfreie Grammatiken
 - Grundlagen der syntaktischen Analyse von Programmiersprachen, Parsing, Compilerbau.
- Theorie der Berechenbarkeit
 - Untersuchung der Grenzen, was Rechner prinzipiell können.
- Komplexitätstheorie
 - Untersuchung der Grenzen, was Rechner mit begrenzten Ressourcen können.

Literatur



John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman.
Introduction to Automata Theory, Languages, and Computation.



Dexter Kozen.
Automata and Computability.



Michael Sipser.
Introduction to the Theory of Computation.



Katrin Erk, Lutz Priese.
Theoretische Informatik: Eine umfassende Einführung.



Uwe Schöning.
Theoretische Informatik — kurzgefasst.

Kapitel II Formale Sprachen

1. Einleitung: Kommunikation mit Rechnern

Kapitel II Formale Sprachen

1. Einleitung: Kommunikation mit Rechnern



Abbildung: Rechenmaschinen für arithmetische Operationen, XVII-XX Jh.
(Addition, Substraktion, Multiplikation, Division)
(By Ezrdr (Own work), via Wikipedia)



Abbildung: Gezeitenrechenmaschine im Deutschen Museum, gebaut 1935-39. (Quelle: Deutsches Museum)

Für die Berechnung von Funktionen der Gestalt $f(t) = \sum_{i=1}^{34} a_i \cos w_i t$

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHIEDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

6. *The universal computing machine.*

It is possible to invent a single machine which can be used to compute any computable sequence. If this machine \mathcal{U} is supplied with a tape on

Abbildung: Alan Turings Artikel (1936), in dem er eine (abstrakte) universelle Rechenmaschine beschreibt

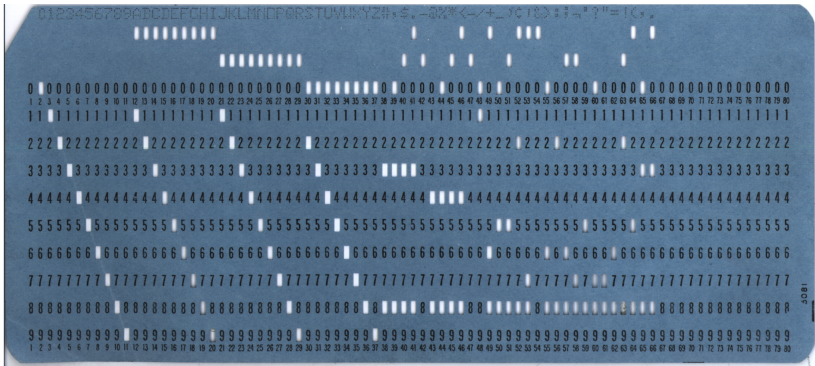


Abbildung: Programmieren mit Lochkarten in den 1960er Jahre

- In den 1950er setzt sich die Idee durch,

Programmiersprachen

für die Kommunikation mit Maschinen zu entwickeln.

- Gleichzeitig entwickelt der Linguist Noam Chomsky eine neue Theorie (**Transformationgrammatik**) der natürlichen Sprachen, mit dem Ziel, die Mechanismen zu erklären, die Menschen verwenden, um Sätze zu bauen.

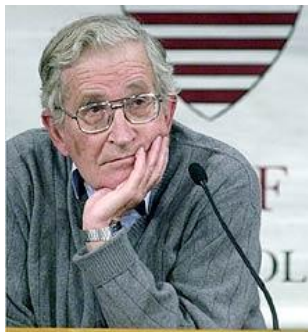


Noam Chomsky.

Three Models for the Description of Language.

Transactions on Information Theory, 113-124, 1956.

- Sein Begriff der **formalen Grammatiken** wird 1960 von dem ALGOL 60 Komitee übernommen (ALGOL = ALGOritmic Language)
- Seitdem wird die Syntax von Programmiersprachen durch formale Grammatiken beschrieben.



Noam Chomsky ist emeritierter Professor für Linguistik am MIT, einer der weltweit bekanntesten linken Intellektuellen und seit den 1960er Jahren einer der prominentesten Kritiker der US-amerikanischen Politik.

Chomsky gehört zu den bekanntesten Linguisten der Gegenwart. Er übte durch die Verbindung der Linguistik, Kognitionswissenschaften und Informatik besonders in der zweiten Hälfte des 20. Jahrhunderts starken Einfluss auf deren Entwicklung aus.

[Quelle: Wikipedia]

- Beispiele von Regeln für den Bau von Sätzen:

Satz → Nominalphrase Verbalphrase
Verbalphrase → Verb Nominalphrase
Nominalphrase → Artikel Nomen
Satz → Präpositionalphrase Verbalphrase

- Beispiele von Regeln für den Bau von Sätzen:

Satz → Nominalphrase Verbalphrase
Verbalphrase → Verb Nominalphrase
Nominalphrase → Artikel Nomen
Satz → Präpositionalphrase Verbalphrase

- Beispiele von Regeln für den Bau von Programmen:

keyword_item → identifier "=" expression
identifier → letter identifier2
identifier2 → letter identifier2
identifier2 → " "

- Das **Erkennungsproblem**: Ist eine gegebene Zeichenkette ein Programm, d.h. kann sie von den Regeln erzeugt werden?
- **Recognizer**: Programm, welches das Erkennungsproblem für eine gegebene Grammatik löst.
Recognizer → Parser → Compiler
- **Hauptfragen**:
 - Für welche Grammatiken gibt es effiziente Recognizer?
 - Gegeben eine Grammatik, wie kann ein Recognizer automatisch konstruiert werden?

2. Grundbegriffe

Definition 2.1

- Ein **Alphabet** Σ ist eine endliche Menge.
Bsp: $\{0, 1\}$, ASCII, Unicode.

2. Grundbegriffe

Definition 2.1

- Ein **Alphabet** Σ ist eine endliche Menge.
Bsp: $\{0, 1\}$, ASCII, Unicode.
- Ein **Wort**/String über Σ ist eine endliche Folge von Zeichen aus Σ , zB 010.

2. Grundbegriffe

Definition 2.1

- Ein **Alphabet** Σ ist eine endliche Menge.
Bsp: $\{0, 1\}$, ASCII, Unicode.
- Ein **Wort**/String über Σ ist eine endliche Folge von Zeichen aus Σ , zB 010.
- $|w|$ bezeichnet die Länge eines Wortes w .

2. Grundbegriffe

Definition 2.1

- Ein **Alphabet** Σ ist eine endliche Menge.
Bsp: $\{0, 1\}$, ASCII, Unicode.
- Ein **Wort**/String über Σ is eine endliche Folge von Zeichen aus Σ , zB 010.
- $|w|$ bezeichnet die Länge eines Wortes w .
- Das **leere Wort** (das einzige Wort der Länge 0) wird mit ϵ bezeichnet.

2. Grundbegriffe

Definition 2.1

- Ein **Alphabet** Σ ist eine endliche Menge.
Bsp: $\{0, 1\}$, ASCII, Unicode.
- Ein **Wort**/String über Σ ist eine endliche Folge von Zeichen aus Σ , zB 010.
- $|w|$ bezeichnet die Länge eines Wortes w .
- Das **leere Wort** (das einzige Wort der Länge 0) wird mit ϵ bezeichnet.
- Sind u und v Wörter, so ist uv ihre Konkatenation.

2. Grundbegriffe

Definition 2.1

- Ein **Alphabet** Σ ist eine endliche Menge.
Bsp: $\{0, 1\}$, ASCII, Unicode.
- Ein **Wort**/String über Σ ist eine endliche Folge von Zeichen aus Σ , zB 010.
- $|w|$ bezeichnet die Länge eines Wortes w .
- Das **leere Wort** (das einzige Wort der Länge 0) wird mit ϵ bezeichnet.
- Sind u und v Wörter, so ist uv ihre Konkatenation.
- Ist w ein Wort, so ist w^n definiert durch $w^0 = \epsilon$ und $w^{n+1} = ww^n$. Bsp: $(ab)^3 = ababab$.

2. Grundbegriffe

Definition 2.1

- Ein **Alphabet** Σ ist eine endliche Menge.
Bsp: $\{0, 1\}$, ASCII, Unicode.
- Ein **Wort**/String über Σ ist eine endliche Folge von Zeichen aus Σ , zB 010.
- $|w|$ bezeichnet die Länge eines Wortes w .
- Das **leere Wort** (das einzige Wort der Länge 0) wird mit ϵ bezeichnet.
- Sind u und v Wörter, so ist uv ihre Konkatenation.
- Ist w ein Wort, so ist w^n definiert durch $w^0 = \epsilon$ und $w^{n+1} = ww^n$. Bsp: $(ab)^3 = ababab$.
- Σ^* ist die Menge aller Wörter über Σ .

2. Grundbegriffe

Definition 2.1

- Ein **Alphabet** Σ ist eine endliche Menge.
Bsp: $\{0, 1\}$, ASCII, Unicode.
- Ein **Wort**/String über Σ ist eine endliche Folge von Zeichen aus Σ , zB 010.
- $|w|$ bezeichnet die Länge eines Wortes w .
- Das **leere Wort** (das einzige Wort der Länge 0) wird mit ϵ bezeichnet.
- Sind u und v Wörter, so ist uv ihre Konkatenation.
- Ist w ein Wort, so ist w^n definiert durch $w^0 = \epsilon$ und $w^{n+1} = ww^n$. Bsp: $(ab)^3 = ababab$.
- Σ^* ist die Menge aller Wörter über Σ .
- Eine Teilmenge $L \subseteq \Sigma^*$ ist eine **(formale) Sprache**.

Beispiel 2.2 (Formale Sprachen)

- Die Menge aller Wörter im Duden (24. Aufl.)

Beispiel 2.2 (Formale Sprachen)

- Die Menge aller Wörter im Duden (24. Aufl.)
- Die Menge der deutschen Sätze ist keine formale Sprache

Beispiel 2.2 (Formale Sprachen)

- Die Menge aller Wörter im Duden (24. Aufl.)
- Die Menge der deutschen Sätze ist keine formale Sprache
- $L_1 = \{\epsilon, ab, abab, ababab, \dots\} = \{(ab)^n \mid n \in \mathbb{N}\}$
($\Sigma_1 = \{a, b\}$)

Beispiel 2.2 (Formale Sprachen)

- Die Menge aller Wörter im Duden (24. Aufl.)
- Die Menge der deutschen Sätze ist keine formale Sprache
- $L_1 = \{\epsilon, ab, abab, ababab, \dots\} = \{(ab)^n \mid n \in \mathbb{N}\}$
($\Sigma_1 = \{a, b\}$)
- $L_2 = \{\epsilon, ab, aabb, aaabbb, \dots\} = \{a^n b^n \mid n \in \mathbb{N}\}$
($\Sigma_2 = \{a, b\}$)

Beispiel 2.2 (Formale Sprachen)

- Die Menge aller Wörter im Duden (24. Aufl.)
- Die Menge der deutschen Sätze ist keine formale Sprache
- $L_1 = \{\epsilon, ab, abab, ababab, \dots\} = \{(ab)^n \mid n \in \mathbb{N}\}$
($\Sigma_1 = \{a, b\}$)
- $L_2 = \{\epsilon, ab, aabb, aaabbb, \dots\} = \{a^n b^n \mid n \in \mathbb{N}\}$
($\Sigma_2 = \{a, b\}$)
- $L_3 = \{\epsilon, 1, 100, 1001, 10000, \dots\} =$
 $\{w \in \{0, 1\}^* \mid w \text{ ist eine binär kodierte Quadratzahl}\}$
($\Sigma_3 = \{0, 1\}$)

Beispiel 2.2 (Formale Sprachen)

- Die Menge aller Wörter im Duden (24. Aufl.)
- Die Menge der deutschen Sätze ist keine formale Sprache
- $L_1 = \{\epsilon, ab, abab, ababab, \dots\} = \{(ab)^n \mid n \in \mathbb{N}\}$
($\Sigma_1 = \{a, b\}$)
- $L_2 = \{\epsilon, ab, aabb, aaabbb, \dots\} = \{a^n b^n \mid n \in \mathbb{N}\}$
($\Sigma_2 = \{a, b\}$)
- $L_3 = \{\epsilon, 1, 100, 1001, 10000, \dots\} =$
 $\{w \in \{0, 1\}^* \mid w \text{ ist eine binär kodierte Quadratzahl}\}$
($\Sigma_3 = \{0, 1\}$)
- \emptyset

Beispiel 2.2 (Formale Sprachen)

- Die Menge aller Wörter im Duden (24. Aufl.)
- Die Menge der deutschen Sätze ist keine formale Sprache
- $L_1 = \{\epsilon, ab, abab, ababab, \dots\} = \{(ab)^n \mid n \in \mathbb{N}\}$
($\Sigma_1 = \{a, b\}$)
- $L_2 = \{\epsilon, ab, aabb, aaabbb, \dots\} = \{a^n b^n \mid n \in \mathbb{N}\}$
($\Sigma_2 = \{a, b\}$)
- $L_3 = \{\epsilon, 1, 100, 1001, 10000, \dots\} =$
 $\{w \in \{0, 1\}^* \mid w \text{ ist eine binär kodierte Quadratzahl}\}$
($\Sigma_3 = \{0, 1\}$)
- \emptyset
- $\{\epsilon\}$

Beispiel 2.2 (Formale Sprachen)

- Die Menge aller Wörter im Duden (24. Aufl.)
- Die Menge der deutschen Sätze ist keine formale Sprache
- $L_1 = \{\epsilon, ab, abab, ababab, \dots\} = \{(ab)^n \mid n \in \mathbb{N}\}$
($\Sigma_1 = \{a, b\}$)
- $L_2 = \{\epsilon, ab, aabb, aaabbb, \dots\} = \{a^n b^n \mid n \in \mathbb{N}\}$
($\Sigma_2 = \{a, b\}$)
- $L_3 = \{\epsilon, 1, 100, 1001, 10000, \dots\} =$
 $\{w \in \{0, 1\}^* \mid w \text{ ist eine binär kodierte Quadratzahl}\}$
($\Sigma_3 = \{0, 1\}$)
- \emptyset
- $\{\epsilon\}$
- ϵ ist keine Sprache

Definition 2.3 (Operationen auf Sprachen)

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

Definition 2.3 (Operationen auf Sprachen)

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

- Konkatenation: $AB = \{uv \mid u \in A \wedge v \in B\}$
Bsp: $\{ab, b\}\{a, bb\} = \{aba, abbb,$

Definition 2.3 (Operationen auf Sprachen)

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

- Konkatenation: $AB = \{uv \mid u \in A \wedge v \in B\}$

Bsp: $\{ab, b\}\{a, bb\} = \{aba, abbb, ba, bbb\}$

NB: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$

Definition 2.3 (Operationen auf Sprachen)

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

- Konkatination: $AB = \{uv \mid u \in A \wedge v \in B\}$
Bsp: $\{ab, b\}\{a, bb\} = \{aba, abbb, ba, bbb\}$
NB: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \dots w_n \mid w_1, \dots, w_n \in A\} = \underbrace{A \cdots A}_n$ Bsp:
 $\{ab, ba\}^2 = \{abab, abba, baab, baba\}$
Rekursiv: $A^0 = \{\epsilon\}$ und $A^{n+1} = AA^n$

Definition 2.3 (Operationen auf Sprachen)

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

- Konkatination: $AB = \{uv \mid u \in A \wedge v \in B\}$
Bsp: $\{ab, b\}\{a, bb\} = \{aba, abbb, ba, bbb\}$
NB: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \dots w_n \mid w_1, \dots, w_n \in A\} = \underbrace{A \dots A}_n$ Bsp:
 $\{ab, ba\}^2 = \{abab, abba, baab, baba\}$
Rekursiv: $A^0 = \{\epsilon\}$ und $A^{n+1} = AA^n$
- $A^* = \{w_1 \dots w_n \mid n \geq 0 \wedge w_1, \dots, w_n \in A\} = \bigcup_{n \in \mathbb{N}} A^n$
Bsp: $\{01\}^* = \{\epsilon, 01, 0101, 010101, \dots\}$

Definition 2.3 (Operationen auf Sprachen)

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

- Konkatination: $AB = \{uv \mid u \in A \wedge v \in B\}$
Bsp: $\{ab, b\}\{a, bb\} = \{aba, abbb, ba, bbb\}$
NB: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \dots w_n \mid w_1, \dots, w_n \in A\} = \underbrace{A \dots A}_n$ Bsp:
 $\{ab, ba\}^2 = \{abab, abba, baab, baba\}$
Rekursiv: $A^0 = \{\epsilon\}$ und $A^{n+1} = AA^n$
- $A^* = \{w_1 \dots w_n \mid n \geq 0 \wedge w_1, \dots, w_n \in A\} = \bigcup_{n \in \mathbb{N}} A^n$
Bsp: $\{01\}^* = \{\epsilon, 01, 0101, 010101, \dots\} \neq \{0, 1\}^*$

Definition 2.3 (Operationen auf Sprachen)

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

- Konkatination: $AB = \{uv \mid u \in A \wedge v \in B\}$
Bsp: $\{ab, b\}\{a, bb\} = \{aba, abbb, ba, bbb\}$
NB: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \dots w_n \mid w_1, \dots, w_n \in A\} = \underbrace{A \dots A}_n$ Bsp:
 $\{ab, ba\}^2 = \{abab, abba, baab, baba\}$
Rekursiv: $A^0 = \{\epsilon\}$ und $A^{n+1} = AA^n$
- $A^* = \{w_1 \dots w_n \mid n \geq 0 \wedge w_1, \dots, w_n \in A\} = \bigcup_{n \in \mathbb{N}} A^n$
Bsp: $\{01\}^* = \{\epsilon, 01, 0101, 010101, \dots\} \neq \{0, 1\}^*$
- $A^+ = AA^* = \bigcup_{n \geq 1} A^n$
Bsp: $\Sigma^+ =$ Menge aller nicht-leeren Wörter über Σ

Definition 2.3 (Operationen auf Sprachen)

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

- Konkatenation: $AB = \{uv \mid u \in A \wedge v \in B\}$
Bsp: $\{ab, b\}\{a, bb\} = \{aba, abbb, ba, bbb\}$
NB: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \dots w_n \mid w_1, \dots, w_n \in A\} = \underbrace{A \dots A}_n$ Bsp:
 $\{ab, ba\}^2 = \{abab, abba, baab, baba\}$
Rekursiv: $A^0 = \{\epsilon\}$ und $A^{n+1} = AA^n$
- $A^* = \{w_1 \dots w_n \mid n \geq 0 \wedge w_1, \dots, w_n \in A\} = \bigcup_{n \in \mathbb{N}} A^n$
Bsp: $\{01\}^* = \{\epsilon, 01, 0101, 010101, \dots\} \neq \{0, 1\}^*$
- $A^+ = AA^* = \bigcup_{n \geq 1} A^n$
Bsp: $\Sigma^+ =$ Menge aller nicht-leeren Wörter über Σ

Achtung:

- Für alle A : $\epsilon \in A^*$
- $\emptyset^* =$

Definition 2.3 (Operationen auf Sprachen)

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

- Konkatenation: $AB = \{uv \mid u \in A \wedge v \in B\}$
Bsp: $\{ab, b\}\{a, bb\} = \{aba, abbb, ba, bbb\}$
NB: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \dots w_n \mid w_1, \dots, w_n \in A\} = \underbrace{A \dots A}_n$ Bsp:
 $\{ab, ba\}^2 = \{abab, abba, baab, baba\}$
Rekursiv: $A^0 = \{\epsilon\}$ und $A^{n+1} = AA^n$
- $A^* = \{w_1 \dots w_n \mid n \geq 0 \wedge w_1, \dots, w_n \in A\} = \bigcup_{n \in \mathbb{N}} A^n$
Bsp: $\{01\}^* = \{\epsilon, 01, 0101, 010101, \dots\} \neq \{0, 1\}^*$
- $A^+ = AA^* = \bigcup_{n \geq 1} A^n$
Bsp: $\Sigma^+ =$ Menge aller nicht-leeren Wörter über Σ

Achtung:

- Für alle A : $\epsilon \in A^*$
- $\emptyset^* = \{\epsilon\}$

Einige Rechenregeln:

Lemma 2.4

- $\emptyset A = \emptyset$
- $\{\epsilon\}A = A$

Einige Rechenregeln:

Lemma 2.4

- $\emptyset A = \emptyset$
- $\{\epsilon\}A = A$

Lemma 2.5

- $A(B \cup C) = AB \cup AC$
- $(A \cup B)C = AC \cup BC$

Einige Rechenregeln:

Lemma 2.4

- $\emptyset A = \emptyset$
- $\{\epsilon\}A = A$

Lemma 2.5

- $A(B \cup C) = AB \cup AC$
- $(A \cup B)C = AC \cup BC$

Achtung: i.A. gilt $A(B \cap C) = AB \cap AC$ *nicht*.

Einige Rechenregeln:

Lemma 2.4

- $\emptyset A = \emptyset$
- $\{\epsilon\}A = A$

Lemma 2.5

- $A(B \cup C) = AB \cup AC$
- $(A \cup B)C = AC \cup BC$

Achtung: i.A. gilt $A(B \cap C) = AB \cap AC$ *nicht*.

Lemma 2.6

$$A^*A^* = A^*$$

2.1 Grammatiken

2.1 Grammatiken

Definition 2.7

Eine **Grammatik** ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei

V ist eine endliche Menge von **Nichtterminalzeichen** (oder **Nichtterminale**, oder **Variablen**),

Σ ist eine endliche Menge von **Terminalzeichen** (oder **Terminale**), disjunkt von V , auch genannt ein **Alphabet**,

$P \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$ ist eine Menge von **Produktionen**, und

$S \in V$ ist das **Startsymbol**.

2.1 Grammatiken

Definition 2.7

Eine **Grammatik** ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei

V ist eine endliche Menge von **Nichtterminalzeichen** (oder **Nichtterminale**, oder **Variablen**),

Σ ist eine endliche Menge von **Terminalzeichen** (oder **Terminale**),
disjunkt von V , auch genannt ein **Alphabet**,

$P \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$ ist eine Menge von **Produktionen**, und
 $S \in V$ ist das **Startsymbol**.

Konventionen:

- A, B, C, \dots sind Nichtterminale,
- a, b, c, \dots (und Sonderzeichen wie $+, *, \dots$) sind Terminale,
- $\alpha, \beta, \gamma, \dots \in (V \cup \Sigma)^*$
- Produktionen schreiben wir $\alpha \rightarrow \beta$ statt $(\alpha, \beta) \in P$.
- Statt $\alpha \rightarrow \beta_1, \dots, \alpha \rightarrow \beta_n$ schreiben wir $\alpha \rightarrow \beta_1 \mid \dots \mid \beta_n$

Beispiel 2.8 (Arithmetische Ausdrücke)

- $V = \{\langle \text{Expr} \rangle, \langle \text{Term} \rangle, \langle \text{Factor} \rangle\}$
- $\Sigma = \{a, b, c, +, *, (,)\}$
- $S = \langle \text{Expr} \rangle$
- Die Menge P enthält folgende Produktionen:

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Definition 2.9

Eine Grammatik $G = (V, \Sigma, P, S)$ induziert eine **Ableitungsrelation** \rightarrow_G auf Wörtern über $V \cup \Sigma$:

$$\alpha \rightarrow_G \alpha'$$

gdw es eine Regel $\beta \rightarrow \beta'$ in P und Wörter α_1, α_2 gibt, so dass

$$\alpha = \alpha_1 \beta \alpha_2 \quad \text{und} \quad \alpha' = \alpha_1 \beta' \alpha_2$$

Beispiel : Für die Grammatik der arithmetischen Ausdrücken gilt

$$a + \langle \text{Term} \rangle + b \quad \rightarrow_G \quad a + \langle \text{Term} \rangle * \langle \text{Factor} \rangle + b$$

Definition 2.9

Eine Grammatik $G = (V, \Sigma, P, S)$ induziert eine **Ableitungsrelation** \rightarrow_G auf Wörtern über $V \cup \Sigma$:

$$\alpha \rightarrow_G \alpha'$$

gdw es eine Regel $\beta \rightarrow \beta'$ in P und Wörter α_1, α_2 gibt, so dass

$$\alpha = \alpha_1 \beta \alpha_2 \quad \text{und} \quad \alpha' = \alpha_1 \beta' \alpha_2$$

Beispiel : Für die Grammatik der arithmetischen Ausdrücken gilt

$$a + \langle \text{Term} \rangle + b \quad \rightarrow_G \quad a + \langle \text{Term} \rangle * \langle \text{Factor} \rangle + b$$

Eine Sequenz $\alpha_1 \rightarrow_G \alpha_2 \rightarrow_G \cdots \rightarrow_G \alpha_n$ ist eine **Ableitung** von α_n aus α_1 .

Wenn $\alpha_1 = S$ und $\alpha_n \in \Sigma^*$, dann **erzeugt** G das Wort α_n .

Die **Sprache** von G ist die Menge aller Wörter, die von G erzeugt werden. Sie wird mit $L(G)$ bezeichnet.

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\langle \text{Expr} \rangle \rightarrow$$

$$\rightarrow a * (b + c)$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\langle \text{Expr} \rangle \rightarrow$$

$$\rightarrow a * (b + c)$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\langle \text{Expr} \rangle \rightarrow$$

$$\rightarrow a * (b + c)$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$$

$$\rightarrow a * (b + c)$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$$

$$\rightarrow a * (b + c)$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$$

$$\rightarrow a * (b + c)$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$$

$$\rightarrow a * (b + c)$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ &\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle\end{aligned}$$

$$\rightarrow a * (b + c)$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ &\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle\end{aligned}$$

$$\rightarrow a * (b + c)$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ &\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle\end{aligned}$$

$$\rightarrow a * (b + c)$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ &\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle\end{aligned}$$

$$\rightarrow a * (b + c)$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ &\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle \\ &\rightarrow a * (\langle \text{Expr} \rangle)\end{aligned}$$

$$\rightarrow a * (b + c)$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ &\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle \\ &\rightarrow a * (\langle \text{Expr} \rangle)\end{aligned}$$

$$\rightarrow a * (b + c)$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ &\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle \\ &\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (b + c)\end{aligned}$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ &\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle \\ &\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (b + c)\end{aligned}$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ &\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle \\ &\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (\langle \text{Term} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (b + c)\end{aligned}$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ &\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle \\ &\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (\langle \text{Term} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (b + c)\end{aligned}$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ &\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle \\ &\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (\langle \text{Term} \rangle + \langle \text{Term} \rangle) \rightarrow a * (\langle \text{Factor} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (b + c)\end{aligned}$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ &\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle \\ &\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (\langle \text{Term} \rangle + \langle \text{Term} \rangle) \rightarrow a * (\langle \text{Factor} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (b + c)\end{aligned}$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ &\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle \\ &\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (\langle \text{Term} \rangle + \langle \text{Term} \rangle) \rightarrow a * (\langle \text{Factor} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (b + \langle \text{Term} \rangle) \\ &\rightarrow a * (b + c)\end{aligned}$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ &\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle \\ &\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (\langle \text{Term} \rangle + \langle \text{Term} \rangle) \rightarrow a * (\langle \text{Factor} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (b + \langle \text{Term} \rangle) \\ &\rightarrow a * (b + c)\end{aligned}$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ &\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle \\ &\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (\langle \text{Term} \rangle + \langle \text{Term} \rangle) \rightarrow a * (\langle \text{Factor} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (b + \langle \text{Term} \rangle) \rightarrow a * (b + \langle \text{Factor} \rangle) \\ &\rightarrow a * (b + c)\end{aligned}$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ &\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle \\ &\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (\langle \text{Term} \rangle + \langle \text{Term} \rangle) \rightarrow a * (\langle \text{Factor} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (b + \langle \text{Term} \rangle) \rightarrow a * (b + \langle \text{Factor} \rangle) \\ &\rightarrow a * (b + c)\end{aligned}$$

Beispiel 2.10 (Arithmetische Ausdrücke)

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\ \langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\ \langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)\end{aligned}$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung von $a * (b + c)$:

$$\begin{aligned}\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\ &\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle \\ &\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (\langle \text{Term} \rangle + \langle \text{Term} \rangle) \rightarrow a * (\langle \text{Factor} \rangle + \langle \text{Term} \rangle) \\ &\rightarrow a * (b + \langle \text{Term} \rangle) \rightarrow a * (b + \langle \text{Factor} \rangle) \\ &\rightarrow a * (b + c)\end{aligned}$$

Beispiel 2.11

Welche Grammatik erzeugt $L = \{a^n b^n c^n \mid n \geq 0\}$?

$$G_1 : \quad S \rightarrow abcS \mid \epsilon$$

$$ba \rightarrow ab$$

$$cb \rightarrow bc$$

$$ca \rightarrow ac$$

$$G_2 : \quad S \rightarrow aBSc \mid \epsilon$$

$$Ba \rightarrow aB$$

$$Bb \rightarrow bB$$

$$Bc \rightarrow bc$$

Beispiel 2.11

Welche Grammatik erzeugt $L = \{a^n b^n c^n \mid n \geq 0\}$?

$$\begin{array}{ll} G_1 : & S \rightarrow abcS \mid \epsilon \\ & ba \rightarrow ab \\ & cb \rightarrow bc \\ & ca \rightarrow ac \end{array} \qquad \begin{array}{ll} G_2 : & S \rightarrow aBSc \mid \epsilon \\ & Ba \rightarrow aB \\ & Bb \rightarrow bB \\ & Bc \rightarrow bc \end{array}$$

Es gilt $L(G_1) \neq L$ weil $S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc$

Beispiel 2.11

Welche Grammatik erzeugt $L = \{a^n b^n c^n \mid n \geq 0\}$?

$$\begin{array}{ll} G_1 : & S \rightarrow abcS \mid \epsilon \\ & ba \rightarrow ab \\ & cb \rightarrow bc \\ & ca \rightarrow ac \\ G_2 : & S \rightarrow aBSc \mid \epsilon \\ & Ba \rightarrow aB \\ & Bb \rightarrow bB \\ & Bc \rightarrow bc \end{array}$$

Es gilt $L(G_1) \neq L$ weil $S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc$

Aber $L \subseteq L(G_1)$. Ableitung von $aabbcc$ aus S :

Beispiel 2.11

Welche Grammatik erzeugt $L = \{a^n b^n c^n \mid n \geq 0\}$?

$$\begin{array}{ll} G_1 : & S \rightarrow abcS \mid \epsilon \\ & ba \rightarrow ab \\ & cb \rightarrow bc \\ & ca \rightarrow ac \\ G_2 : & S \rightarrow aBSc \mid \epsilon \\ & Ba \rightarrow aB \\ & Bb \rightarrow bB \\ & Bc \rightarrow bc \end{array}$$

Es gilt $L(G_1) \neq L$ weil $S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc$

Aber $L \subseteq L(G_1)$. Ableitung von $aabbcc$ aus S :

$$S \rightarrow abcS$$

Beispiel 2.11

Welche Grammatik erzeugt $L = \{a^n b^n c^n \mid n \geq 0\}$?

$$\begin{array}{ll} G_1 : & S \rightarrow abcS \mid \epsilon \\ & ba \rightarrow ab \\ & cb \rightarrow bc \\ & ca \rightarrow ac \end{array} \qquad \begin{array}{ll} G_2 : & S \rightarrow aBSc \mid \epsilon \\ & Ba \rightarrow aB \\ & Bb \rightarrow bB \\ & Bc \rightarrow bc \end{array}$$

Es gilt $L(G_1) \neq L$ weil $S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc$

Aber $L \subseteq L(G_1)$. Ableitung von $aabbcc$ aus S :

$$S \rightarrow abcS \rightarrow abcabcS$$

Beispiel 2.11

Welche Grammatik erzeugt $L = \{a^n b^n c^n \mid n \geq 0\}$?

$$\begin{array}{ll} G_1 : & S \rightarrow abcS \mid \epsilon \\ & ba \rightarrow ab \\ & cb \rightarrow bc \\ & ca \rightarrow ac \\ G_2 : & S \rightarrow aBSc \mid \epsilon \\ & Ba \rightarrow aB \\ & Bb \rightarrow bB \\ & Bc \rightarrow bc \end{array}$$

Es gilt $L(G_1) \neq L$ weil $S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc$

Aber $L \subseteq L(G_1)$. Ableitung von $aabbcc$ aus S :

$S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc$

Beispiel 2.11

Welche Grammatik erzeugt $L = \{a^n b^n c^n \mid n \geq 0\}$?

$$\begin{array}{ll} G_1 : & S \rightarrow abcS \mid \epsilon \\ & ba \rightarrow ab \\ & cb \rightarrow bc \\ & ca \rightarrow ac \\ G_2 : & S \rightarrow aBSc \mid \epsilon \\ & Ba \rightarrow aB \\ & Bb \rightarrow bB \\ & Bc \rightarrow bc \end{array}$$

Es gilt $L(G_1) \neq L$ weil $S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc$

Aber $L \subseteq L(G_1)$. Ableitung von $aabbcc$ aus S :

$S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc \rightarrow abacbc$

Beispiel 2.11

Welche Grammatik erzeugt $L = \{a^n b^n c^n \mid n \geq 0\}$?

$$\begin{array}{ll} G_1 : & S \rightarrow abcS \mid \epsilon \\ & ba \rightarrow ab \\ & cb \rightarrow bc \\ & ca \rightarrow ac \end{array} \qquad \begin{array}{ll} G_2 : & S \rightarrow aBSc \mid \epsilon \\ & Ba \rightarrow aB \\ & Bb \rightarrow bB \\ & Bc \rightarrow bc \end{array}$$

Es gilt $L(G_1) \neq L$ weil $S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc$

Aber $L \subseteq L(G_1)$. Ableitung von $aabbcc$ aus S :

$S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc \rightarrow abacbc \rightarrow aabc bc$

Beispiel 2.11

Welche Grammatik erzeugt $L = \{a^n b^n c^n \mid n \geq 0\}$?

$$\begin{array}{ll} G_1 : & S \rightarrow abcS \mid \epsilon \\ & ba \rightarrow ab \\ & cb \rightarrow bc \\ & ca \rightarrow ac \end{array} \qquad \begin{array}{ll} G_2 : & S \rightarrow aBSc \mid \epsilon \\ & Ba \rightarrow aB \\ & Bb \rightarrow bB \\ & Bc \rightarrow bc \end{array}$$

Es gilt $L(G_1) \neq L$ weil $S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc$

Aber $L \subseteq L(G_1)$. Ableitung von $aabbcc$ aus S :

$S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc \rightarrow abacbc \rightarrow aabcbc \rightarrow aabbcc$

Beispiel 2.11

Welche Grammatik erzeugt $L = \{a^n b^n c^n \mid n \geq 0\}$?

$$\begin{array}{ll} G_1 : & S \rightarrow abcS \mid \epsilon \\ & ba \rightarrow ab \\ & cb \rightarrow bc \\ & ca \rightarrow ac \end{array} \qquad \begin{array}{ll} G_2 : & S \rightarrow aBSc \mid \epsilon \\ & Ba \rightarrow aB \\ & Bb \rightarrow bB \\ & Bc \rightarrow bc \end{array}$$

Es gilt $L(G_1) \neq L$ weil $S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc$

Aber $L \subseteq L(G_1)$. Ableitung von $aabbcc$ aus S :

$S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc \rightarrow abacbc \rightarrow aabcbc \rightarrow aabbcc$

Es gilt: $L(G_2) = L$. Ableitung von $aabbcc$ aus S :

Beispiel 2.11

Welche Grammatik erzeugt $L = \{a^n b^n c^n \mid n \geq 0\}$?

$$\begin{array}{ll} G_1 : & S \rightarrow abcS \mid \epsilon \\ & ba \rightarrow ab \\ & cb \rightarrow bc \\ & ca \rightarrow ac \\ G_2 : & S \rightarrow aBSc \mid \epsilon \\ & Ba \rightarrow aB \\ & Bb \rightarrow bB \\ & Bc \rightarrow bc \end{array}$$

Es gilt $L(G_1) \neq L$ weil $S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc$

Aber $L \subseteq L(G_1)$. Ableitung von $aabbcc$ aus S :

$S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc \rightarrow abacbc \rightarrow aabc bc \rightarrow aabbcc$

Es gilt: $L(G_2) = L$. Ableitung von $aabbcc$ aus S :

$S \rightarrow aBSc$

Beispiel 2.11

Welche Grammatik erzeugt $L = \{a^n b^n c^n \mid n \geq 0\}$?

$$\begin{array}{ll} G_1 : & S \rightarrow abcS \mid \epsilon \\ & ba \rightarrow ab \\ & cb \rightarrow bc \\ & ca \rightarrow ac \\ G_2 : & S \rightarrow aBSc \mid \epsilon \\ & Ba \rightarrow aB \\ & Bb \rightarrow bB \\ & Bc \rightarrow bc \end{array}$$

Es gilt $L(G_1) \neq L$ weil $S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc$

Aber $L \subseteq L(G_1)$. Ableitung von $aabbcc$ aus S :

$S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc \rightarrow abacbc \rightarrow aabcbc \rightarrow aabbcc$

Es gilt: $L(G_2) = L$. Ableitung von $aabbcc$ aus S :

$S \rightarrow aBSc \rightarrow aBaBSc$

Beispiel 2.11

Welche Grammatik erzeugt $L = \{a^n b^n c^n \mid n \geq 0\}$?

$$\begin{array}{ll} G_1 : & S \rightarrow abcS \mid \epsilon \\ & ba \rightarrow ab \\ & cb \rightarrow bc \\ & ca \rightarrow ac \end{array} \qquad \begin{array}{ll} G_2 : & S \rightarrow aBSc \mid \epsilon \\ & Ba \rightarrow aB \\ & Bb \rightarrow bB \\ & Bc \rightarrow bc \end{array}$$

Es gilt $L(G_1) \neq L$ weil $S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc$

Aber $L \subseteq L(G_1)$. Ableitung von $aabbcc$ aus S :

$$S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc \rightarrow abacbc \rightarrow aabcbc \rightarrow aabbcc$$

Es gilt: $L(G_2) = L$. Ableitung von $aabbcc$ aus S :

$$S \rightarrow aBSc \rightarrow aBaBSc \rightarrow aBaBcc$$

Beispiel 2.11

Welche Grammatik erzeugt $L = \{a^n b^n c^n \mid n \geq 0\}$?

$$\begin{array}{ll} G_1 : & S \rightarrow abcS \mid \epsilon \\ & ba \rightarrow ab \\ & cb \rightarrow bc \\ & ca \rightarrow ac \\ G_2 : & S \rightarrow aBSc \mid \epsilon \\ & Ba \rightarrow aB \\ & Bb \rightarrow bB \\ & Bc \rightarrow bc \end{array}$$

Es gilt $L(G_1) \neq L$ weil $S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc$

Aber $L \subseteq L(G_1)$. Ableitung von $aabbcc$ aus S :

$S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc \rightarrow abacbc \rightarrow aabcbc \rightarrow aabbcc$

Es gilt: $L(G_2) = L$. Ableitung von $aabbcc$ aus S :

$S \rightarrow aBSc \rightarrow aBaBSc \rightarrow aBaBcc \rightarrow aaBBcc$

Beispiel 2.11

Welche Grammatik erzeugt $L = \{a^n b^n c^n \mid n \geq 0\}$?

$$\begin{array}{ll} G_1 : & S \rightarrow abcS \mid \epsilon \\ & ba \rightarrow ab \\ & cb \rightarrow bc \\ & ca \rightarrow ac \\ G_2 : & S \rightarrow aBSc \mid \epsilon \\ & Ba \rightarrow aB \\ & Bb \rightarrow bB \\ & Bc \rightarrow bc \end{array}$$

Es gilt $L(G_1) \neq L$ weil $S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc$

Aber $L \subseteq L(G_1)$. Ableitung von $aabbcc$ aus S :

$$S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc \rightarrow abacbc \rightarrow aabcbc \rightarrow aabbcc$$

Es gilt: $L(G_2) = L$. Ableitung von $aabbcc$ aus S :

$$S \rightarrow aBSc \rightarrow aBaBSc \rightarrow aBaBcc \rightarrow aaBBcc \rightarrow aaBbcc$$

Beispiel 2.11

Welche Grammatik erzeugt $L = \{a^n b^n c^n \mid n \geq 0\}$?

$$\begin{array}{ll} G_1 : & S \rightarrow abcS \mid \epsilon \\ & ba \rightarrow ab \\ & cb \rightarrow bc \\ & ca \rightarrow ac \\ G_2 : & S \rightarrow aBSc \mid \epsilon \\ & Ba \rightarrow aB \\ & Bb \rightarrow bB \\ & Bc \rightarrow bc \end{array}$$

Es gilt $L(G_1) \neq L$ weil $S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc$

Aber $L \subseteq L(G_1)$. Ableitung von $aabbcc$ aus S :

$S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc \rightarrow abacbc \rightarrow aabc bc \rightarrow aabbcc$

Es gilt: $L(G_2) = L$. Ableitung von $aabbcc$ aus S :

$S \rightarrow aBSc \rightarrow aBaBSc \rightarrow aBaBcc \rightarrow aaBBcc \rightarrow aaBbcc \rightarrow aabBcc$

Beispiel 2.11

Welche Grammatik erzeugt $L = \{a^n b^n c^n \mid n \geq 0\}$?

$$\begin{array}{ll} G_1 : & S \rightarrow abcS \mid \epsilon \\ & ba \rightarrow ab \\ & cb \rightarrow bc \\ & ca \rightarrow ac \\ G_2 : & S \rightarrow aBSc \mid \epsilon \\ & Ba \rightarrow aB \\ & Bb \rightarrow bB \\ & Bc \rightarrow bc \end{array}$$

Es gilt $L(G_1) \neq L$ weil $S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc$

Aber $L \subseteq L(G_1)$. Ableitung von $aabbcc$ aus S :

$S \rightarrow abcS \rightarrow abcabcS \rightarrow abcabc \rightarrow abacbc \rightarrow aabcbc \rightarrow aabbcc$

Es gilt: $L(G_2) = L$. Ableitung von $aabbcc$ aus S :

$S \rightarrow aBSc \rightarrow aBaBSc \rightarrow aBaBcc \rightarrow aaBBcc \rightarrow aaBbcc \rightarrow aabBcc \rightarrow aabbcc$

Definition 2.12 (Reflexive transitive Hülle)

$$\alpha \rightarrow_G^0 \alpha$$

$$\alpha \rightarrow_G^{n+1} \gamma \quad :\Leftrightarrow \quad \exists \beta. \alpha \rightarrow_G^n \beta \rightarrow_G \gamma$$

$$\alpha \rightarrow_G^* \beta \quad :\Leftrightarrow \quad \exists n. \alpha \rightarrow_G^n \beta$$

$$\alpha \rightarrow_G^+ \beta \quad :\Leftrightarrow \quad \exists n > 0. \alpha \rightarrow_G^n \beta$$

Beispiel: $\langle \text{Expr} \rangle \rightarrow_G^{11} a * (b + c)$ und so $\langle \text{Expr} \rangle \rightarrow_G^* a * (b + c)$.

Es gilt: $L(G) = \{w \in \Sigma^* \mid S \rightarrow_G^* w\}$

2.2 Die Chomsky-Hierarchie

Eine Grammatik G ist vom

2.2 Die Chomsky-Hierarchie

Eine Grammatik G ist vom

Typ 0 immer.

2.2 Die Chomsky-Hierarchie

Eine Grammatik G ist vom

Typ 0 immer.

Typ 1 falls für jede Produktion $\alpha \rightarrow \beta$ außer $S \rightarrow \epsilon$
gilt $|\alpha| \leq |\beta|$

2.2 Die Chomsky-Hierarchie

Eine Grammatik G ist vom

Typ 0 immer.

Typ 1 falls für jede Produktion $\alpha \rightarrow \beta$ außer $S \rightarrow \epsilon$
gilt $|\alpha| \leq |\beta|$

Typ 2 falls G vom Typ 1 ist und
für jede Produktion $\alpha \rightarrow \beta$ gilt $\alpha \in V$.

2.2 Die Chomsky-Hierarchie

Eine Grammatik G ist vom

Typ 0 immer.

Typ 1 falls für jede Produktion $\alpha \rightarrow \beta$ außer $S \rightarrow \epsilon$ gilt $|\alpha| \leq |\beta|$

Typ 2 falls G vom Typ 1 ist und für jede Produktion $\alpha \rightarrow \beta$ gilt $\alpha \in V$.

Typ 3 falls G vom Typ 2 ist und für jede Produktion $\alpha \rightarrow \beta$ außer $S \rightarrow \epsilon$ gilt $\beta \in \Sigma \cup \Sigma V$.

2.2 Die Chomsky-Hierarchie

Eine Grammatik G ist vom

Typ 0 immer.

Typ 1 falls für jede Produktion $\alpha \rightarrow \beta$ außer $S \rightarrow \epsilon$ gilt $|\alpha| \leq |\beta|$

Typ 2 falls G vom Typ 1 ist und für jede Produktion $\alpha \rightarrow \beta$ gilt $\alpha \in V$.

Typ 3 falls G vom Typ 2 ist und für jede Produktion $\alpha \rightarrow \beta$ außer $S \rightarrow \epsilon$ gilt $\beta \in \Sigma \cup \Sigma V$.

Offensichtlich gilt:

$$\text{Typ 3} \subset \text{Typ 2} \subset \text{Typ 1} \subset \text{Typ 0}$$

Grammatiken und Sprachklassen:

Typ 3 Rechtslineare Grammatik

Reguläre Sprachen

Grammatiken und Sprachklassen:

Typ 3 Rechtslineare Grammatik

Reguläre Sprachen

Typ 2 Kontextfreie Grammatik

Kontextfreie Sprachen

Grammatiken und Sprachklassen:

Typ 3	Rechtslineare Grammatik	Reguläre Sprachen
Typ 2	Kontextfreie Grammatik	Kontextfreie Sprachen
Typ 1	Kontextsensitive Grammatik	Kontextsens. Sprachen

Grammatiken und Sprachklassen:

Typ 3	Rechtslineare Grammatik	Reguläre Sprachen
Typ 2	Kontextfreie Grammatik	Kontextfreie Sprachen
Typ 1	Kontextsensitive Grammatik	Kontextsens. Sprachen
Typ 0	Phrasenstrukturgrammatik	Rekursiv aufzählbare Sprachen

Grammatiken und Sprachklassen:

Typ 3	Rechtslineare Grammatik	Reguläre Sprachen
Typ 2	Kontextfreie Grammatik	Kontextfreie Sprachen
Typ 1	Kontextsensitive Grammatik	Kontextsens. Sprachen
Typ 0	Phrasenstrukturgrammatik	Rekursiv aufzählbare Sprachen

Satz 2.13

$$L(\text{Typ } 3) \subset L(\text{Typ } 2) \subset L(\text{Typ } 1) \subset L(\text{Typ } 0)$$

Grammatiken und Sprachklassen:

Typ 3	Rechtslineare Grammatik	Reguläre Sprachen
Typ 2	Kontextfreie Grammatik	Kontextfreie Sprachen
Typ 1	Kontextsensitive Grammatik	Kontextsens. Sprachen
Typ 0	Phrasenstrukturgrammatik	Rekursiv aufzählbare Sprachen

Satz 2.13

$$L(\text{Typ } 3) \subset L(\text{Typ } 2) \subset L(\text{Typ } 1) \subset L(\text{Typ } 0)$$

Bemerkung Wir benutzen später eine etwas liberalere Definition von kontextfreien Grammatiken, die aber die gleiche Klasse von kontextfreien Sprachen erzeugt.

Das **Wortproblem**:

Gegeben: eine Grammatik G , ein Wort $w \in \Sigma^$*

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

Das **Wortproblem**:

Gegeben: eine Grammatik G , ein Wort $w \in \Sigma^$*

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

In den kommenden Wochen untersuchen wir das Wortproblem für Grammatiken vom Typ 3 und Typ 2.

Das **Wortproblem**:

Gegeben: eine Grammatik G , ein Wort $w \in \Sigma^$*

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

In den kommenden Wochen untersuchen wir das Wortproblem für Grammatiken vom Typ 3 und Typ 2.

Wir studieren Algorithmen, die für eine gegebene Grammatik G einen **Automaten** A_G konstruieren, und untersuchen ihre Laufzeit.

Das **Wortproblem**:

Gegeben: eine Grammatik G , ein Wort $w \in \Sigma^$*

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

In den kommenden Wochen untersuchen wir das Wortproblem für Grammatiken vom Typ 3 und Typ 2.

Wir studieren Algorithmen, die für eine gegebene Grammatik G einen **Automaten** A_G konstruieren, und untersuchen ihre Laufzeit.

A_G ist eine abstrakte Beschreibung eines Programms zur Lösung des Wortproblems.

Das **Wortproblem**:

Gegeben: eine Grammatik G , ein Wort $w \in \Sigma^$*

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

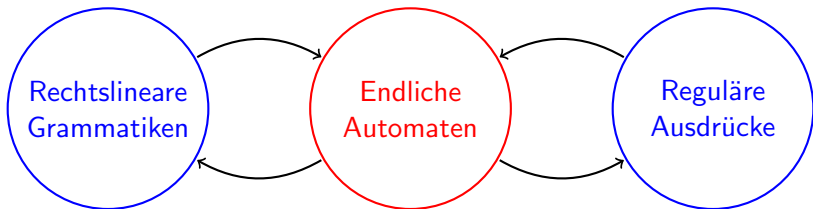
In den kommenden Wochen untersuchen wir das Wortproblem für Grammatiken vom Typ 3 und Typ 2.

Wir studieren Algorithmen, die für eine gegebene Grammatik G einen **Automaten** A_G konstruieren, und untersuchen ihre Laufzeit.

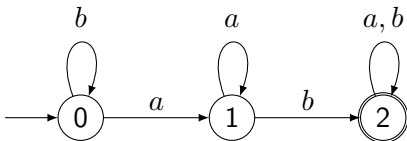
A_G ist eine abstrakte Beschreibung eines Programms zur Lösung des Wortproblems.

Der Algorithmus mit G als Eingabe und A_G als Ausgabe ist eine abstrakte Beschreibung eines Programms für die automatische Synthese von Recognizern (lex, yacc).

3. Reguläre Sprachen



3.1 Deterministische endliche Automaten



Eingabewort $baba \rightsquigarrow$ Zustandsfolge 0,0,1,2,2.

Erkannte Sprache: Menge der Wörter, die vom Startzustand in einen Endzustand führen.

Recognizer, die nur einmal das Wort durchläuft und in linearer Zeit es akzeptiert oder ablehnt.

Definition 3.1

Ein **deterministischer endlicher Automat** (*deterministic finite automaton*, DFA) $M = (Q, \Sigma, \delta, q_0, F)$ besteht aus

- einer endlichen Menge von **Zuständen** Q ,
- einem (endlichen) **Eingabealphabet** Σ ,
- einer (totalen!) **Übergangsfunktion** $\delta : Q \times \Sigma \rightarrow Q$,
- einem **Startzustand** $q_0 \in Q$, und
- einer Menge $F \subseteq Q$ von **Endzuständen** (**akzeptierenden Zust.**)

Definition 3.2

Die von M akzeptierte Sprache ist

$$L(M) := \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\},$$

wobei $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ induktiv definiert ist durch

$$\begin{aligned}\hat{\delta}(q, \epsilon) &= q \\ \hat{\delta}(q, aw) &= \hat{\delta}(\delta(q, a), w) \quad \text{für } a \in \Sigma, w \in \Sigma^* .\end{aligned}$$

($\hat{\delta}(q, w)$ bezeichnet den Zustand, den man aus q mit w erreicht.)

Eine Sprache ist regulär gdw sie von einem DFA akzeptiert wird.

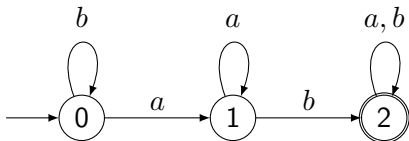
Eigenschaften von $\hat{\delta}$:

- $\hat{\delta}(q, a) = \delta(q, a)$.
- $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$.

Graphische Darstellung

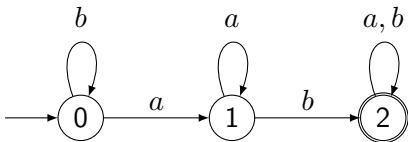
- Endliche Automaten können durch gerichtete und markierte **Zustandsgraphen** veranschaulicht werden:
 - Knoten $\hat{=}$ Zuständen
 - Kanten $\hat{=}$ Übergängen: $p \xrightarrow{a} q \hat{=} \delta(p, a) = q$
- Der Anfangszustand wird durch einen Pfeil, Endzustände werden durch doppelte Kreise gekennzeichnet.

Beispiel 3.3



Die Sprache des DFAs ist die Menge aller Wörter über $\{a, b\}$, die ab enthalten.

Beispiel 3.3



Die Sprache des DFAs ist die Menge aller Wörter über $\{a, b\}$, die ab enthalten.

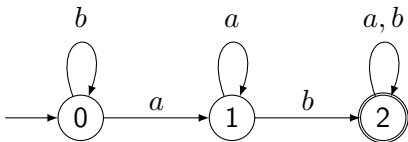
- Jedes Wort, das akzeptiert wird, enthält ab .

Sei w ein Wort, welches akzeptiert wird.

Betrachte in der Zustandsfolge für w den Zeitpunkt, in dem Zustand 1 zum letzten Mal besucht wird (existiert weil die Folge in Zustand 2 endet).

Unmittelbar davor wird a gelesen und unmittelbar danach b .

Beispiel 3.3



Die Sprache des DFAs ist die Menge aller Wörter über $\{a, b\}$, die ab enthalten.

- Jedes Wort, das akzeptiert wird, enthält ab .

Sei w ein Wort, welches akzeptiert wird.

Betrachte in der Zustandsfolge für w den Zeitpunkt, in dem Zustand 1 zum letzten Mal besucht wird (existiert weil die Folge in Zustand 2 endet).

Unmittelbar davor wird a gelesen und unmittelbar danach b .

- Jedes Wort, das ab enthält, wird akzeptiert.

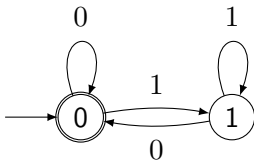
Für alle $q \in \{0, 1, 2\}$ gilt: $\hat{\delta}(q, ab) = 2$ und Zustand 2 kann nicht verlassen werden.

Beispiel 3.4

Für $w \in \{0, 1\}^*$ sei $\#w$ die von w binär repräsentierte Zahl,
zB $\#100 = 4$.

Beispiel 3.4

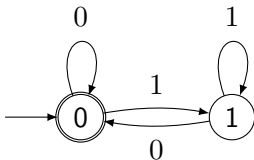
Für $w \in \{0, 1\}^*$ sei $\#w$ die von w binär repräsentierte Zahl,
zB $\#100 = 4$.



Der DFA akzeptiert genau die $w \in \{0, 1\}^*$ so dass $\#w$ gerade ist.

Beispiel 3.4

Für $w \in \{0, 1\}^*$ sei $\#w$ die von w binär repräsentierte Zahl,
zB $\#100 = 4$.



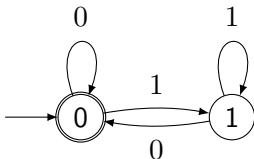
Der DFA akzeptiert genau die $w \in \{0, 1\}^*$ so dass $\#w$ gerade ist.

Beweis:

Für alle $w \neq \epsilon$, $\#w$ ist gerade gdw w endet mit 0.

Beispiel 3.4

Für $w \in \{0, 1\}^*$ sei $\#w$ die von w binär repräsentierte Zahl, zB $\#100 = 4$.



Der DFA akzeptiert genau die $w \in \{0, 1\}^*$ so dass $\#w$ gerade ist.

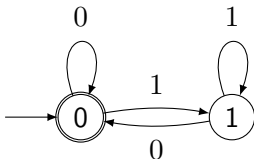
Beweis:

Für alle $w \neq \epsilon$, $\#w$ ist gerade gdw w endet mit 0.

1. $\hat{\delta}(0, w0) = \delta(\hat{\delta}(0, w), 0) = 0 \in F$, daher $w0 \in L(A)$

Beispiel 3.4

Für $w \in \{0, 1\}^*$ sei $\#w$ die von w binär repräsentierte Zahl,
zB $\#100 = 4$.



Der DFA akzeptiert genau die $w \in \{0, 1\}^*$ so dass $\#w$ gerade ist.

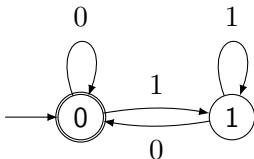
Beweis:

Für alle $w \neq \epsilon$, $\#w$ ist gerade gdw w endet mit 0.

1. $\hat{\delta}(0, w0) = \delta(\hat{\delta}(0, w), 0) = 0 \in F$, daher $w0 \in L(A)$
2. $\hat{\delta}(0, w1) = \delta(\hat{\delta}(0, w), 1) = 1 \notin F$, daher $w1 \notin L(A)$

Beispiel 3.4

Für $w \in \{0, 1\}^*$ sei $\#w$ die von w binär repräsentierte Zahl, zB $\#100 = 4$.



Der DFA akzeptiert genau die $w \in \{0, 1\}^*$ so dass $\#w$ gerade ist.

Beweis:

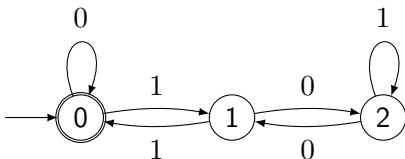
Für alle $w \neq \epsilon$, $\#w$ ist gerade gdw w endet mit 0.

1. $\hat{\delta}(0, w0) = \delta(\hat{\delta}(0, w), 0) = 0 \in F$, daher $w0 \in L(A)$

2. $\hat{\delta}(0, w1) = \delta(\hat{\delta}(0, w), 1) = 1 \notin F$, daher $w1 \notin L(A)$

Für $w = \epsilon$, $\hat{\delta}(0, w) = 0$, daher $\epsilon \in L(A)$ und $\#\epsilon = 0$. □

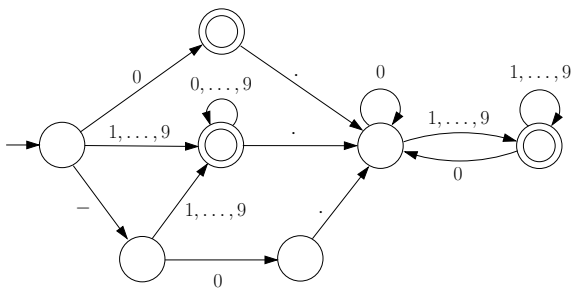
Beispiel 3.5



Für $w \in \{0, 1\}^*$ sei $\#w$ die von w binär repräsentierte Zahl, zB $\#100 = 4$.

Der DFA akzeptiert genau die Wörter $w \in \{0, 1\}^*$ mit: $\#w$ ist ein vielfaches von 3.

Beispiel 3.6



Ein DFA für die Dezimalzahlen

3.2 Von rechtslinearen Grammatiken zu DFA (und zurück)

Wir zeigen:

- Für jede rechtslineare Grammatik G gibt es einen DFA M mit $L(M) = L(G)$.
- Für jeden DFA M gibt es eine rechtslineare Grammatik G mit $L(G) = L(M)$.

3.2 Von rechtslinearen Grammatiken zu DFA (und zurück)

Wir zeigen:

- Für jede rechtslineare Grammatik G gibt es einen DFA M mit $L(M) = L(G)$.
- Für jeden DFA M gibt es eine rechtslineare Grammatik G mit $L(G) = L(M)$.

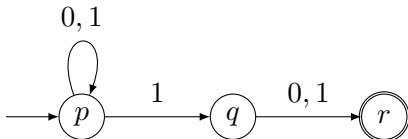
Wir führen **nichtdeterministische endliche Automaten** (NFA) als “Zwischenschritt” ein. Wir zeigen:

- Für jede rechtslineare Grammatik G gibt es einen NFA N mit $L(N) = L(G)$.
- Für jeden NFA N gibt es einen DFA M mit $L(M) = L(N)$.
- Für jeden DFA M gibt es eine rechtslineare Grammatik G mit $L(G) = L(M)$.

NFA sind eine Verallgemeinerung von DFA: Aus einem Zustand eines NFAs können 0, 1, 2, oder mehr Transitionen mit derselben Beschriftung ausgehen.

Beispiel 3.7

Erkennung der Binärzahlen, deren vorletzte Ziffer 1 ist:



Wort wird akzeptiert gdw es einen Weg zum Endzustand gibt.

Intuitive Vorstellung: Der Automat "rät" den richtigen Weg.

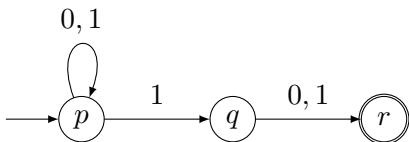
Nicht nützlich als Recognizer, aber doch als Zwischenschritt (oder als Datenstruktur).

Definition 3.8

Ein **nichtdeterministischer endlicher Automat** (*nondeterministic finite automaton*, NFA) ist ein 5-Tupel $N = (Q, \Sigma, \delta, q_0, F)$, so dass

- Q , Σ , q_0 und F sind wie bei einem DFA
- $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$
 $\mathcal{P}(Q)$ = Menge aller Teilmengen von $Q = 2^Q$.
Alternative: Relation $\delta \subseteq Q \times \Sigma \times Q$.

Beispiel



δ	0	1
p	$\{p\}$	$\{p, q\}$
q	$\{r\}$	$\{r\}$
r	\emptyset	\emptyset

Erweiterung von $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$
auf $\bar{\delta} : \mathcal{P}(Q) \times \Sigma \rightarrow \mathcal{P}(Q)$ definiert durch

$$\bar{\delta}(S, a) := \bigcup_{q \in S} \delta(q, a)$$

Es folgt: $\hat{\delta} : \mathcal{P}(Q) \times \Sigma^* \rightarrow \mathcal{P}(Q)$

Erweiterung von $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$
auf $\bar{\delta} : \mathcal{P}(Q) \times \Sigma \rightarrow \mathcal{P}(Q)$ definiert durch

$$\bar{\delta}(S, a) := \bigcup_{q \in S} \delta(q, a)$$

Es folgt: $\hat{\delta} : \mathcal{P}(Q) \times \Sigma^* \rightarrow \mathcal{P}(Q)$

Intuition:

$\hat{\delta}(S, w)$ ist Menge aller Zustände,
die sich von einem Zustand in S aus mit w erreichen lassen.

Erweiterung von $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$
auf $\bar{\delta} : \mathcal{P}(Q) \times \Sigma \rightarrow \mathcal{P}(Q)$ definiert durch

$$\bar{\delta}(S, a) := \bigcup_{q \in S} \delta(q, a)$$

Es folgt: $\hat{\delta} : \mathcal{P}(Q) \times \Sigma^* \rightarrow \mathcal{P}(Q)$

Intuition:

$\hat{\delta}(S, w)$ ist Menge aller Zustände,
die sich von einem Zustand in S aus mit w erreichen lassen.

Die von $N = (Q, \Sigma, \delta, q_0, F)$ **akzeptierte** Sprache ist

$$L(N) := \{w \in \Sigma^* \mid \hat{\delta}(\{q_0\}, w) \cap F \neq \emptyset\}$$

Erweiterung von $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$
auf $\bar{\delta} : \mathcal{P}(Q) \times \Sigma \rightarrow \mathcal{P}(Q)$ definiert durch

$$\bar{\delta}(S, a) := \bigcup_{q \in S} \delta(q, a)$$

Es folgt: $\hat{\delta} : \mathcal{P}(Q) \times \Sigma^* \rightarrow \mathcal{P}(Q)$

Intuition:

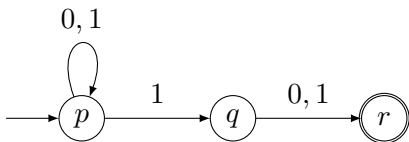
$\hat{\delta}(S, w)$ ist Menge aller Zustände,
die sich von einem Zustand in S aus mit w erreichen lassen.

Die von $N = (Q, \Sigma, \delta, q_0, F)$ **akzeptierte** Sprache ist

$$L(N) := \{w \in \Sigma^* \mid \hat{\delta}(\{q_0\}, w) \cap F \neq \emptyset\}$$

Oft schreiben wir nur δ statt $\bar{\delta}$ und $\hat{\delta}$ statt $\hat{\delta}$
(vgl. overloading in Programmiersprachen).

Beispiel



$$\begin{aligned} & \hat{\delta}(\{p, q\}, 10) \\ = & \hat{\delta}(\bar{\delta}(\{p, q\}, 1), 0) \\ = & \hat{\delta}(\delta(p, 1) \cup \delta(q, 1), 0) \\ = & \hat{\delta}(\{p, q, r\}, 0) \\ = & \bar{\delta}(\{p, q, r\}, 0) \\ = & \delta(p, 0) \cup \delta(q, 0) \cup \delta(r, 0) \\ = & \{p\} \cup \{r\} \cup \emptyset = \{p, r\} \end{aligned}$$

Satz 3.9

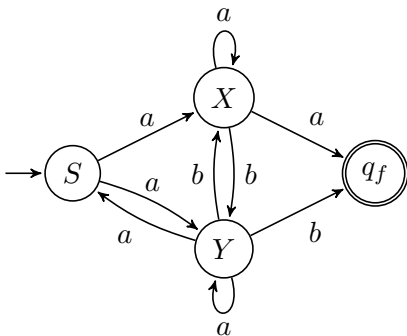
Für jede rechtslineare Grammatik G gibt es einen NFA M mit $L(G) = L(M)$.

Satz 3.9

Für jede rechtslineare Grammatik G gibt es einen NFA M mit $L(G) = L(M)$.

Beispiel

$S \rightarrow aX \mid aY$ $X \rightarrow aX \mid bY \mid a$ $Y \rightarrow aS \mid bX \mid aY \mid b$

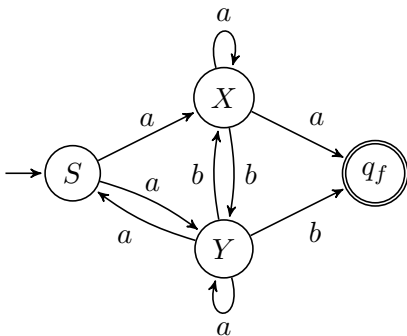


Satz 3.9

Für jede rechtslineare Grammatik G gibt es einen NFA M mit $L(G) = L(M)$.

Beispiel

$S \rightarrow aX \mid aY$ $X \rightarrow aX \mid bY \mid a$ $Y \rightarrow aS \mid bX \mid aY \mid b$



Aufgabe: Und wenn $S \rightarrow aX \mid aY \mid \epsilon$?

Beweis:

Sei $G = (V, \Sigma, P, S)$ eine rechtslineare Grammatik ohne die Produktion $S \rightarrow \epsilon$. Definiere den NFA $A = (Q, \Sigma, \delta, q_0, F)$ mit

- $Q = V \cup \{q_f\}$ (wobei $q_f \notin V$)
- $Y \in \delta(X, a)$ gdw $(X \rightarrow aY) \in P$
- $q_f \in \delta(X, a)$ gdw $(X \rightarrow a) \in P$
- $q_0 = S$
- $F = \{q_f\}$

Beweis:

Sei $G = (V, \Sigma, P, S)$ eine rechtslineare Grammatik ohne die Produktion $S \rightarrow \epsilon$. Definiere den NFA $A = (Q, \Sigma, \delta, q_0, F)$ mit

- $Q = V \cup \{q_f\}$ (wobei $q_f \notin V$)
- $Y \in \delta(X, a)$ gdw $(X \rightarrow aY) \in P$
- $q_f \in \delta(X, a)$ gdw $(X \rightarrow a) \in P$
- $q_0 = S$
- $F = \{q_f\}$

Es gilt (Induktion über n):

$$S \rightarrow a_1 X_1 \rightarrow_G a_1 a_2 X_2 \rightarrow_G \cdots \rightarrow_G a_1 \dots a_{n-1} X_{n-1} \rightarrow_G a_1 \dots a_n$$

ist eine Ableitung von G gdw

$$q_f \in \delta(S, a_1 \dots a_n) .$$

Damit gilt $L(G) = L(A)$. (Fall mit $S \rightarrow \epsilon$: Aufgabe)



Lösung der Aufgabe: Wenn die Grammatik die Produktion $S \rightarrow \epsilon$ enthält, dann setze $F = \{S, q_f\}$.

Aufgabe: Sei $G = (V, T, P, S)$ eine rechtslineare Grammatik, die die Produktion $S \rightarrow \epsilon$ nicht enthält. Definiere die Grammatik $G' = (V, T, P \cup \{S \rightarrow \epsilon\}, S)$ (d.h., wir fügen $S \rightarrow \epsilon$ zu P hinzu).

- Gilt immer $L(G') = L(G) \cup \{\epsilon\}$?
- Geben Sie eine Grammatik G'' mit $L(G'') = L(G) \cup \{\epsilon\}$ an.

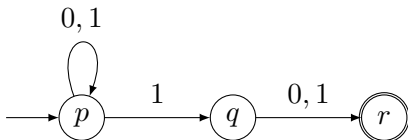
Satz 3.10

Für jeden NFA N gibt es einen DFA M mit $L(N) = L(M)$.

Satz 3.10

Für jeden NFA N gibt es einen DFA M mit $L(N) = L(M)$.

Beispiel



Beweis:

Sei $N = (Q, \Sigma, \delta, q_0, F)$ ein NFA.

Definiere den DFA $M = (\mathcal{P}(Q), \Sigma, \bar{\delta}, \{q_0\}, F_M)$:

$$F_M := \{S \subseteq Q \mid S \cap F \neq \emptyset\}$$

Beweis:

Sei $N = (Q, \Sigma, \delta, q_0, F)$ ein NFA.

Definiere den DFA $M = (\mathcal{P}(Q), \Sigma, \bar{\delta}, \{q_0\}, F_M)$:

$$F_M := \{S \subseteq Q \mid S \cap F \neq \emptyset\}$$

Dann gilt:

$$\begin{aligned} w \in L(N) &\Leftrightarrow \hat{\delta}(\{q_0\}, w) \cap F \neq \emptyset && \text{Def.} \\ &\Leftrightarrow \hat{\delta}(\{q_0\}, w) \in F_M && \text{Def.} \\ &\Leftrightarrow w \in L(M) && \text{Def.} \end{aligned}$$



Beweis:

Sei $N = (Q, \Sigma, \delta, q_0, F)$ ein NFA.

Definiere den DFA $M = (\mathcal{P}(Q), \Sigma, \hat{\delta}, \{q_0\}, F_M)$:

$$F_M := \{S \subseteq Q \mid S \cap F \neq \emptyset\}$$

Dann gilt:

$$w \in L(N) \Leftrightarrow \hat{\delta}(\{q_0\}, w) \cap F \neq \emptyset \quad \text{Def.}$$

$$\Leftrightarrow \hat{\delta}(\{q_0\}, w) \in F_M \quad \text{Def.}$$

$$\Leftrightarrow w \in L(M) \quad \text{Def.} \quad \square$$

Dies nennt man die **Potenzmengen-** oder **Teilmengenkonstruktion**.

In der Praxis werden nur die aus q_0 erreichbaren Zuständen konstruiert.

Trotzdem: Für einen NFA mit n Zuständen kann der entsprechende DFA bis zu 2^n Zustände haben.

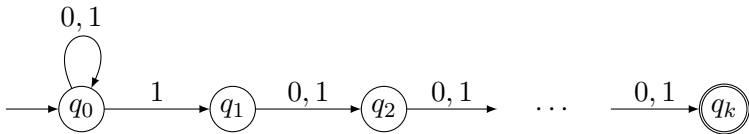
Beispiel 3.11

$$L_k := \{w \in \{0, 1\}^* \mid \text{das } k\text{-letzte Bit von } w \text{ ist } 1\}$$

Beispiel 3.11

$$L_k := \{w \in \{0, 1\}^* \mid \text{das } k\text{-letzte Bit von } w \text{ ist } 1\}$$

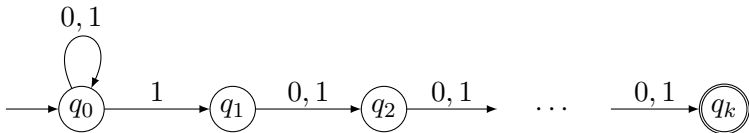
Ein NFA für diese Sprache:



Beispiel 3.11

$$L_k := \{w \in \{0, 1\}^* \mid \text{das } k\text{-letzte Bit von } w \text{ ist } 1\}$$

Ein NFA für diese Sprache:

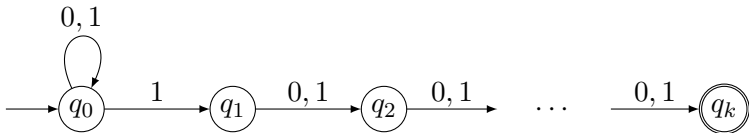


Die Potenzmengenkonstruktion liefert einen DFA für L_k mit 2^k Zuständen. Geht es kompakter?

Beispiel 3.11

$$L_k := \{w \in \{0, 1\}^* \mid \text{das } k\text{-letzte Bit von } w \text{ ist } 1\}$$

Ein NFA für diese Sprache:



Die Potenzmengenkonstruktion liefert einen DFA für L_k mit 2^k Zuständen. Geht es kompakter?

Lemma 3.12

Jeder DFA M mit $L(M) = L_k$ hat mindestens 2^k Zuständen.

Im *schlimmsten* Fall ist ein exponentieller Sprung unvermeidlich.

Beweis:

Indirekt. Sei M ein DFA mit $< 2^k$ Zuständen, so dass $L(M) = L_k$.

Beweis:

Indirekt. Sei M ein DFA mit $< 2^k$ Zuständen, so dass $L(M) = L_k$.

- Zuerst zeigen wir für alle $w_1, w_2 \in \{0, 1\}^k$:
wenn $w_1 \neq w_2$ dann $\hat{\delta}(q_0, w_1) \neq \hat{\delta}(q_0, w_2)$.
Annahme: Es gibt $w_1, w_2 \in \{0, 1\}^k$ mit $w_1 \neq w_2$ aber
 $\hat{\delta}(q_0, w_1) = \hat{\delta}(q_0, w_2)$. Wir leiten einen Widerspruch ab:

Beweis:

Indirekt. Sei M ein DFA mit $< 2^k$ Zuständen, so dass $L(M) = L_k$.

- Zuerst zeigen wir für alle $w_1, w_2 \in \{0, 1\}^k$:
wenn $w_1 \neq w_2$ dann $\hat{\delta}(q_0, w_1) \neq \hat{\delta}(q_0, w_2)$.
Annahme: Es gibt $w_1, w_2 \in \{0, 1\}^k$ mit $w_1 \neq w_2$ aber
 $\hat{\delta}(q_0, w_1) = \hat{\delta}(q_0, w_2)$. Wir leiten einen Widerspruch ab:
- Sei $w_1 = wa_i \dots a_k$ und $w_2 = wb_i \dots b_k$ mit $a_i \neq b_i$

OE sei $a_i = 1, b_i = 0$.

Es gilt einerseits: $w_1 0^{i-1} = wa_i \dots a_k 0^{i-1} \in L_k$
 $w_2 0^{i-1} = wb_i \dots b_k 0^{i-1} \notin L_k$

Aber es gilt auch: $\hat{\delta}(q_0, w_1 0^{i-1}) = \hat{\delta}(\hat{\delta}(q_0, w_1), 0^{i-1}) =$
 $\hat{\delta}(\hat{\delta}(q_0, w_2), 0^{i-1}) = \hat{\delta}(q_0, w_2 0^{i-1}) \quad \color{red}{\nabla}$

Beweis:

Indirekt. Sei M ein DFA mit $< 2^k$ Zuständen, so dass $L(M) = L_k$.

- Zuerst zeigen wir für alle $w_1, w_2 \in \{0, 1\}^k$:
wenn $w_1 \neq w_2$ dann $\hat{\delta}(q_0, w_1) \neq \hat{\delta}(q_0, w_2)$.
Annahme: Es gibt $w_1, w_2 \in \{0, 1\}^k$ mit $w_1 \neq w_2$ aber
 $\hat{\delta}(q_0, w_1) = \hat{\delta}(q_0, w_2)$. Wir leiten einen Widerspruch ab:
- Sei $w_1 = wa_i \dots a_k$ und $w_2 = wb_i \dots b_k$ mit $a_i \neq b_i$
OE sei $a_i = 1, b_i = 0$.
Es gilt einerseits: $w_1 0^{i-1} = wa_i \dots a_k 0^{i-1} \in L_k$
 $w_2 0^{i-1} = wb_i \dots b_k 0^{i-1} \notin L_k$
Aber es gilt auch: $\hat{\delta}(q_0, w_1 0^{i-1}) = \hat{\delta}(\hat{\delta}(q_0, w_1), 0^{i-1}) =$
 $\hat{\delta}(\hat{\delta}(q_0, w_2), 0^{i-1}) = \hat{\delta}(q_0, w_2 0^{i-1}) \quad \color{red}{\nabla}$
- Es folgt: M hat mindestens 2^k Zustände.



Satz 3.13

Für jeden DFA M gibt es eine rechtslineare Grammatik G mit $L(M) = L(G)$.

Beweis:

Sei $M = (Q, \Sigma, \delta, q_0, F)$. Die Grammatik $G = (V, T, P, S)$ mit

- $V = Q, T = \Sigma, S = q_0,$
- $(q_1 \rightarrow aq_2) \in P$ gdw $\delta(q_1, a) = q_2,$
- $(q_1 \rightarrow a) \in P$ gdw $\delta(q_1, a) \in F,$ und
- $(q_0 \rightarrow \epsilon) \in P$ gdw $q_0 \in F,$

ist von Typ 3 und erfüllt $L(M) = L(G)$. □

3.3 NFAs mit ϵ -Übergängen

Grammatiken von Programmiersprachen enthalten viele Produktionen der Gestalt $A \rightarrow B$.

Beispiel 3.14

Ein kleines Fragment der Grammatik von Java:

Literal:

```
IntegerLiteral  
FloatingPointLiteral  
CharacterLiteral  
StringLiteral  
BooleanLiteral  
NullLiteral
```

Wir suchen Recognizer für Typ 3 Grammatiken, die auch solche Produktionen enthalten.

Definition 3.15

Ein NFA mit ϵ -Übergängen (auch ϵ -NFA) ist ein NFA mit einem speziellen Symbol $\epsilon \notin \Sigma$ und mit

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q) .$$

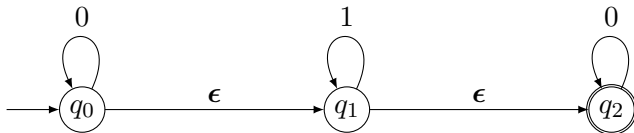
Ein ϵ -Übergang darf ausgeführt werden, ohne dass ein Eingabezeichen gelesen wird.

Definition 3.15

Ein NFA mit ϵ -Übergängen (auch ϵ -NFA) ist ein NFA mit einem speziellen Symbol $\epsilon \notin \Sigma$ und mit

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q) .$$

Ein ϵ -Übergang darf ausgeführt werden, ohne dass ein Eingabezeichen gelesen wird.



Akzeptiert: ϵ , 00, 11, ...

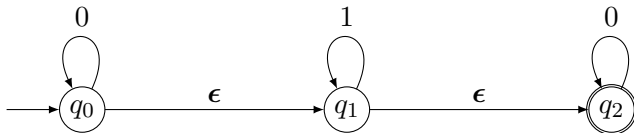
Nicht akzeptiert: 101, ...

Definition 3.15

Ein NFA mit ϵ -Übergängen (auch ϵ -NFA) ist ein NFA mit einem speziellen Symbol $\epsilon \notin \Sigma$ und mit

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q) .$$

Ein ϵ -Übergang darf ausgeführt werden, ohne dass ein Eingabezeichen gelesen wird.



Akzeptiert: ϵ , 00, 11, ... Nicht akzeptiert: 101, ...

Bemerkung: $\epsilon \neq \epsilon$; ϵ ist ein einzelnes Symbol, ϵ das leere Wort.

Lemma 3.16

Für jeden ϵ -NFA N gibt es einen NFA N' mit $L(N) = L(N')$.

Lemma 3.16

Für jeden ϵ -NFA N gibt es einen NFA N' mit $L(N) = L(N')$.

Beweis:

Sei $N = (Q, \Sigma, \delta, q_0, F)$ ein ϵ -NFA. Wir definieren den NFA $N' = (Q, \Sigma, \delta', q_0, F')$ mit folgenden Definitionen für δ' und F' :

- $\delta' : Q \times \Sigma \rightarrow \mathcal{P}(Q)$

$$\delta'(q, a) := \bigcup_{i \geq 0, j \geq 0} \hat{\delta}(\{q\}, \epsilon^i a \epsilon^j).$$

- Falls N das leere Wort ϵ akzeptiert, also falls

$$\exists i \geq 0. \hat{\delta}(\{q_0\}, \epsilon^i) \cap F \neq \emptyset$$

dann setze $F' := F \cup \{q_0\}$, sonst setze $F' := F$.



Lemma 3.16

Für jeden ϵ -NFA N gibt es einen NFA N' mit $L(N) = L(N')$.

Beweis:

Sei $N = (Q, \Sigma, \delta, q_0, F)$ ein ϵ -NFA. Wir definieren den NFA $N' = (Q, \Sigma, \delta', q_0, F')$ mit folgenden Definitionen für δ' und F' :

- $\delta' : Q \times \Sigma \rightarrow \mathcal{P}(Q)$

$$\delta'(q, a) := \bigcup_{i \geq 0, j \geq 0} \hat{\delta}(\{q\}, \epsilon^i a \epsilon^j).$$

- Falls N das leere Wort ϵ akzeptiert, also falls

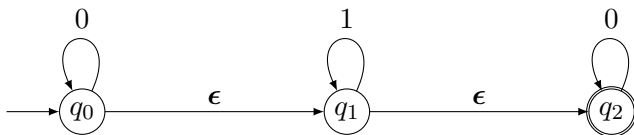
$$\exists i \geq 0. \hat{\delta}(\{q_0\}, \epsilon^i) \cap F \neq \emptyset$$

dann setze $F' := F \cup \{q_0\}$, sonst setze $F' := F$.

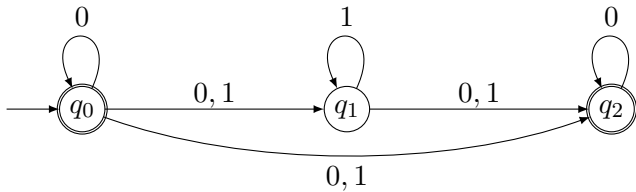
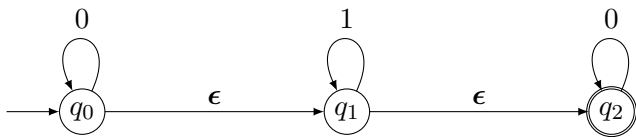


Ab jetzt: “ ϵ -Übergang” und “ ϵ -NFA”.

Beispiel 3.17



Beispiel 3.17



Fazit: Die Automatentypen

- DFA
- NFA
- ϵ -NFA

sind gleich mächtig und Erkennen die Sprachen der Grammatiken mit Produktionen folgender Gestalt:

- $X \rightarrow aY$
- $X \rightarrow a$
- $X \rightarrow Y$
- $X \rightarrow \epsilon$

3.4 Reguläre Ausdrücke

Reguläre Ausdrücke sind eine alternative Notation für die Definition von formalen Sprachen.

Werden oft in Kombination mit Grammatiken verwendet.

3.4 Reguläre Ausdrücke

Reguläre Ausdrücke sind eine alternative Notation für die Definition von formalen Sprachen.

Werden oft in Kombination mit Grammatiken verwendet.

Beispiel 3.18

Eine Fragment der Grammatik für Python
(<https://docs.python.org/2/reference/grammar.html>)

```
simple_stmt: small_stmt (';' small_stmt)* [';'] NEWLINE
```

Steht für eine unendliche Menge von Produktionen:

```
simple_stmt → small_stmt NEWLINE  
simple_stmt → small_stmt ';' NEWLINE  
simple_stmt → small_stmt ';' small_stmt NEWLINE  
simple_stmt → small_stmt ';' small_stmt ';' NEWLINE  
...
```

Definition 3.19

Reguläre Ausdrücke (*regular expressions*, REs) sind induktiv definiert:

- \emptyset ist ein regulärer Ausdruck.
- ϵ ist ein regulärer Ausdruck.
- Für jedes $a \in \Sigma$ ist a ein regulärer Ausdruck.
- Wenn α und β reguläre Ausdrücke sind, dann auch
 - $\alpha\beta$
 - $\alpha \mid \beta$ (oft $\alpha + \beta$ geschrieben)
 - α^*
- Nichts sonst ist ein regulärer Ausdruck.

Definition 3.19

Reguläre Ausdrücke (*regular expressions*, REs) sind induktiv definiert:

- \emptyset ist ein regulärer Ausdruck.
- ϵ ist ein regulärer Ausdruck.
- Für jedes $a \in \Sigma$ ist a ein regulärer Ausdruck.
- Wenn α und β reguläre Ausdrücke sind, dann auch
 - $\alpha\beta$
 - $\alpha \mid \beta$ (oft $\alpha + \beta$ geschrieben)
 - α^*
- Nichts sonst ist ein regulärer Ausdruck.

* ist die Kleene'sche Iteration (Kleene iteration, Kleene star).

Bindungsstärke: * bindet stärker als Konkatenation stärker als |

- $ab^* = a(b^*) \neq (ab)^*$
- $ab \mid c = (ab) \mid c \neq a(b \mid c)$

Definition 3.20

Zu einem regulären Ausdruck γ ist die zugehörige Sprache $L(\gamma)$ rekursiv definiert:

- $L(\emptyset) = \emptyset$
- $L(\epsilon) = \{\epsilon\}$
- $L(a) = \{a\}$
- $L(\alpha\beta) = L(\alpha)L(\beta)$
- $L(\alpha \mid \beta) = L(\alpha) \cup L(\beta)$
- $L(\alpha^*) = L(\alpha)^*$

Beispiel 3.21

Sei das zugrunde liegende Alphabet $\Sigma = \{0, 1\}$.

- Alle Wörter, die mit 00 enden:

$$(0|1)^*00$$

Beispiel 3.21

Sei das zugrunde liegende Alphabet $\Sigma = \{0, 1\}$.

- Alle Wörter, die mit 00 enden:

$$(0|1)^*00$$

- Alle Wörter gerader Länge, in denen 0 und 1 alternieren:

$$(01)^* \mid (10)^*$$

Beispiel 3.21

Sei das zugrunde liegende Alphabet $\Sigma = \{0, 1\}$.

- Alle Wörter, die mit 00 enden:

$$(0|1)^*00$$

- Alle Wörter gerader Länge, in denen 0 und 1 alternieren:

$$(01)^* \mid (10)^*$$

- Alle Wörter, die eine gerade Anzahl von 1'en enthalten:

$$(0^*10^*1)^*0^*$$

Beispiel 3.21

Sei das zugrunde liegende Alphabet $\Sigma = \{0, 1\}$.

- Alle Wörter, die mit 00 enden:

$$(0|1)^*00$$

- Alle Wörter gerader Länge, in denen 0 und 1 alternieren:

$$(01)^* \mid (10)^*$$

- Alle Wörter, die eine gerade Anzahl von 1'en enthalten:

$$(0^*10^*1)^*0^*$$

- Alle Wörter, die die Binärdarstellung einer durch 3 teilbaren Zahl darstellen, also

0, 11, 110, 1001, 1100, 1111, 10010, ...

Beispiel 3.21

Sei das zugrunde liegende Alphabet $\Sigma = \{0, 1\}$.

- Alle Wörter, die mit 00 enden:

$$(0|1)^*00$$

- Alle Wörter gerader Länge, in denen 0 und 1 alternieren:

$$(01)^* \mid (10)^*$$

- Alle Wörter, die eine gerade Anzahl von 1'en enthalten:

$$(0^*10^*1)^*0^*$$

- Alle Wörter, die die Binärdarstellung einer durch 3 teilbaren Zahl darstellen, also

$$0, 11, 110, 1001, 1100, 1111, 10010, \dots$$

Hausaufgabe!

Beispiel 3.22

Gleitkommazahlen: $\Sigma = \{+, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, .\}$

$$(+ \mid - \mid \epsilon)(DD^* \mid DD^*.D^* \mid D^*.DD^*)$$

wobei $D = (0|1|2|3|4|5|6|7|8|9)$

Erweiterte reguläre Ausdrücke in UNIX:

Erweiterte reguläre Ausdrücke in UNIX:

$$\cdot = a_1 | \dots | a_n \text{ wobei } \Sigma = \{a_1, \dots, a_n\}$$

$$[a_1 \dots a_n] = a_1 | \dots | a_n$$

$$[\hat{a}_1 \dots a_n] = b_1 | \dots | b_m \text{ wobei } \{b_1, \dots, b_m\} = \Sigma \setminus \{a_1, \dots, a_n\}$$

$$\alpha? = \epsilon | \alpha$$

$$\alpha+ = \alpha\alpha^*$$

$$\alpha\{n\} = \alpha \dots \alpha \text{ (} n \text{ copies)}$$

Strukturelle Induktion

Da die regulären Ausdrücke induktiv definiert sind, gilt für sie das Prinzip der **strukturellen Induktion**:

Um zu beweisen, dass Eigenschaft P für alle regulären Ausdrücke γ gilt, also $P(\gamma)$, beweise

- $P(\emptyset)$
- $P(\epsilon)$
- $P(a)$ für alle $a \in \Sigma$
- $P(\alpha) \wedge P(\beta) \Rightarrow P(\alpha\beta)$
- $P(\alpha) \wedge P(\beta) \Rightarrow P(\alpha \mid \beta)$
- $P(\alpha) \Rightarrow P(\alpha^*)$

Satz 3.23 (Kleene 1956)

Eine Sprache $L \subseteq \Sigma^$ ist genau dann durch einen regulären Ausdruck darstellbar, wenn sie regulär ist.*

Satz 3.23 (Kleene 1956)

Eine Sprache $L \subseteq \Sigma^$ ist genau dann durch einen regulären Ausdruck darstellbar, wenn sie regulär ist.*

Beweis:

“ \implies ”:

Sei $L = L(\gamma)$.

Wir konstruieren einen ϵ -NFA N mit $L = L(N)$ mit Hilfe struktureller Induktion über γ .

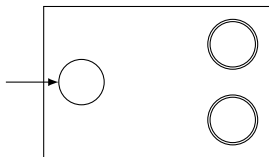
Die Basisfälle $\gamma = \emptyset$, $\gamma = \epsilon$, und $\gamma = a \in \Sigma$ sind offensichtlich.

Fall $\gamma = \alpha\beta$:

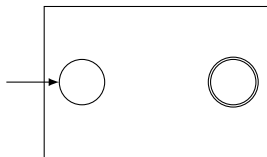
Nach Induktionsannahme können wir ϵ -NFAs N_α und N_β konstruieren mit $L(N_\alpha) = L(\alpha)$ und $L(N_\beta) = L(\beta)$.

Fall $\gamma = \alpha\beta$:

Nach Induktionsannahme können wir ϵ -NFAs N_α und N_β konstruieren mit $L(N_\alpha) = L(\alpha)$ und $L(N_\beta) = L(\beta)$.



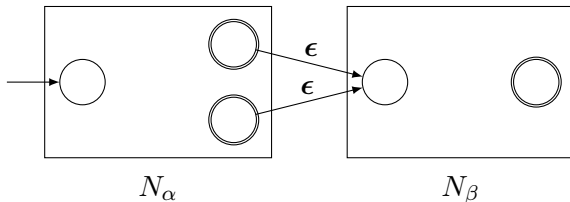
N_α



N_β

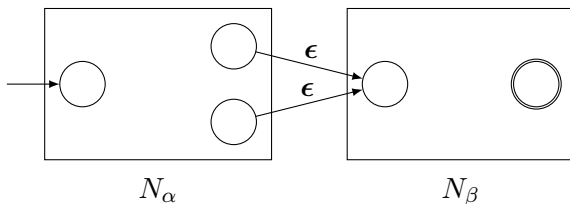
Fall $\gamma = \alpha\beta$:

Nach Induktionsannahme können wir ϵ -NFAs N_α und N_β konstruieren mit $L(N_\alpha) = L(\alpha)$ und $L(N_\beta) = L(\beta)$.



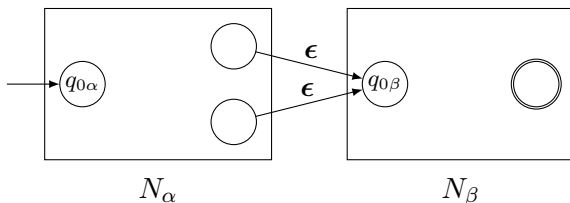
Fall $\gamma = \alpha\beta$:

Nach Induktionsannahme können wir ϵ -NFAs N_α und N_β konstruieren mit $L(N_\alpha) = L(\alpha)$ und $L(N_\beta) = L(\beta)$.



Fall $\gamma = \alpha\beta$:

Nach Induktionsannahme können wir ϵ -NFAs N_α und N_β konstruieren mit $L(N_\alpha) = L(\alpha)$ und $L(N_\beta) = L(\beta)$.



Formal:

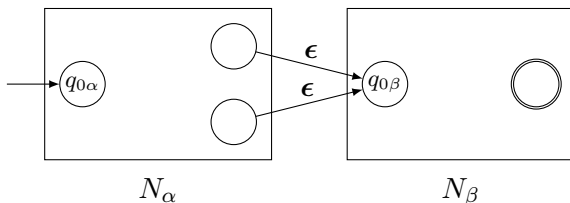
$$N_\alpha = (Q_\alpha, \Sigma, \delta_\alpha, q_{0\alpha}, F_\alpha)$$

$$N_\beta = (Q_\beta, \Sigma, \delta_\beta, q_{0\beta}, F_\beta) \quad (\text{mit } Q_\alpha \cap Q_\beta = \emptyset)$$

$$N_{\alpha\beta} := (Q_\alpha \cup Q_\beta, \Sigma, \delta, q_{0\alpha}, F_\beta)$$

Fall $\gamma = \alpha\beta$:

Nach Induktionsannahme können wir ϵ -NFAs N_α und N_β konstruieren mit $L(N_\alpha) = L(\alpha)$ und $L(N_\beta) = L(\beta)$.



Formal:

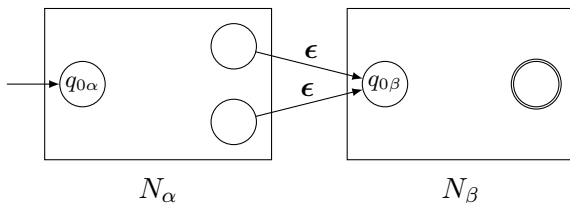
$$N_\alpha = (Q_\alpha, \Sigma, \delta_\alpha, q_{0\alpha}, F_\alpha)$$

$$N_\beta = (Q_\beta, \Sigma, \delta_\beta, q_{0\beta}, F_\beta) \quad (\text{mit } Q_\alpha \cap Q_\beta = \emptyset)$$

$$N_{\alpha\beta} := (Q_\alpha \cup Q_\beta, \Sigma, \delta, q_{0\alpha}, F_\beta)$$

Fall $\gamma = \alpha\beta$:

Nach Induktionsannahme können wir ϵ -NFAs N_α und N_β konstruieren mit $L(N_\alpha) = L(\alpha)$ und $L(N_\beta) = L(\beta)$.



Formal:

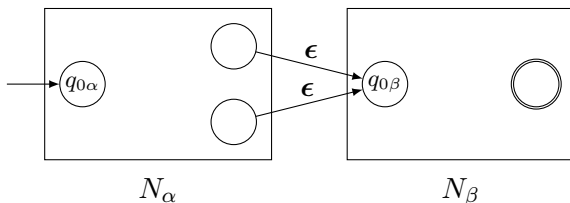
$$N_\alpha = (Q_\alpha, \Sigma, \delta_\alpha, q_{0\alpha}, F_\alpha)$$

$$N_\beta = (Q_\beta, \Sigma, \delta_\beta, q_{0\beta}, F_\beta) \quad (\text{mit } Q_\alpha \cap Q_\beta = \emptyset)$$

$$N_{\alpha\beta} := (Q_\alpha \cup Q_\beta, \Sigma, \delta, q_{0\alpha}, F_\beta)$$

Fall $\gamma = \alpha\beta$:

Nach Induktionsannahme können wir ϵ -NFAs N_α und N_β konstruieren mit $L(N_\alpha) = L(\alpha)$ und $L(N_\beta) = L(\beta)$.



Formal:

$$N_\alpha = (Q_\alpha, \Sigma, \delta_\alpha, q_{0\alpha}, F_\alpha)$$

$$N_\beta = (Q_\beta, \Sigma, \delta_\beta, q_{0\beta}, F_\beta) \quad (\text{mit } Q_\alpha \cap Q_\beta = \emptyset)$$

$$N_{\alpha\beta} := (Q_\alpha \cup Q_\beta, \Sigma, \delta, q_{0\alpha}, F_\beta)$$

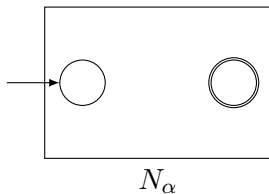
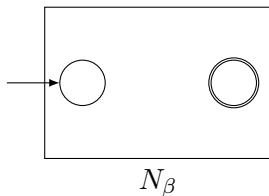
$$\delta := \delta_\alpha \cup \delta_\beta \cup \{(f, \epsilon) \mapsto \{q_{0\beta}\} \mid f \in F_\alpha\}$$

Fall $\gamma = \alpha \mid \beta$:

Nach Induktionsannahme können wir ϵ -NFAs N_α und N_β konstruieren mit $L(N_\alpha) = L(\alpha)$ und $L(N_\beta) = L(\beta)$.

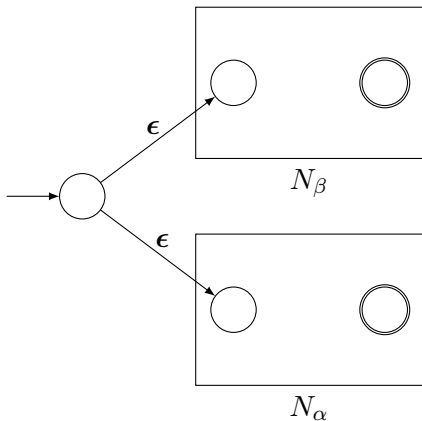
Fall $\gamma = \alpha \mid \beta$:

Nach Induktionsannahme können wir ϵ -NFAs N_α und N_β konstruieren mit $L(N_\alpha) = L(\alpha)$ und $L(N_\beta) = L(\beta)$.



Fall $\gamma = \alpha \mid \beta$:

Nach Induktionsannahme können wir ϵ -NFAs N_α und N_β konstruieren mit $L(N_\alpha) = L(\alpha)$ und $L(N_\beta) = L(\beta)$.



Fall $\gamma = \alpha^*$:

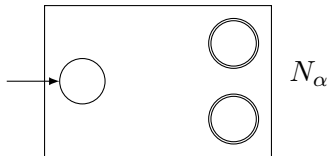
Nach Induktionsannahme können wir einen ϵ -NFA N_α konstruieren mit

$$L(N_\alpha) = L(\alpha) .$$

Fall $\gamma = \alpha^*$:

Nach Induktionsannahme können wir einen ϵ -NFA N_α konstruieren mit

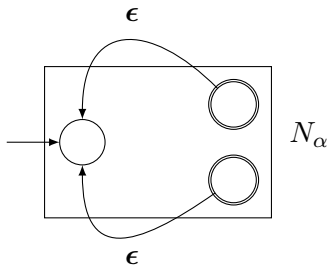
$$L(N_\alpha) = L(\alpha) .$$



Fall $\gamma = \alpha^*$:

Nach Induktionsannahme können wir einen ϵ -NFA N_α konstruieren mit

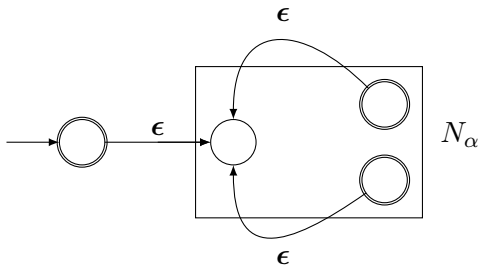
$$L(N_\alpha) = L(\alpha) .$$



Fall $\gamma = \alpha^*$:

Nach Induktionsannahme können wir einen ϵ -NFA N_α konstruieren mit

$$L(N_\alpha) = L(\alpha) .$$



“ \Leftarrow ”:

Sei $M = (Q, \Sigma, \delta, q_1, F)$ ein DFA.

Wir konstruieren einen RE γ mit $L(M) = L(\gamma)$.

“ \Leftarrow ”:

Sei $M = (Q, \Sigma, \delta, q_1, F)$ ein DFA.

Wir konstruieren einen RE γ mit $L(M) = L(\gamma)$.

Sei $Q = \{q_1, \dots, q_n\}$. Wir setzen

$R_{ij}^k := \{w \in \Sigma^* \mid \text{die Eingabe } w \text{ f\u00fchrt von } q_i \text{ in } q_j, \text{ wobei alle}$
Zwischenzust\u00e4nde (ohne ersten und letzten)
einen Index $\leq k$ haben }
}

“ \Leftarrow ”:

Sei $M = (Q, \Sigma, \delta, q_1, F)$ ein DFA.

Wir konstruieren einen RE γ mit $L(M) = L(\gamma)$.

Sei $Q = \{q_1, \dots, q_n\}$. Wir setzen

$R_{ij}^k := \{w \in \Sigma^* \mid \text{die Eingabe } w \text{ f\u00fchrt von } q_i \text{ in } q_j, \text{ wobei alle}$
Zwischenzust\u00e4nde (ohne ersten und letzten)
einen Index $\leq k$ haben }
}

Behauptung: F\u00fcr alle $i, j \in \{1, \dots, n\}$ und $k \in \{0, \dots, n\}$ k\u00f6nnen wir einen RE α_{ij}^k konstruieren mit $L(\alpha_{ij}^k) = R_{ij}^k$.

Bew.:

Induktion über k :

$k = 0$: Hier gilt

$$R_{ij}^0 = \begin{cases} \{a \in \Sigma \mid \delta(q_i, a) = q_j\}, & \text{falls } i \neq j \\ \{a \in \Sigma \mid \delta(q_i, a) = q_j\} \cup \{\epsilon\}, & \text{falls } i = j \end{cases}$$

Setze

$$\alpha_{ij}^0 := \begin{cases} a_1 \mid \dots \mid a_l & \text{falls } i \neq j \\ a_1 \mid \dots \mid a_l \mid \epsilon & \text{falls } i = j \end{cases}$$

wobei $\{a_1 \dots, a_l\} = \{a \in \Sigma \mid \delta(q_i, a) = q_j\}$.

Bew.:

Induktion über k :

$k \Rightarrow k + 1$: Hier gilt

$$R_{ij}^{k+1} = R_{ij}^k \cup R_{i(k+1)}^k (R_{(k+1)(k+1)}^k)^* R_{(k+1)j}^k$$

weil man jede Folge von Zahlen $\leq k + 1$ schreiben kann als

$$F_1, k + 1, F_2, k + 1, \dots, k + 1, F_m$$

wobei jede F_l Folge von Zahlen $\leq k$ ist.

Bew.:

Induktion über k :

$k \Rightarrow k + 1$: Hier gilt

$$R_{ij}^{k+1} = R_{ij}^k \cup R_{i(k+1)}^k (R_{(k+1)(k+1)}^k)^* R_{(k+1)j}^k$$

weil man jede Folge von Zahlen $\leq k + 1$ schreiben kann als

$$F_1, k + 1, F_2, k + 1, \dots, k + 1, F_m$$

wobei jede F_l Folge von Zahlen $\leq k$ ist.

Wir definieren rekursiv

$$\alpha_{ij}^{k+1} = \alpha_{ij}^k \mid \alpha_{i(k+1)}^k (\alpha_{(k+1)(k+1)}^k)^* \alpha_{(k+1)j}^k$$

Bew.:

Induktion über k :

$k \Rightarrow k + 1$: Hier gilt

$$R_{ij}^{k+1} = R_{ij}^k \cup R_{i(k+1)}^k (R_{(k+1)(k+1)}^k)^* R_{(k+1)j}^k$$

weil man jede Folge von Zahlen $\leq k + 1$ schreiben kann als

$$F_1, k + 1, F_2, k + 1, \dots, k + 1, F_m$$

wobei jede F_l Folge von Zahlen $\leq k$ ist.

Wir definieren rekursiv

$$\alpha_{ij}^{k+1} = \alpha_{ij}^k \mid \alpha_{i(k+1)}^k (\alpha_{(k+1)(k+1)}^k)^* \alpha_{(k+1)j}^k$$

Somit gilt

$$L(M) = L(\alpha_{1i_1}^n \mid \dots \mid \alpha_{1i_r}^n)$$

wobei $F = \{i_1, \dots, i_r\}$.



Unsere Konversionen auf einen Blick:

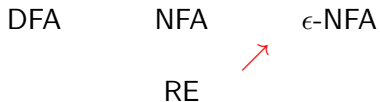
DFA

NFA

ϵ -NFA

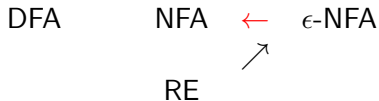
RE

Unsere Konversionen auf einen Blick:



RE \rightarrow ϵ -NFA: RE der Länge $n \rightsquigarrow O(n)$ Zustände

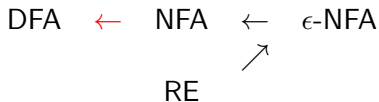
Unsere Konversionen auf einen Blick:



RE \rightarrow ϵ -NFA: RE der Länge $n \rightsquigarrow O(n)$ Zustände

ϵ -NFA \rightarrow NFA: $Q \rightsquigarrow Q$

Unsere Konversionen auf einen Blick:



RE \rightarrow ϵ -NFA: RE der Länge $n \rightsquigarrow O(n)$ Zustände

ϵ -NFA \rightarrow NFA: $Q \rightsquigarrow Q$

NFA \rightarrow DFA: n Zustände $\rightsquigarrow O(2^n)$ Zustände

Unsere Konversionen auf einen Blick:



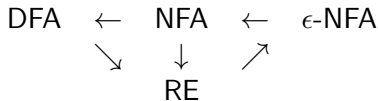
RE \rightarrow ϵ -NFA: RE der Länge $n \rightsquigarrow O(n)$ Zustände

ϵ -NFA \rightarrow NFA: $Q \rightsquigarrow Q$

NFA \rightarrow DFA: n Zustände $\rightsquigarrow O(2^n)$ Zustände

FA \rightarrow RE: n Zustände \rightsquigarrow RE der Länge $O(n4^n)$

Unsere Konversionen auf einen Blick:



RE → ε-NFA: RE der Länge $n \rightsquigarrow O(n)$ Zustände

ε-NFA → NFA: $Q \rightsquigarrow Q$

NFA → DFA: n Zustände $\rightsquigarrow O(2^n)$ Zustände

FA → RE: n Zustände \rightsquigarrow RE der Länge $O(n4^n)$

Beweis FA→RE: $m_k :=$ maximale Länge von α_{ij}^k

- $m_0 = c_1 |\Sigma|$
- $m_{k+1} = 4 * m_k + c_2$
- $m_k \in O(4^k)$
- Länge des Gesamtausdrucks: $O(n4^n)$

3.5 Abschlusseigenschaften regulärer Sprachen

Satz 3.24

Seien $R, R_1, R_2 \subseteq \Sigma^*$ reguläre Sprachen. Dann sind auch

$$R_1 R_2, R_1 \cup R_2, R^*, \overline{R} \quad (:= \Sigma^* \setminus R), R_1 \cap R_2, R_1 \setminus R_2$$

reguläre Sprachen.

3.5 Abschlusseigenschaften regulärer Sprachen

Satz 3.24

Seien $R, R_1, R_2 \subseteq \Sigma^*$ reguläre Sprachen. Dann sind auch

$$R_1 R_2, R_1 \cup R_2, R^*, \overline{R} \quad (:= \Sigma^* \setminus R), R_1 \cap R_2, R_1 \setminus R_2$$

reguläre Sprachen.

Beweis:

$R_1 R_2, R_1 \cup R_2$, und R^* : klar.

\overline{R} Sei $R = L(A)$ für einen DFA $A = (Q, \Sigma, \delta, q_0, F)$.

Betrachte $A' = (Q, \Sigma, \delta, q_0, Q \setminus F)$.

Dann ist $L(A') = \overline{L(A)} = \overline{R}$

$$R_1 \cap R_2 = \overline{\overline{R_1} \cup \overline{R_2}} \quad (\text{De Morgan})$$

$$R_1 \setminus R_2 = R_1 \cap \overline{R_2}$$



Bemerkung

Komplementierung (\overline{R}) durch Vertauschen von Endzuständen und Nicht-Endzuständen funktioniert nur bei DFAs, nicht bei NFAs!

Bemerkung

Komplementierung (\overline{R}) durch Vertauschen von Endzuständen und Nicht-Endzuständen funktioniert nur bei DFAs, nicht bei NFAs!

Bei NFAs:

Komplementierung erzwingt Determinierung.

Komplementierung ist (potenziell) teuer.

Die **Produkt-Konstruktion**: Durchschnitt direkt auf DFAs, ohne Umweg über de Morgan.

Die **Produkt-Konstruktion**: Durchschnitt direkt auf DFAs, ohne Umweg über de Morgan.

Beide DFAs laufen synchron parallel, Wort wird akzeptiert wenn *beide* akzeptieren.

Parallelismus = Kreuzprodukt der Zustandsräume

Die **Produkt-Konstruktion**: Durchschnitt direkt auf DFAs, ohne Umweg über de Morgan.

Beide DFAs laufen synchron parallel, Wort wird akzeptiert wenn *beide* akzeptieren.

Parallelismus = Kreuzprodukt der Zustandsräume

Satz 3.25

Sind $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ und $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ DFAs, dann ist der **Produkt-Automat**

$$M := (Q_1 \times Q_2, \Sigma, \delta, (s_1, s_2), F_1 \times F_2)$$
$$\delta((q_1, q_2), a) := (\delta_1(q_1, a), \delta_2(q_2, a))$$

ein DFA der $L(M_1) \cap L(M_2)$ akzeptiert.

Die **Produkt-Konstruktion**: Durchschnitt direkt auf DFAs, ohne Umweg über de Morgan.

Beide DFAs laufen synchron parallel, Wort wird akzeptiert wenn *beide* akzeptieren.

Parallelismus = Kreuzprodukt der Zustandsräume

Satz 3.25

Sind $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ und $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ DFAs, dann ist der **Produkt-Automat**

$$M := (Q_1 \times Q_2, \Sigma, \delta, (s_1, s_2), F_1 \times F_2)$$
$$\delta((q_1, q_2), a) := (\delta_1(q_1, a), \delta_2(q_2, a))$$

ein DFA der $L(M_1) \cap L(M_2)$ akzeptiert.

Erinnerung: $|Q_1 \times Q_2| = |Q_1| |Q_2|$.

Beweis:

Durch Induktion über w läßt sich zeigen:

$$\hat{\delta}((q_1, q_2), w) = (\hat{\delta}_1(q_1, w), \hat{\delta}_2(q_2, w)).$$

Beweis:

Durch Induktion über w läßt sich zeigen:

$$\hat{\delta}((q_1, q_2), w) = (\hat{\delta}_1(q_1, w), \hat{\delta}_2(q_2, w)).$$

Damit gilt:

$$\begin{aligned} & w \in L(M) \\ \Leftrightarrow & (\hat{\delta}((s_1, s_2), w) \in F_1 \times F_2 \\ \Leftrightarrow & (\hat{\delta}_1(s_1, w), \hat{\delta}_2(s_2, w)) \in F_1 \times F_2 \\ \Leftrightarrow & \hat{\delta}_1(s_1, w) \in F_1 \wedge \hat{\delta}_2(s_2, w) \in F_2 \\ \Leftrightarrow & w \in L(M_1) \wedge w \in L(M_2) \\ \Leftrightarrow & w \in L(M_1) \cap L(M_2) \end{aligned}$$



Beweis:

Durch Induktion über w läßt sich zeigen:

$$\hat{\delta}((q_1, q_2), w) = (\hat{\delta}_1(q_1, w), \hat{\delta}_2(q_2, w)).$$

Damit gilt:

$$\begin{aligned} & w \in L(M) \\ \Leftrightarrow & (\hat{\delta}((s_1, s_2), w) \in F_1 \times F_2 \\ \Leftrightarrow & (\hat{\delta}_1(s_1, w), \hat{\delta}_2(s_2, w)) \in F_1 \times F_2 \\ \Leftrightarrow & \hat{\delta}_1(s_1, w) \in F_1 \wedge \hat{\delta}_2(s_2, w) \in F_2 \\ \Leftrightarrow & w \in L(M_1) \wedge w \in L(M_2) \\ \Leftrightarrow & w \in L(M_1) \cap L(M_2) \end{aligned}$$



Funktioniert Durchschnitt durch Produkt auch für NFAs?

3.6 Rechnen mit regulären Ausdrücken

Definition 3.26

Zwei reguläre Ausdrücke sind **äquivalent** gdw sie die gleiche Sprache darstellen:

$$\alpha \equiv \beta \quad :\Leftrightarrow \quad L(\alpha) = L(\beta)$$

3.6 Rechnen mit regulären Ausdrücken

Definition 3.26

Zwei reguläre Ausdrücke sind **äquivalent** gdw sie die gleiche Sprache darstellen:

$$\alpha \equiv \beta \iff L(\alpha) = L(\beta)$$

Beispiel zum Unterschied von $=$ (syntaktische Identität) und \equiv (Bedeutungsäquivalenz):

$(\alpha \mid \beta) \equiv (\beta \mid \alpha)$ aber $(\alpha \mid \beta) \neq (\beta \mid \alpha)$.

Null und Eins:

Lemma 3.27

- $\emptyset \mid \alpha \equiv \alpha \mid \emptyset \equiv \alpha$
- $\emptyset \alpha \equiv \alpha \emptyset \equiv \emptyset$
- $\epsilon \alpha \equiv \alpha \epsilon \equiv \alpha$
- $\emptyset^* \equiv \epsilon$
- $\epsilon^* \equiv \epsilon$

Lemma 3.28

Assoziativität:

- $(\alpha \mid \beta) \mid \gamma \equiv \alpha \mid (\beta \mid \gamma)$
- $(\alpha\beta)\gamma \equiv \alpha(\beta\gamma)$

Kommutativität:

- $\alpha \mid \beta \equiv \beta \mid \alpha$

Distributivität:

- $\alpha(\beta \mid \gamma) \equiv \alpha\beta \mid \alpha\gamma$
- $(\alpha \mid \beta)\gamma \equiv \alpha\gamma \mid \beta\gamma$

Idempotenz:

- $\alpha \mid \alpha \equiv \alpha$

Stern:

Lemma 3.29

- $\epsilon \mid \alpha\alpha^* \equiv \alpha^*$
- $\alpha^*\alpha \equiv \alpha\alpha^*$
- $(\alpha^*)^* \equiv \alpha^*$

Stern:

Lemma 3.29

- $\epsilon \mid \alpha\alpha^* \equiv \alpha^*$
- $\alpha^*\alpha \equiv \alpha\alpha^*$
- $(\alpha^*)^* \equiv \alpha^*$

Beispiel 3.30

Herleitung einer Äquivalenz aus obigen Lemmas:

$$\epsilon \mid \alpha^*$$

Stern:

Lemma 3.29

- $\epsilon \mid \alpha\alpha^* \equiv \alpha^*$
- $\alpha^*\alpha \equiv \alpha\alpha^*$
- $(\alpha^*)^* \equiv \alpha^*$

Beispiel 3.30

Herleitung einer Äquivalenz aus obigen Lemmas:

$$\epsilon \mid \alpha^*$$

$$\equiv \epsilon \mid (\epsilon \mid \alpha\alpha^*) \quad \text{Stern Lemma}$$

Stern:

Lemma 3.29

- $\epsilon \mid \alpha\alpha^* \equiv \alpha^*$
- $\alpha^*\alpha \equiv \alpha\alpha^*$
- $(\alpha^*)^* \equiv \alpha^*$

Beispiel 3.30

Herleitung einer Äquivalenz aus obigen Lemmas:

$$\epsilon \mid \alpha^*$$

$$\equiv \epsilon \mid (\epsilon \mid \alpha\alpha^*) \quad \text{Stern Lemma}$$

$$\equiv (\epsilon \mid \epsilon) \mid \alpha\alpha^* \quad \text{Assoziativität}$$

Stern:

Lemma 3.29

- $\epsilon \mid \alpha\alpha^* \equiv \alpha^*$
- $\alpha^*\alpha \equiv \alpha\alpha^*$
- $(\alpha^*)^* \equiv \alpha^*$

Beispiel 3.30

Herleitung einer Äquivalenz aus obigen Lemmas:

$$\begin{aligned} & \epsilon \mid \alpha^* \\ \equiv & \epsilon \mid (\epsilon \mid \alpha\alpha^*) && \text{Stern Lemma} \\ \equiv & (\epsilon \mid \epsilon) \mid \alpha\alpha^* && \text{Assoziativität} \\ \equiv & \epsilon \mid \alpha\alpha^* && \text{Idempotenz} \end{aligned}$$

Stern:

Lemma 3.29

- $\epsilon \mid \alpha\alpha^* \equiv \alpha^*$
- $\alpha^*\alpha \equiv \alpha\alpha^*$
- $(\alpha^*)^* \equiv \alpha^*$

Beispiel 3.30

Herleitung einer Äquivalenz aus obigen Lemmas:

$$\begin{aligned} & \epsilon \mid \alpha^* \\ \equiv & \epsilon \mid (\epsilon \mid \alpha\alpha^*) && \text{Stern Lemma} \\ \equiv & (\epsilon \mid \epsilon) \mid \alpha\alpha^* && \text{Assoziativität} \\ \equiv & \epsilon \mid \alpha\alpha^* && \text{Idempotenz} \\ \equiv & \alpha^* && \text{Stern Lemma} \end{aligned}$$

Lässt sich jede gültige Äquivalenz $\alpha \equiv \beta$ aus den obigen Lemmas für \equiv herleiten?

Lässt sich jede gültige Äquivalenz $\alpha \equiv \beta$ aus den obigen Lemmas für \equiv herleiten?

Satz 3.31 (Redko 1964)

Es gibt keine endliche Menge von gültigen Äquivalenzen aus denen sich alle gültigen Äquivalenzen herleiten lassen.

Lässt sich jede gültige Äquivalenz $\alpha \equiv \beta$ aus den obigen Lemmas für \equiv herleiten?

Satz 3.31 (Redko 1964)

Es gibt keine endliche Menge von gültigen Äquivalenzen aus denen sich alle gültigen Äquivalenzen herleiten lassen.

Wenn man mehr als nur Äquivalenzen zulässt:



Arto Salomaa.

Two Complete Axiom Systems for the Algebra of Regular Events. Journal of the ACM, 1966.

Für Typ 3 Sprachen können wir automatisch effiziente Recognizer synthetisieren.

Sind Typ 3 Grammatiken mächtig genug für den Entwurf von Programmiersprachen?

Für Typ 3 Sprachen können wir automatisch effiziente Recognizer synthetisieren.

Sind Typ 3 Grammatiken mächtig genug für den Entwurf von Programmiersprachen?

- Jede Programmiersprache braucht arithmetische Ausdrücke.
- Die Grammatik für arithmetische Ausdrücke von Beispiel 2.11 ist nicht rechtslinear.
- Das zeigt jedoch noch nicht dass die Sprache der arithmetischen Ausdrücke nicht regulär ist: Es könnte eine andere rechtslineare Grammatik (vielleicht viel größer als die vom Beispiel 2.11), die die selbe Sprache erzeugt.

3.7 Pumping Lemma

Oder: *Wie zeigt man, dass eine Sprache nicht regulär ist?*

3.7 Pumping Lemma

Oder: *Wie zeigt man, dass eine Sprache nicht regulär ist?*

Satz 3.32 (Pumping Lemma für reguläre Sprachen)

Sei $R \subseteq \Sigma^$ regulär. Dann gibt es ein $n > 0$, so dass sich jedes $z \in R$ mit $|z| \geq n$ so in $z = uvw$ zerlegen lässt, dass*

- $v \neq \epsilon$,
- $|uv| \leq n$, und
- $\forall i \geq 0. uv^i w \in R$.

Beweis:

Sei $R = L(A)$, $A = (Q, \Sigma, \delta, q_0, F)$.

Sei $n = |Q|$. Sei nun $z = a_1 \dots a_m \in R$ mit $m \geq n$.

Die beim Lesen von z durchlaufene Zustandsfolge sei

$$q_0 = p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \dots \xrightarrow{a_m} p_m$$

Beweis:

Sei $R = L(A)$, $A = (Q, \Sigma, \delta, q_0, F)$.

Sei $n = |Q|$. Sei nun $z = a_1 \dots a_m \in R$ mit $m \geq n$.

Die beim Lesen von z durchlaufene Zustandsfolge sei

$$q_0 = p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \dots \xrightarrow{a_m} p_m$$

Dann muss es $0 \leq i < j \leq n$ geben mit $p_i = p_j$.

Beweis:

Sei $R = L(A)$, $A = (Q, \Sigma, \delta, q_0, F)$.

Sei $n = |Q|$. Sei nun $z = a_1 \dots a_m \in R$ mit $m \geq n$.

Die beim Lesen von z durchlaufene Zustandsfolge sei

$$q_0 = p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \dots \xrightarrow{a_m} p_m$$

Dann muss es $0 \leq i < j \leq n$ geben mit $p_i = p_j$.

Wir teilen z wie folgt auf: $\underbrace{a_1 \dots a_i}_u \underbrace{a_{i+1} \dots a_j}_v \underbrace{a_{j+1} \dots a_{|z|}}_w$

Beweis:

Sei $R = L(A)$, $A = (Q, \Sigma, \delta, q_0, F)$.

Sei $n = |Q|$. Sei nun $z = a_1 \dots a_m \in R$ mit $m \geq n$.

Die beim Lesen von z durchlaufene Zustandsfolge sei

$$q_0 = p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \dots \xrightarrow{a_m} p_m$$

Dann muss es $0 \leq i < j \leq n$ geben mit $p_i = p_j$.

Wir teilen z wie folgt auf: $\underbrace{a_1 \dots a_i}_u \underbrace{a_{i+1} \dots a_j}_v \underbrace{a_{j+1} \dots a_{|z|}}_w$

Damit gilt:

- $|uv| \leq n$,
- $v \neq \epsilon$, und
- $\forall l \geq 0. uv^l w \in R$.



Beweis:

Sei $R = L(A)$, $A = (Q, \Sigma, \delta, q_0, F)$.

Sei $n = |Q|$. Sei nun $z = a_1 \dots a_m \in R$ mit $m \geq n$.

Die beim Lesen von z durchlaufene Zustandsfolge sei

$$q_0 = p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \dots \xrightarrow{a_m} p_m$$

Dann muss es $0 \leq i < j \leq n$ geben mit $p_i = p_j$.

Wir teilen z wie folgt auf: $\underbrace{a_1 \dots a_i}_u \underbrace{a_{i+1} \dots a_j}_v \underbrace{a_{j+1} \dots a_{|z|}}_w$

Damit gilt:

- $|uv| \leq n$,
- $v \neq \epsilon$, und
- $\forall l \geq 0. uv^l w \in R$.



[Darf A NFA sein?]

- Logische Struktur: für jede reguläre Sprache R

$$\exists n > 0.$$

- Logische Struktur: für jede reguläre Sprache R

$$\exists n > 0. \forall z \in R \text{ mit } |z| \geq n.$$

- Logische Struktur: für jede reguläre Sprache R

$\exists n > 0. \forall z \in R \text{ mit } |z| \geq n. \exists u, v, w. z = uvw \wedge \dots$

- Logische Struktur: für jede reguläre Sprache R

$$\exists n > 0. \forall z \in R \text{ mit } |z| \geq n. \exists u, v, w. z = uvw \wedge \dots$$

- Spielbasierte Formulierung.

Sei R eine Sprache (regulär oder nicht). Eine Partie des R -Spiels zwischen zwei Spielern läuft so:

- **Spieler I** wählt eine Zahl $n > 0$
- **Spieler II** wählt ein Wort $z \in R$ mit $|z| \geq n$
- **Spieler I** wählt eine Zerlegung uvw von z .

Spieler I gewinnt, wenn uvw die gewünschten Eigenschaften erfüllt, sonst gewinnt **Spieler II**.

- Logische Struktur: für jede reguläre Sprache R

$\exists n > 0. \forall z \in R \text{ mit } |z| \geq n. \exists u, v, w. z = uvw \wedge \dots$

- Spielbasierte Formulierung.

Sei R eine Sprache (regulär oder nicht). Eine Partie des R -Spiels zwischen zwei Spielern läuft so:

- **Spieler I** wählt eine Zahl $n > 0$
- **Spieler II** wählt ein Wort $z \in R$ mit $|z| \geq n$
- **Spieler I** wählt eine Zerlegung uvw von z .

Spieler I gewinnt, wenn uvw die gewünschten Eigenschaften erfüllt, sonst gewinnt **Spieler II**.

- Das Pumping Lemma sagt: für jede reguläre Sprache R , **Spieler I** hat eine *Gewinnstrategie* im R -Spiel, d.h., wenn er richtig spielt, dann gewinnt er jede Partie.

- Logische Struktur: für jede reguläre Sprache R

$\exists n > 0. \forall z \in R \text{ mit } |z| \geq n. \exists u, v, w. z = uvw \wedge \dots$

- Spielbasierte Formulierung.

Sei R eine Sprache (regulär oder nicht). Eine Partie des R -Spiels zwischen zwei Spielern läuft so:

- **Spieler I** wählt eine Zahl $n > 0$
- **Spieler II** wählt ein Wort $z \in R$ mit $|z| \geq n$
- **Spieler I** wählt eine Zerlegung uvw von z .

Spieler I gewinnt, wenn uvw die gewünschten Eigenschaften erfüllt, sonst gewinnt **Spieler II**.

- Das Pumping Lemma sagt: für jede reguläre Sprache R , **Spieler I** hat eine *Gewinnstrategie* im R -Spiel, d.h., wenn er richtig spielt, dann gewinnt er jede Partie.
- Gewinnstrategie für **Spieler I**: Regeln, um erst eine **Pumping-Lemma-Zahl** n , und dann eine **gute** Zerlegung uvw zu wählen (eine Zerlegung, die die Eigenschaften erfüllt).

Falls $L(M) = R$ so ist $|Q_M|$ eine Pumping-Lemma-Zahl für R .

Falls $L(M) = R$ so ist $|Q_M|$ eine Pumping-Lemma-Zahl für R .

Pumping-Lemma-Zahl für $L(ab^*c)$? Und für $\{aaaaaa\}$?

Falls $L(M) = R$ so ist $|Q_M|$ eine Pumping-Lemma-Zahl für R .

Pumping-Lemma-Zahl für $L(ab^*c)$? Und für $\{aaaaaa\}$?

Ist $|Q_M| + 1$ auch eine Pumping-Lemma-Zahl für R ?

Anwendung des Pumping Lemmas:

- Um zu beweisen, dass R nicht regulär ist, zeige, dass **Spieler I** keine Gewinnstrategie hat.
 - Durch Widerspruch
 - Äquivalent: Zeige, dass **Spieler II** eine Gewinnstrategie hat (Regeln, um z in Abhängigkeit von R und n zu wählen).

Satz 3.33

Die Sprache $\{a^i b^i \mid i \in \mathbb{N}\}$ ist nicht regulär.

Satz 3.33

Die Sprache $\{a^i b^i \mid i \in \mathbb{N}\}$ ist nicht regulär.

Beweis:

Angenommen, L sei doch regulär.

Satz 3.33

Die Sprache $\{a^i b^i \mid i \in \mathbb{N}\}$ ist nicht regulär.

Beweis:

Angenommen, L sei doch regulär.

Sei n eine Pumping-Lemma-Zahl für L .

Satz 3.33

Die Sprache $\{a^i b^i \mid i \in \mathbb{N}\}$ ist nicht regulär.

Beweis:

Angenommen, L sei doch regulär.

Sei n eine Pumping-Lemma-Zahl für L .

Wähle $z = a^n b^n \in L$.

Satz 3.33

Die Sprache $\{a^i b^i \mid i \in \mathbb{N}\}$ ist nicht regulär.

Beweis:

Angenommen, L sei doch regulär.

Sei n eine Pumping-Lemma-Zahl für L .

Wähle $z = a^n b^n \in L$.

Jede gute Zerlegung uvw von z erfüllt
 $u, v \in \{a\}^*$ (weil $|uv| \leq n$) und $v \neq \epsilon$.

Satz 3.33

Die Sprache $\{a^i b^i \mid i \in \mathbb{N}\}$ ist nicht regulär.

Beweis:

Angenommen, L sei doch regulär.

Sei n eine Pumping-Lemma-Zahl für L .

Wähle $z = a^n b^n \in L$.

Jede gute Zerlegung uvw von z erfüllt

$u, v \in \{a\}^*$ (weil $|uv| \leq n$) und $v \neq \epsilon$.

Damit müsste gelten $a^{n-|v|} b^n = uv \in L$. ⚡



“Endliche Automaten können nicht unbegrenzt zählen”

Satz 3.33

Die Sprache $\{a^i b^i \mid i \in \mathbb{N}\}$ ist nicht regulär.

Beweis:

Angenommen, L sei doch regulär.

Sei n eine Pumping-Lemma-Zahl für L .

Wähle $z = a^n b^n \in L$.

Jede gute Zerlegung uvw von z erfüllt

$u, v \in \{a\}^*$ (weil $|uv| \leq n$) und $v \neq \epsilon$.

Damit müsste gelten $a^{n-|v|} b^n = uv \in L$. \downarrow



“Endliche Automaten können nicht unbegrenzt zählen”

Satz 3.33

Die Sprache $\{a^i b^i \mid i \in \mathbb{N}\}$ ist nicht regulär.

Beweis:

Angenommen, L sei doch regulär.

Sei n eine Pumping-Lemma-Zahl für L .

Wähle $z = a^n b^n \in L$.

Jede gute Zerlegung uvw von z erfüllt

$u, v \in \{a\}^*$ (weil $|uv| \leq n$) und $v \neq \epsilon$.

Damit müsste gelten $a^{n-|v|} b^n = uv \in L$. \nexists



“Endliche Automaten können nicht unbegrenzt zählen”

Ist die Sprache $\{a^n b^n \mid n \leq 10^6\}$ regulär?

Satz 3.34

$L = \{0^{m^2} \mid m \geq 0\}$ ist nicht regulär.

Satz 3.34

$L = \{0^{m^2} \mid m \geq 0\}$ ist nicht regulär.

Beweis:

Angenommen, L sei doch regulär.

Satz 3.34

$L = \{0^{m^2} \mid m \geq 0\}$ ist nicht regulär.

Beweis:

Angenommen, L sei doch regulär.

Sei n eine Pumping-Lemma-Zahl für L .

Satz 3.34

$L = \{0^{m^2} \mid m \geq 0\}$ ist nicht regulär.

Beweis:

Angenommen, L sei doch regulär.

Sei n eine Pumping-Lemma-Zahl für L .

Wähle $z = 0^{n^2} \in L$. Jede gute Zerlegung uvw von z erfüllt

$$1 \leq |v| \leq |uv| \leq n$$

und $uv^l w \in L$ für alle $l \in \mathbb{N}$. D.h. insb. $|uv^2 w|$ ist Quadratzahl.

Satz 3.34

$L = \{0^{m^2} \mid m \geq 0\}$ ist nicht regulär.

Beweis:

Angenommen, L sei doch regulär.

Sei n eine Pumping-Lemma-Zahl für L .

Wähle $z = 0^{n^2} \in L$. Jede gute Zerlegung uvw von z erfüllt

$$1 \leq |v| \leq |uv| \leq n$$

und $uv^l w \in L$ für alle $l \in \mathbb{N}$. D.h. insb. $|uv^2 w|$ ist Quadratzahl.

Dies führt zu einem Widerspruch:

$$n^2 = |z| = |uvw| < |uv^2 w| \leq n^2 + n < n^2 + 2n + 1 = (n + 1)^2$$

Satz 3.34

$L = \{0^{m^2} \mid m \geq 0\}$ ist nicht regulär.

Beweis:

Angenommen, L sei doch regulär.

Sei n eine Pumping-Lemma-Zahl für L .

Wähle $z = 0^{n^2} \in L$. Jede gute Zerlegung uvw von z erfüllt

$$1 \leq |v| \leq |uv| \leq n$$

und $uv^l w \in L$ für alle $l \in \mathbb{N}$. D.h. insb. $|uv^2 w|$ ist Quadratzahl.

Dies führt zu einem Widerspruch:

$$n^2 = |z| = |uvw| < |uv^2 w| \leq n^2 + n < n^2 + 2n + 1 = (n + 1)^2$$

Denn zwischen n^2 und $(n + 1)^2$ liegt keine Quadratzahl. □

Satz 3.35

Die Sprache der wohlgeklammerten Ausdrücke über $\{(,)\}$ ist nicht regulär.

Beweis:

Aufgabe.



Satz 3.35

Die Sprache der wohlgeklammerten Ausdrücke über $\{(,)\}$ ist nicht regulär.

Beweis:

Aufgabe. □

Satz 3.36

Die Sprache *Arith* der arithmetischen Ausdrücken ist nicht regulär.

Satz 3.35

Die Sprache der wohlgeklammerten Ausdrücke über $\{(,)\}$ ist nicht regulär.

Beweis:

Aufgabe. □

Satz 3.36

Die Sprache *Arith* der arithmetischen Ausdrücken ist nicht regulär.

Beweis:

Angenommen, *Arith* sei doch regulär.

Dann gibt es einen DFA A mit $L(A) = \textit{Arith}$.

Ersetze alle Transitionen von A , die nicht mit $($ oder $)$ beschriftet sind durch ϵ -Transitionen.

Der resultierende ϵ -NFA erkennt die Sprache der wohlgeklammerten Ausdrücke. Widerspruch zu Satz 3.35. □

Bemerkung

Es gibt nicht-reguläre Sprachen, für die das Pumping-Lemma gilt!

⇒ Pumping-Lemma hinreichend aber nicht notwendig um Nicht-Regularität zu zeigen.

regulär \subset Pumping-Lemma gilt \subset alle Sprachen

3.8 Entscheidungsverfahren

Wir untersuchen **Entscheidungsprobleme für reguläre Sprachen**,
d.h., Probleme der Gestalt

Eingabe: Ein oder mehrere Objekte, die reguläre Sprachen
beschreiben (DFA, NFA, RE Typ 3 Gram., ...)

Frage: Haben die Sprachen dieser Objekte eine Eigenschaft X?

3.8 Entscheidungsverfahren

Wir untersuchen **Entscheidungsprobleme für reguläre Sprachen**, d.h., Probleme der Gestalt

Eingabe: Ein oder mehrere Objekte, die reguläre Sprachen beschreiben (DFA, NFA, RE Typ 3 Gram., ...)

Frage: Haben die Sprachen dieser Objekte eine Eigenschaft X?

Ein (Entscheidungs)Problem ist **entscheidbar**, wenn es einen Algorithmus gibt, der bei jeder Eingabe in endlicher Zeit die richtige Antwort auf die Frage feststellt.

3.8 Entscheidungsverfahren

Wir untersuchen **Entscheidungsprobleme für reguläre Sprachen**, d.h., Probleme der Gestalt

Eingabe: Ein oder mehrere Objekte, die reguläre Sprachen beschreiben (DFA, NFA, RE Typ 3 Gram., ...)

Frage: Haben die Sprachen dieser Objekte eine Eigenschaft X?

Ein (Entscheidungs)Problem ist **entscheidbar**, wenn es einen Algorithmus gibt, der bei jeder Eingabe in endlicher Zeit die richtige Antwort auf die Frage feststellt.

In der zweiten Hälfte der Vorlesung wird gezeigt, dass nicht alle Entscheidungsprobleme für Grammatiken entscheidbar sind!

3.8 Entscheidungsverfahren

Wir untersuchen **Entscheidungsprobleme für reguläre Sprachen**, d.h., Probleme der Gestalt

Eingabe: Ein oder mehrere Objekte, die reguläre Sprachen beschreiben (DFA, NFA, RE Typ 3 Gram., ...)

Frage: Haben die Sprachen dieser Objekte eine Eigenschaft X?

Ein (Entscheidungs)Problem ist **entscheidbar**, wenn es einen Algorithmus gibt, der bei jeder Eingabe in endlicher Zeit die richtige Antwort auf die Frage feststellt.

In der zweiten Hälfte der Vorlesung wird gezeigt, dass nicht alle Entscheidungsprobleme für Grammatiken entscheidbar sind!

Welche Entscheidungsprobleme sind für rechtslineare Grammatiken (oder DFA; NFA; RE ...) entscheidbar?

Wie hängt die Laufzeit des Algorithmus mit der Beschreibung zusammen?

Definition 3.37

Sei D ein DFA, NFA, RE, rechtslineare Grammatik

Definition 3.37

Sei D ein DFA, NFA, RE, rechtslineare Grammatik

Wortproblem: Gegeben w und D , gilt $w \in L(D)$?

Leerheitsproblem: Gegeben D , gilt $L(D) = \emptyset$?

Endlichkeitsproblem: Gegeben D , ist $L(D)$ endlich?

Äquivalenzproblem: Gegeben D_1, D_2 , gilt $L(D_1) = L(D_2)$?

Lemma 3.38

Das Wortproblem ist für ein Wort w und DFA M in Zeit $O(|w| + |M|)$ entscheidbar.

Lemma 3.38

Das Wortproblem ist für ein Wort w und DFA M in Zeit $O(|w| + |M|)$ entscheidbar.

Lemma 3.39

Das Wortproblem ist für ein Wort w und NFA N in Zeit $O(|Q|^2|w| + |N|)$ entscheidbar.

Lemma 3.38

Das Wortproblem ist für ein Wort w und DFA M in Zeit $O(|w| + |M|)$ entscheidbar.

Lemma 3.39

Das Wortproblem ist für ein Wort w und NFA N in Zeit $O(|Q|^2|w| + |N|)$ entscheidbar.

Beweis:

Sei $w = a_1 \dots a_n$.

$S := \{q_0\}$

for $i := 1$ **to** n **do** $S := \bigcup_{q \in S} \delta(q, a_i)$

return $(S \cap F \neq \emptyset)$



Lemma 3.40

Das Leerheitsproblem ist für NFAs und DFAs entscheidbar (in Zeit $O(|Q|^2|\Sigma|)$ bzw $O(|Q||\Sigma|)$).

Lemma 3.40

Das Leerheitsproblem ist für NFAs und DFAs entscheidbar (in Zeit $O(|Q|^2|\Sigma|)$ bzw $O(|Q||\Sigma|)$).

Beweis:

$L(M) = \emptyset$ gdw kein Endzustand von q_0 erreichbar ist.

Dies ist eine einfache Suche in einem Graphen, die jede Kante maximal ein Mal benutzen muss.

Ein NFA hat $\leq |Q|^2|\Sigma|$ Kanten, ein DFA hat $\leq |Q||\Sigma|$ Kanten. \square

Lemma 3.40

Das Leerheitsproblem ist für NFAs und DFAs entscheidbar (in Zeit $O(|Q|^2|\Sigma|)$ bzw $O(|Q||\Sigma|)$).

Beweis:

$L(M) = \emptyset$ gdw kein Endzustand von q_0 erreichbar ist.

Dies ist eine einfache Suche in einem Graphen, die jede Kante maximal ein Mal benutzen muss.

Ein NFA hat $\leq |Q|^2|\Sigma|$ Kanten, ein DFA hat $\leq |Q||\Sigma|$ Kanten. \square

Ist Σ fix, z.B. ASCII, so wird daraus $O(|Q|^2)$ bzw $O(|Q|)$.

Lemma 3.41

Das Endlichkeitsproblem ist für DFAs oder NFAs entscheidbar.

Lemma 3.41

Das Endlichkeitsproblem ist für DFAs oder NFAs entscheidbar.

Beweis:

$|L(M)| = \infty$ gdw von q_0 aus eine nicht-leere Schleife erreichbar ist, von der aus F erreichbar ist.

Lemma 3.41

Das Endlichkeitsproblem ist für DFAs oder NFAs entscheidbar.

Beweis:

$|L(M)| = \infty$ gdw von q_0 aus eine nicht-leere Schleife erreichbar ist, von der aus F erreichbar ist.

```
Reach(K) =   R :=  $\emptyset$ ; W := K
              while W  $\neq \emptyset$  do
                pick and remove some  $p \in W$ 
                if  $p \notin R$  then
                  R :=  $R \cup \{p\}$ ; W :=  $W \cup \bigcup_{a \in \Sigma} \delta(p, a)$ 
              return R
```

```
Finite(Q,  $\Sigma$ ,  $\delta$ ,  $q_0$ , F) = R := Reach( $\{q_0\}$ )
                                C :=  $\{p \in R \mid p \in \text{Reach}(\bigcup_{a \in \Sigma} \delta(p, a))\}$ 
                                return (Reach(C)  $\cap F = \emptyset$ )
```



Lemma 3.42

Das Äquivalenzproblem ist für DFAs entscheidbar.

Beweis:

Folgt direkt aus

$$L_1 \subseteq L_2 \Leftrightarrow L_1 \cap \overline{L_2} = \emptyset$$

$$L_1 = L_2 \Leftrightarrow L_1 \subseteq L_2 \wedge L_2 \subseteq L_1$$

da für DFAs Komplement und Durchschnitt wieder endliche Automaten liefern und das Leerheitsproblem für endliche Automaten entscheidbar ist. □

Satz 3.43

Das Äquivalenzproblem für DFAs ist in Zeit $O(|Q_1||Q_2||\Sigma|)$ entscheidbar.

Satz 3.43

Das Äquivalenzproblem für DFAs ist in Zeit $O(|Q_1||Q_2||\Sigma|)$ entscheidbar.

Beweis:

Gegeben: DFAs M_1 mit m und M_2 mit n Zuständen.

Mit Hilfe der Produkt-Konstruktion für \cap folgt:

	Anzahl der Zustände
$\overline{L(M_1)} \cap \overline{L(M_2)}$	mn
$\overline{L(M_1)} \cap L(M_2)$	mn



Korollar 3.44

Das Äquivalenzproblem für NFAs ist in Zeit $O(2^{|Q_1|+|Q_2|})$ entscheidbar (bei fixem Σ).

Korollar 3.44

Das Äquivalenzproblem für NFAs ist in Zeit $O(2^{|Q_1|+|Q_2|})$ entscheidbar (bei fixem Σ).

Beweis:

2 NFAs mit m und n Zuständen \rightsquigarrow

2 DFAs mit 2^m und 2^n Zuständen \rightsquigarrow

Äquivalenztest in Zeit $O(2^m 2^n)$



Korollar 3.44

Das Äquivalenzproblem für NFAs ist in Zeit $O(2^{|Q_1|+|Q_2|})$ entscheidbar (bei fixem Σ).

Beweis:

2 NFAs mit m und n Zuständen \rightsquigarrow

2 DFAs mit 2^m und 2^n Zuständen \rightsquigarrow

Äquivalenztest in Zeit $O(2^m 2^n)$



Korollar 3.45

Das Äquivalenzproblem für reguläre Ausdrücke ist entscheidbar.

Fazit:

Die Kodierung der Eingabe (DFA, NFA, RE, ...) kann entscheidend für die Komplexität eines Problems sein.

3.9 Automaten und Gleichungssysteme

Nicht mehr *Beweisen* von Äquivalenzen

Gilt $XX^* \equiv X^*X$ für alle X ?

3.9 Automaten und Gleichungssysteme

Nicht mehr *Beweisen* von Äquivalenzen

Gilt $XX^* \equiv X^*X$ für alle X ?

sondern *Lösen*:

Für welches X gilt $X \equiv aX \mid b$?

3.9 Automaten und Gleichungssysteme

Nicht mehr *Beweisen* von Äquivalenzen

Gilt $XX^* \equiv X^*X$ für alle X ?

sondern *Lösen*:

Für welches X gilt $X \equiv aX \mid b$?

Anwendung:

Automat \rightsquigarrow Gleichungssystem \rightsquigarrow RE

3.9 Automaten und Gleichungssysteme

Nicht mehr *Beweisen* von Äquivalenzen

Gilt $XX^* \equiv X^*X$ für alle X ?

sondern *Lösen*:

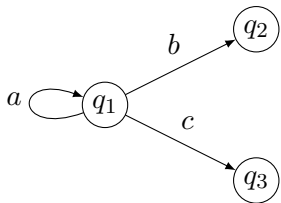
Für welches X gilt $X \equiv aX \mid b$?

Anwendung:

Automat \rightsquigarrow Gleichungssystem \rightsquigarrow RE

Beispiel 3.46

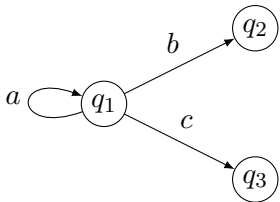
Ein Automatenfragment:



$$L_i := \{w \mid \hat{\delta}(q_i, w) \in F\}$$
$$\implies$$
$$L_1 = \{a\}L_1 \cup \{b\}L_2 \cup \{c\}L_3$$

Beispiel 3.46

Ein Automatenfragment:



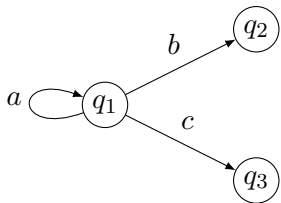
$$\begin{aligned} L_i &:= \{w \mid \hat{\delta}(q_i, w) \in F\} \\ &\implies \\ L_1 &= \{a\}L_1 \cup \{b\}L_2 \cup \{c\}L_3 \end{aligned}$$

Da die L_i regulär sein müssen, arbeiten wir direkt mit REs:

$$X_1 \equiv aX_1 \mid bX_2 \mid cX_3$$

Beispiel 3.46

Ein Automatenfragment:



$$\begin{aligned} L_i &:= \{w \mid \hat{\delta}(q_i, w) \in F\} \\ &\implies \\ L_1 &= \{a\}L_1 \cup \{b\}L_2 \cup \{c\}L_3 \end{aligned}$$

Da die L_i regulär sein müssen, arbeiten wir direkt mit REs:

$$X_1 \equiv aX_1 \mid bX_2 \mid cX_3$$

Lösung X_i ist RE für die von q_i aus akzeptierte Sprache.

Satz 3.47 (Ardens Lemma)

Sind A , B und X Sprachen mit $\epsilon \notin A$, so gilt

$$X = AX \cup B \implies X = A^*B$$

Satz 3.47 (Ardens Lemma)

Sind A , B und X Sprachen mit $\epsilon \notin A$, so gilt

$$X = AX \cup B \implies X = A^*B$$

Korollar 3.48

Sind α , β und X reguläre Ausdrücke mit $\epsilon \notin L(\alpha)$, so gilt

$$X \equiv \alpha X \mid \beta \implies X \equiv \alpha^* \beta$$

Satz 3.47 (Ardens Lemma)

Sind A, B und X Sprachen mit $\epsilon \notin A$, so gilt

$$X = AX \cup B \implies X = A^*B$$

Korollar 3.48

Sind α, β und X reguläre Ausdrücke mit $\epsilon \notin L(\alpha)$, so gilt

$$X \equiv \alpha X \mid \beta \implies X \equiv \alpha^* \beta$$

Bemerkungen

- $X = \{\epsilon\}X \cup B$ hat keine eindeutige Lösung:
jede Sprache $X \supseteq B$ ist Lösung.
- $X \equiv aXb \mid \epsilon$ hat keine reguläre Lösung.

Satz 3.47 (Ardens Lemma)

Sind A , B und X Sprachen mit $\epsilon \notin A$, so gilt

$$X = AX \cup B \implies X = A^*B$$

Korollar 3.48

Sind α , β und X reguläre Ausdrücke mit $\epsilon \notin L(\alpha)$, so gilt

$$X \equiv \alpha X \mid \beta \implies X \equiv \alpha^* \beta$$

Bemerkungen

- $X = \{\epsilon\}X \cup B$ hat keine eindeutige Lösung:
jede Sprache $X \supseteq B$ ist Lösung.
- $X \equiv aXb \mid \epsilon$ hat keine reguläre Lösung.

Satz 3.47 (Ardens Lemma)

Sind A , B und X Sprachen mit $\epsilon \notin A$, so gilt

$$X = AX \cup B \implies X = A^*B$$

Korollar 3.48

Sind α , β und X reguläre Ausdrücke mit $\epsilon \notin L(\alpha)$, so gilt

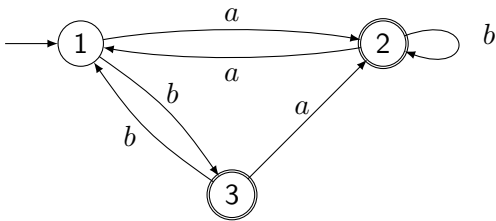
$$X \equiv \alpha X \mid \beta \implies X \equiv \alpha^* \beta$$

Bemerkungen

- $X = \{\epsilon\}X \cup B$ hat keine eindeutige Lösung:
jede Sprache $X \supseteq B$ ist Lösung.
- $X \equiv aXb \mid \epsilon$ hat keine reguläre Lösung.

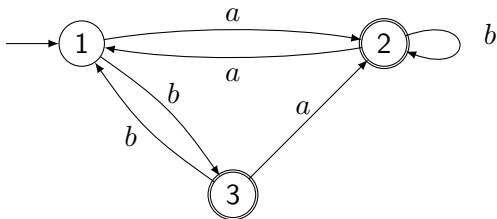
Umwandlung eines DFAs in einen äquivalenten regulären Ausdruck:

Beispiel 3.49



Umwandlung eines DFAs in einen äquivalenten regulären Ausdruck:

Beispiel 3.49



Äquivalentes Gleichungssystem:

X_i ist ein RE für die von q_i aus akzeptierte Sprache.

$$X_1 \equiv aX_2 \mid bX_3$$

$$X_2 \equiv aX_1 \mid bX_2 \mid \epsilon$$

$$X_3 \equiv bX_1 \mid aX_2 \mid \epsilon$$

Lösen des Gleichungssystems:

$$X_1 \equiv aX_2 \mid bX_3$$

$$X_2 \equiv aX_1 \mid bX_2 \mid \epsilon$$

$$X_3 \equiv bX_1 \mid aX_2 \mid \epsilon$$

Lösen des Gleichungssystems:

$$X_1 \equiv aX_2 \mid bX_3$$

$$X_2 \equiv aX_1 \mid bX_2 \mid \epsilon$$

$$X_3 \equiv bX_1 \mid aX_2 \mid \epsilon$$

Löse $X_2 \equiv bX_2 \mid (aX_1 \mid \epsilon)$ nach X_2 auf:

$$X_2 \equiv b^*(aX_1 \mid \epsilon)$$

Lösen des Gleichungssystems:

$$X_1 \equiv aX_2 \mid bX_3$$

$$X_2 \equiv aX_1 \mid bX_2 \mid \epsilon$$

$$X_3 \equiv bX_1 \mid aX_2 \mid \epsilon$$

Löse $X_2 \equiv bX_2 \mid (aX_1 \mid \epsilon)$ nach X_2 auf:

$$X_2 \equiv b^*(aX_1 \mid \epsilon)$$

Zurück einsetzen:

$$X_1 \equiv a(b^*(aX_1 \mid \epsilon)) \mid bX_3$$

$$X_3 \equiv bX_1 \mid a(b^*(aX_1 \mid \epsilon)) \mid \epsilon$$

Lösen des Gleichungssystems:

$$X_1 \equiv aX_2 \mid bX_3$$

$$X_2 \equiv aX_1 \mid bX_2 \mid \epsilon$$

$$X_3 \equiv bX_1 \mid aX_2 \mid \epsilon$$

Löse $X_2 \equiv bX_2 \mid (aX_1 \mid \epsilon)$ nach X_2 auf:

$$X_2 \equiv b^*(aX_1 \mid \epsilon)$$

Zurück einsetzen:

$$X_1 \equiv a(b^*(aX_1 \mid \epsilon)) \mid bX_3$$

$$X_3 \equiv bX_1 \mid a(b^*(aX_1 \mid \epsilon)) \mid \epsilon$$

Ausmultiplizieren und X_i ausklammern:

$$X_1 \equiv ab^*aX_1 \mid ab^* \mid bX_3$$

$$X_3 \equiv (b \mid ab^*a)X_1 \mid ab^* \mid \epsilon$$

$$\begin{aligned} X_1 &\equiv ab^*aX_1 \mid bX_3 \mid ab^* \\ X_3 &\equiv (b \mid ab^*a)X_1 \mid ab^* \mid \epsilon \end{aligned}$$

$$\begin{aligned}
 X_1 &\equiv ab^*aX_1 \mid bX_3 \mid ab^* \\
 X_3 &\equiv (b \mid ab^*a)X_1 \mid ab^* \mid \epsilon
 \end{aligned}$$

X_3 ist gelöst, in 1. Gleichung einsetzen:

$$X_1 \equiv ab^*aX_1 \mid b((b \mid ab^*a)X_1 \mid ab^* \mid \epsilon) \mid ab^*$$

$$\begin{aligned}
 X_1 &\equiv ab^*aX_1 \mid bX_3 \mid ab^* \\
 X_3 &\equiv (b \mid ab^*a)X_1 \mid ab^* \mid \epsilon
 \end{aligned}$$

X_3 ist gelöst, in 1. Gleichung einsetzen:

$$X_1 \equiv ab^*aX_1 \mid b((b \mid ab^*a)X_1 \mid ab^* \mid \epsilon) \mid ab^*$$

Ausmultiplizieren und X_1 ausklammern:

$$X_1 \equiv (ab^*a \mid bb \mid bab^*a)X_1 \mid bab^* \mid b \mid ab^*$$

$$\begin{aligned}
 X_1 &\equiv ab^*aX_1 \mid bX_3 \mid ab^* \\
 X_3 &\equiv (b \mid ab^*a)X_1 \mid ab^* \mid \epsilon
 \end{aligned}$$

X_3 ist gelöst, in 1. Gleichung einsetzen:

$$X_1 \equiv ab^*aX_1 \mid b((b \mid ab^*a)X_1 \mid ab^* \mid \epsilon) \mid ab^*$$

Ausmultiplizieren und X_1 ausklammern:

$$X_1 \equiv (ab^*a \mid bb \mid bab^*a)X_1 \mid bab^* \mid b \mid ab^*$$

Nach X_1 auflösen:

$$X_1 \equiv (ab^*a \mid bb \mid bab^*a)^*(bab^* \mid b \mid ab^*)$$

Umwandlung eines FAs in einen äquivalenten regulären Ausdruck:

Umwandlung eines FAs in einen äquivalenten regulären Ausdruck:

- Wandle FA mit n Zuständen in ein System von n Gleichungen mit n Variablen um:

$$X_i \equiv a_{i1}X_1 \mid \cdots \mid a_{in}X_n \mid b_i$$

Umwandlung eines FAs in einen äquivalenten regulären Ausdruck:

- Wandle FA mit n Zuständen in ein System von n Gleichungen mit n Variablen um:

$$X_i \equiv a_{i1}X_1 \mid \cdots \mid a_{in}X_n \mid b_i$$

$$a_{ij} := c_1 \mid \cdots \mid c_k \quad \text{falls } \{c_1, \dots, c_k\} = \{c \in \Sigma \mid q_i \xrightarrow{c} q_j\}$$

wobei $a_{ij} := \emptyset$ falls $q_i \xrightarrow{c} q_j$ für kein $c \in \Sigma$

Umwandlung eines FAs in einen äquivalenten regulären Ausdruck:

- Wandle FA mit n Zuständen in ein System von n Gleichungen mit n Variablen um:

$$X_i \equiv a_{i1}X_1 \mid \cdots \mid a_{in}X_n \mid b_i$$

$$a_{ij} := c_1 \mid \cdots \mid c_k \quad \text{falls } \{c_1, \dots, c_k\} = \{c \in \Sigma \mid q_i \xrightarrow{c} q_j\}$$

wobei $a_{ij} := \emptyset$ falls $q_i \xrightarrow{c} q_j$ für kein $c \in \Sigma$

$$b_i := \begin{cases} \epsilon & \text{falls } q_i \in F \\ \emptyset & \text{sonst} \end{cases}$$

Umwandlung eines FAs in einen äquivalenten regulären Ausdruck:

- Wandle FA mit n Zuständen in ein System von n Gleichungen mit n Variablen um:

$$X_i \equiv a_{i1}X_1 \mid \cdots \mid a_{in}X_n \mid b_i$$

$$a_{ij} := c_1 \mid \cdots \mid c_k \quad \text{falls } \{c_1, \dots, c_k\} = \{c \in \Sigma \mid q_i \xrightarrow{c} q_j\}$$

wobei $a_{ij} := \emptyset$ falls $q_i \xrightarrow{c} q_j$ für kein $c \in \Sigma$

$$b_i := \begin{cases} \epsilon & \text{falls } q_i \in F \\ \emptyset & \text{sonst} \end{cases}$$

- Löse das System durch schrittweise Elimination von Variablen mit Hilfe von Ardens Lemma für REs (Korollar 3.48).

Umwandlung eines FAs in einen äquivalenten regulären Ausdruck:

- Wandle FA mit n Zuständen in ein System von n Gleichungen mit n Variablen um:

$$X_i \equiv a_{i1}X_1 \mid \cdots \mid a_{in}X_n \mid b_i$$

$$a_{ij} := c_1 \mid \cdots \mid c_k \quad \text{falls } \{c_1, \dots, c_k\} = \{c \in \Sigma \mid q_i \xrightarrow{c} q_j\}$$

wobei $a_{ij} := \emptyset$ falls $q_i \xrightarrow{c} q_j$ für kein $c \in \Sigma$

$$b_i := \begin{cases} \epsilon & \text{falls } q_i \in F \\ \emptyset & \text{sonst} \end{cases}$$

- Löse das System durch schrittweise Elimination von Variablen mit Hilfe von Ardens Lemma für REs (Korollar 3.48).
- Ist k der Startzustand, so beschreibt X_k die vom Automaten akzeptierte Sprache.

Beweis von Ardens Lemma:

Wir nehmen an $X = AX \cup B$.

$$X = A(AX \cup B) \cup B = A^2X \cup AB \cup B$$

Beweis von Ardens Lemma:

Wir nehmen an $X = AX \cup B$.

$$\begin{aligned} X &= A(AX \cup B) \cup B = A^2X \cup AB \cup B \\ &= A^2(AX \cup B) \cup AB \cup B = A^3X \cup A^2B \cup AB \cup B = \dots \end{aligned}$$

Beweis von Ardens Lemma:

Wir nehmen an $X = AX \cup B$.

$$\begin{aligned} X &= A(AX \cup B) \cup B = A^2X \cup AB \cup B \\ &= A^2(AX \cup B) \cup AB \cup B = A^3X \cup A^2B \cup AB \cup B = \dots \end{aligned}$$

Mit Induktion zeigt man für alle $n \in \mathbb{N}$:

$$X = A^{n+1}X \cup \bigcup_{i \leq n} A^i B \quad (1)$$

Beweis von Ardens Lemma:

Wir nehmen an $X = AX \cup B$.

$$\begin{aligned} X &= A(AX \cup B) \cup B = A^2X \cup AB \cup B \\ &= A^2(AX \cup B) \cup AB \cup B = A^3X \cup A^2B \cup AB \cup B = \dots \end{aligned}$$

Mit Induktion zeigt man für alle $n \in \mathbb{N}$:

$$X = A^{n+1}X \cup \bigcup_{i \leq n} A^i B \quad (1)$$

0: Behauptung wird zu $X = AX \cup B$, der Annahme.

Beweis von Ardens Lemma:

Wir nehmen an $X = AX \cup B$.

$$\begin{aligned} X &= A(AX \cup B) \cup B = A^2X \cup AB \cup B \\ &= A^2(AX \cup B) \cup AB \cup B = A^3X \cup A^2B \cup AB \cup B = \dots \end{aligned}$$

Mit Induktion zeigt man für alle $n \in \mathbb{N}$:

$$X = A^{n+1}X \cup \bigcup_{i \leq n} A^i B \quad (1)$$

0: Behauptung wird zu $X = AX \cup B$, der Annahme.

$$\begin{aligned} n+1: \quad X &= A^{n+1}X \cup \bigcup_{i \leq n} A^i B \\ &= A^{n+1}(AX \cup B) \cup \bigcup_{i \leq n} A^i B \\ &= A^{n+2}X \cup A^{n+1}B \cup \bigcup_{i \leq n} A^i B \\ &= A^{n+2}X \cup \bigcup_{i \leq n+1} A^i B \end{aligned}$$

Wir zeigen nun $X = A*B$.

Wir zeigen nun $X = A^*B$.

$$\begin{aligned} A^*B \subseteq X: \quad w \in A^*B &= \bigcup_{i \in \mathbb{N}} A^i B \\ &\implies \exists n. w \in A^n B \\ &\implies w \in X \end{aligned}$$

Wir zeigen nun $X = A^*B$.

$$\begin{aligned} A^*B \subseteq X: \quad w \in A^*B &= \bigcup_{i \in \mathbb{N}} A^i B \\ &\implies \exists n. w \in A^n B \\ &\implies w \in X \end{aligned}$$

$$X \subseteq A^*B:$$

Wir zeigen nun $X = A^*B$.

$$\begin{aligned} A^*B \subseteq X: \quad w \in A^*B &= \bigcup_{i \in \mathbb{N}} A^i B \\ &\implies \exists n. w \in A^n B \\ &\implies w \in X \end{aligned}$$

$X \subseteq A^*B$: Sei $w \in X$ und $n := |w|$.

$$\begin{aligned} \epsilon &\notin A \\ &\implies \forall u \in A^{n+1}. |u| \geq n+1 \\ &\implies w \notin A^{n+1}X \\ &\implies w \in \bigcup_{i \leq n} A^i B \quad (\text{wegen (1)}) \\ &\implies w \in \bigcup_{i \in \mathbb{N}} A^i B = A^*B \end{aligned}$$

□

3.10 Minimierung endlicher Automaten

Wir zeigen, dass jede reguläre Sprache einen **einzigsten minimalen Recognizer** hat und geben Algorithmen an, die diesen Recognizer konstruieren.

- 1 Beispiele
- 2 Algorithmen
- 3 Minimalitätsbeweis

Algorithmus zur Minimierung eines DFA

- 1 Entferne alle von q_0 aus nicht erreichbaren Zustände.
- 2 Berechne die **äquivalenten** Zustände des Automaten.
- 3 Kollabiere den Automaten durch Zusammenfassung aller äquivalenten Zustände.

Algorithmus zur Minimierung eines DFA

- 1 Entferne alle von q_0 aus nicht erreichbaren Zustände.
- 2 Berechne die **äquivalenten** Zustände des Automaten.
- 3 Kollabiere den Automaten durch Zusammenfassung aller äquivalenten Zustände.

Zustände p und q sind **unterscheidbar** wenn es $w \in \Sigma^*$ gibt mit $\hat{\delta}(p, w) \in F$ und $\hat{\delta}(q, w) \notin F$ oder umgekehrt.

Zustände sind **äquivalent** wenn sie nicht unterscheidbar sind, d.h. wenn für alle $w \in \Sigma^*$ gilt:

$$\hat{\delta}(p, w) \in F \quad \Leftrightarrow \quad \hat{\delta}(q, w) \in F$$

Algorithmus zur Minimierung eines DFA

- 1 Entferne alle von q_0 aus nicht erreichbaren Zustände.
- 2 Berechne die **äquivalenten** Zustände des Automaten.
- 3 Kollabiere den Automaten durch Zusammenfassung aller äquivalenten Zustände.

Zustände p und q sind **unterscheidbar** wenn es $w \in \Sigma^*$ gibt mit $\hat{\delta}(p, w) \in F$ und $\hat{\delta}(q, w) \notin F$ oder umgekehrt.

Zustände sind **äquivalent** wenn sie nicht unterscheidbar sind, d.h. wenn für alle $w \in \Sigma^*$ gilt:

$$\hat{\delta}(p, w) \in F \quad \Leftrightarrow \quad \hat{\delta}(q, w) \in F$$

- Gilt $p \in F$ und $q \notin F$, dann sind p und q unterscheidbar.
- Sind $\delta(p, a)$ und $\delta(q, a)$ unterscheidbar, dann auch p und q .
 \Rightarrow Unterscheidbarkeit pflanzt sich rückwärts fort.

Berechnung äquivalenter Zustände eines DFA

Berechnung äquivalenter Zustände eines DFA

Eingabe: DFA $A = (Q, \Sigma, \delta, q_0, F)$

Ausgabe: Äquivalenzrelation auf Q .

Berechnung äquivalenter Zustände eines DFA

Eingabe: DFA $A = (Q, \Sigma, \delta, q_0, F)$

Ausgabe: Äquivalenzrelation auf Q .

Datenstruktur: Eine Menge U ungeordneter Paare $\{p, q\} \subseteq Q$.

Berechnung äquivalenter Zustände eines DFA

Eingabe: DFA $A = (Q, \Sigma, \delta, q_0, F)$

Ausgabe: Äquivalenzrelation auf Q .

Datenstruktur: Eine Menge U ungeordneter Paare $\{p, q\} \subseteq Q$.

Algorithmus U:

- 1 $U := \{\{p, q\} \mid p \in F \wedge q \notin F\}$
- 2 **while** $\exists \{p, q\} \notin U. \exists a \in \Sigma. \{\delta(p, a), \delta(q, a)\} \in U$
do $U := U \cup \{\{p, q\}\}$

Berechnung äquivalenter Zustände eines DFA

Eingabe: DFA $A = (Q, \Sigma, \delta, q_0, F)$

Ausgabe: Äquivalenzrelation auf Q .

Datenstruktur: Eine Menge U ungeordneter Paare $\{p, q\} \subseteq Q$.

Algorithmus U:

- 1 $U := \{\{p, q\} \mid p \in F \wedge q \notin F\}$
- 2 **while** $\exists \{p, q\} \notin U. \exists a \in \Sigma. \{\delta(p, a), \delta(q, a)\} \in U$
 do $U := U \cup \{\{p, q\}\}$

Invariante: $\{p, q\} \in U \implies p$ und q unterscheidbar

Berechnung äquivalenter Zustände eines DFA

Eingabe: DFA $A = (Q, \Sigma, \delta, q_0, F)$

Ausgabe: Äquivalenzrelation auf Q .

Datenstruktur: Eine Menge U ungeordneter Paare $\{p, q\} \subseteq Q$.

Algorithmus U:

- 1 $U := \{\{p, q\} \mid p \in F \wedge q \notin F\}$
- 2 **while** $\exists \{p, q\} \notin U. \exists a \in \Sigma. \{\delta(p, a), \delta(q, a)\} \in U$
do $U := U \cup \{\{p, q\}\}$

Invariante: $\{p, q\} \in U \implies p$ und q unterscheidbar

Lemma 3.50

Am Ende gilt: U ist Menge aller unterscheidbaren Zustandspaare.

Berechnung äquivalenter Zustände eines DFA

Eingabe: DFA $A = (Q, \Sigma, \delta, q_0, F)$

Ausgabe: Äquivalenzrelation auf Q .

Datenstruktur: Eine Menge U ungeordneter Paare $\{p, q\} \subseteq Q$.

Algorithmus U:

- 1 $U := \{\{p, q\} \mid p \in F \wedge q \notin F\}$
- 2 **while** $\exists \{p, q\} \notin U. \exists a \in \Sigma. \{\delta(p, a), \delta(q, a)\} \in U$
do $U := U \cup \{\{p, q\}\}$

Invariante: $\{p, q\} \in U \implies p$ und q unterscheidbar

Lemma 3.50

Am Ende gilt: U ist Menge aller unterscheidbaren Zustandspaare.

Beweis:

$\{p, q\} \in U \implies p$ und q unterscheidbar:

Invariante p und q unterscheidbar $\implies \{p, q\} \in U$:

Induktion über die Länge eines unterscheidenden Worts. □

Implementierung von U :

Tabelle von anfangs unmarkierten Paaren $\{p, q\}$, $p \neq q$.

Implementierung von U :

Tabelle von anfangs unmarkierten Paaren $\{p, q\}$, $p \neq q$.

	0			
		1		
			2	
				3

Implementierung von U :

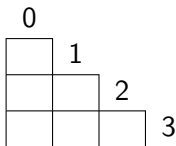
Tabelle von anfangs unmarkierten Paaren $\{p, q\}$, $p \neq q$.

	0			
		1		
			2	
				3

for all $p \in F$, $q \in Q \setminus F$ **do** markiere $\{p, q\}$
while \exists unmarkiertes $\{p, q\}$

Implementierung von U :

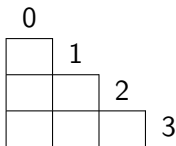
Tabelle von anfangs unmarkierten Paaren $\{p, q\}$, $p \neq q$.



for all $p \in F$, $q \in Q \setminus F$ **do** markiere $\{p, q\}$
while \exists unmarkiertes $\{p, q\} \exists a \in \Sigma$. $\{\delta(p, a), \delta(q, a)\}$ ist markiert
do markiere $\{p, q\}$

Implementierung von U :

Tabelle von anfangs unmarkierten Paaren $\{p, q\}$, $p \neq q$.



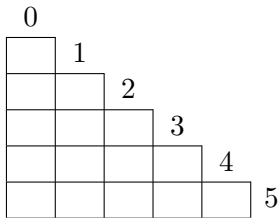
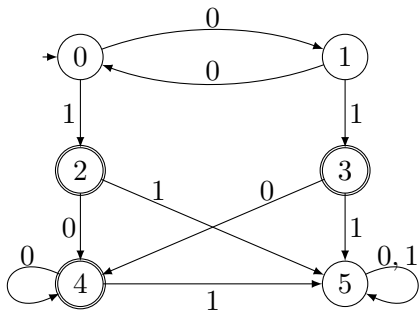
for all $p \in F$, $q \in Q \setminus F$ **do** markiere $\{p, q\}$
while \exists unmarkiertes $\{p, q\} \exists a \in \Sigma$. $\{\delta(p, a), \delta(q, a)\}$ ist markiert
do markiere $\{p, q\}$

Komplexität:

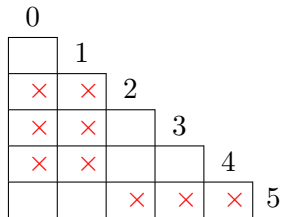
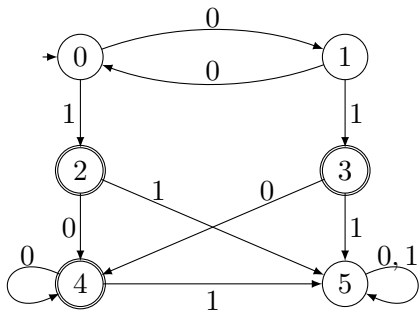
$$O\left(\binom{n}{2} + \binom{n}{2} \binom{n}{2} |\Sigma|\right) = O\left(\frac{n(n-1)}{2} + \frac{n^2(n-1)^2}{4} |\Sigma|\right) = O(n^4)$$

bei fixem Σ .

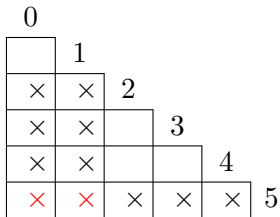
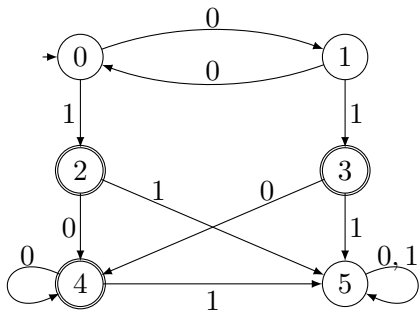
Beispiel 3.51



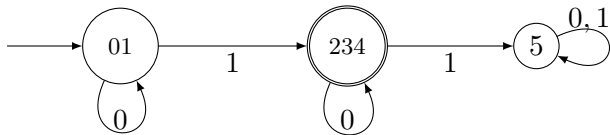
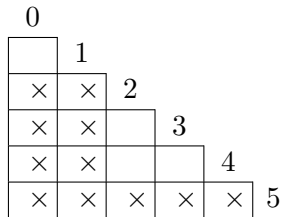
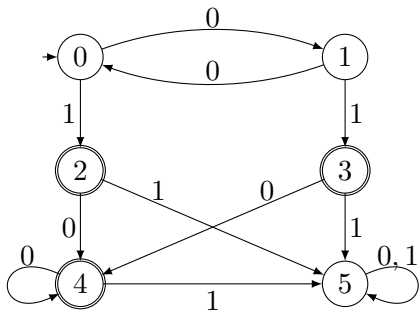
Beispiel 3.51



Beispiel 3.51



Beispiel 3.51



Von $O(n^4)$ zu $O(n^2)$ mit Abhängigkeitsanalyse:

Von $O(n^4)$ zu $O(n^2)$ mit Abhängigkeitsanalyse:

$\{p', q'\}$ unterscheidbar \implies

$\{p, q\}$ unterscheidbar falls $\{p, q\} \xrightarrow{a} \{p', q'\}$

Von $O(n^4)$ zu $O(n^2)$ mit Abhängigkeitsanalyse:

$\{p', q'\}$ unterscheidbar \implies
 $\{p, q\}$ unterscheidbar falls $\{p, q\} \xrightarrow{a} \{p', q'\}$

$D[\{p', q'\}]$: Menge der Paare $\{p, q\}$ wie oben, anfangs leer

Von $O(n^4)$ zu $O(n^2)$ mit Abhängigkeitsanalyse:

$\{p', q'\}$ unterscheidbar \implies
 $\{p, q\}$ unterscheidbar falls $\{p, q\} \xrightarrow{a} \{p', q'\}$

$D[\{p', q'\}]$: Menge der Paare $\{p, q\}$ wie oben, anfangs leer

for all $\{p, q\} \subseteq Q$ mit $p \neq q$, $a \in \Sigma$ **do**

$p' := \delta(p, a)$; $q' := \delta(q, a)$

if $p' \neq q'$ **then** $D[\{p', q'\}] := D[\{p', q'\}] \cup \{\{p, q\}\}$

for all $p' \in F$, $q' \in Q \setminus F$ **do** $mark(\{p', q'\})$

$mark(\{p', q'\}) =$

if $\{p', q'\}$ unmarkiert **then**

markiere $\{p', q'\}$

for all $\{p, q\} \in D[\{p', q'\}]$ **do** $mark(\{p, q\})$

Von $O(n^4)$ zu $O(n^2)$ mit Abhängigkeitsanalyse:

$\{p', q'\}$ unterscheidbar \implies
 $\{p, q\}$ unterscheidbar falls $\{p, q\} \xrightarrow{a} \{p', q'\}$

$D[\{p', q'\}]$: Menge der Paare $\{p, q\}$ wie oben, anfangs leer

for all $\{p, q\} \subseteq Q$ mit $p \neq q$, $a \in \Sigma$ **do**

$p' := \delta(p, a)$; $q' := \delta(q, a)$

if $p' \neq q'$ **then** $D[\{p', q'\}] := D[\{p', q'\}] \cup \{\{p, q\}\}$

for all $p' \in F$, $q' \in Q \setminus F$ **do** $mark(\{p', q'\})$

$mark(\{p', q'\}) =$

if $\{p', q'\}$ unmarkiert **then**

markiere $\{p', q'\}$

for all $\{p, q\} \in D[\{p', q'\}]$ **do** $mark(\{p, q\})$

Komplexität: $O(n^2 + n^2) = O(n^2)$.



John Hopcroft.

An $n \log n$ Algorithm for Minimizing the States in a Finite Automaton. 1971.

Eine weitere Anwendung: Äquivalenztest von DFAs.

Eine weitere Anwendung: Äquivalenztest von DFAs.

- 1 Gegeben DFAs M_1 und M_2 , bilde disjunkte Vereinigung.
(„Male M_1 und M_2 nebeneinander.“)
- 2 Berechne Menge der äquivalenten Zustände.
- 3 $L(M_1) = L(M_2)$ gdw die beiden Startzustände äquivalent sind.

Bisher: Der Minimierungsalgorithmus (zur Erinnerung).

- 1 Entferne alle von q_0 aus nicht erreichbaren Zustände.
- 2 Berechne die *äquivalenten* Zustände des Automaten.
- 3 Kollabiere den Automaten durch Zusammenfassung aller äquivalenten Zustände.

Formale Definition des “kollabierten Automaten”

Eine Relation $\approx \subseteq A \times A$ ist eine **Äquivalenzrelation** falls

- Reflexivität: $\forall a \in A. a \approx a$
- Symmetrie: $\forall a, b \in A. a \approx b \implies b \approx a$
- Transitivität: $\forall a, b, c \in A. a \approx b \wedge b \approx c \implies a \approx c$

Formale Definition des “kollabierten Automaten”

Eine Relation $\approx \subseteq A \times A$ ist eine **Äquivalenzrelation** falls

- Reflexivität: $\forall a \in A. a \approx a$
- Symmetrie: $\forall a, b \in A. a \approx b \implies b \approx a$
- Transitivität: $\forall a, b, c \in A. a \approx b \wedge b \approx c \implies a \approx c$

Äquivalenzklasse:

$$[a]_{\approx} := \{b \mid a \approx b\}$$

Es gilt:

$$[a]_{\approx} = [b]_{\approx} \iff a \approx b$$

Formale Definition des “kollabierten Automaten”

Eine Relation $\approx \subseteq A \times A$ ist eine **Äquivalenzrelation** falls

- Reflexivität: $\forall a \in A. a \approx a$
- Symmetrie: $\forall a, b \in A. a \approx b \implies b \approx a$
- Transitivität: $\forall a, b, c \in A. a \approx b \wedge b \approx c \implies a \approx c$

Äquivalenzklasse:

$$[a]_{\approx} := \{b \mid a \approx b\}$$

Es gilt:

$$[a]_{\approx} = [b]_{\approx} \iff a \approx b$$

Quotientenmenge:

$$A/\approx := \{[a]_{\approx} \mid a \in A\}$$

Im Folgenden sei $M = (Q, \Sigma, \delta, q_0, F)$ ein DFA ohne unerreichbare Zustände und sei $L = L(M)$.

Im Folgenden sei $M = (Q, \Sigma, \delta, q_0, F)$ ein DFA ohne unerreichbare Zustände und sei $L = L(M)$.

Die Sprache des Zustands $q \in Q$ ist die Menge

$$L_M(q) := \{w \in \Sigma^* \mid \hat{\delta}(q, w) \in F\} .$$

Definition 3.52 (Äquivalenz von Zuständen eines DFAs)

$$p \equiv_M q \quad \Leftrightarrow \quad L_M(p) = L_M(q)$$

(Intuition: *Zwei Zustände sind äquivalent wenn sie die gleiche Sprache erkennen.*)

Fakt: \equiv_M ist eine Äquivalenzrelation und die Anzahl der Äquivalenzklassen von \equiv_M (d.h. $|Q/\equiv_M|$) ist die Anzahl der Sprachen, die von den Zuständen von M erkannt werden.

Wir schreiben \equiv statt \equiv_M wenn M klar ist.

Die „Kollabierung“ von M bzgl. \equiv ist der *Quotientenautomat*:

Definition 3.53 (Quotientenautomat)

$$\begin{aligned} M/\equiv & := (Q/\equiv, \Sigma, \delta', [q_0]_{\equiv}, F/\equiv) \\ \delta'([p]_{\equiv}, a) & := [\delta(p, a)]_{\equiv} \end{aligned}$$

Die „Kollabierung“ von M bzgl. \equiv ist der *Quotientenautomat*:

Definition 3.53 (Quotientenautomat)

$$\begin{aligned}M/\equiv & := (Q/\equiv, \Sigma, \delta', [q_0]_{\equiv}, F/\equiv) \\ \delta'([p]_{\equiv}, a) & := [\delta(p, a)]_{\equiv}\end{aligned}$$

Die Definition von δ' ist wohlgeformt da unabhängig von der Wahl des Repräsentanten p :

$$\begin{aligned}[p]_{\equiv} = [p']_{\equiv} & \implies p \equiv p' \implies \delta(p, a) \equiv \delta(p', a) \\ & \implies [\delta(p, a)]_{\equiv} = [\delta(p', a)]_{\equiv}\end{aligned}$$

Die „Kollabierung“ von M bzgl. \equiv ist der *Quotientenautomat*:

Definition 3.53 (Quotientenautomat)

$$\begin{aligned}M/\equiv & := (Q/\equiv, \Sigma, \delta', [q_0]_{\equiv}, F/\equiv) \\ \delta'([p]_{\equiv}, a) & := [\delta(p, a)]_{\equiv}\end{aligned}$$

Die Definition von δ' ist wohlgeformt da unabhängig von der Wahl des Repräsentanten p :

$$\begin{aligned}[p]_{\equiv} = [p']_{\equiv} & \implies p \equiv p' \implies \delta(p, a) \equiv \delta(p', a) \\ & \implies [\delta(p, a)]_{\equiv} = [\delta(p', a)]_{\equiv}\end{aligned}$$

Lemma 3.54

M und M/\equiv erkennen die gleiche Sprache: $L(M/\equiv) = L(M)$.

Die „Kollabierung“ von M bzgl. \equiv ist der *Quotientenautomat*:

Definition 3.53 (Quotientenautomat)

$$\begin{aligned}M/\equiv & := (Q/\equiv, \Sigma, \delta', [q_0]_{\equiv}, F/\equiv) \\ \delta'([p]_{\equiv}, a) & := [\delta(p, a)]_{\equiv}\end{aligned}$$

Die Definition von δ' ist wohlgeformt da unabhängig von der Wahl des Repräsentanten p :

$$\begin{aligned}[p]_{\equiv} = [p']_{\equiv} & \implies p \equiv p' \implies \delta(p, a) \equiv \delta(p', a) \\ & \implies [\delta(p, a)]_{\equiv} = [\delta(p', a)]_{\equiv}\end{aligned}$$

Lemma 3.54

M und M/\equiv erkennen die gleiche Sprache: $L(M/\equiv) = L(M)$.

Beweis: Übung.

Minimalität des Quotientenautomaten

Die Residualsprache von L bzgl. $w \in \Sigma^*$ ist die Menge

$$L^w := \{z \in \Sigma^* \mid wz \in L\} .$$

$L' \subseteq \Sigma^*$ ist Residualsprache von L wenn es w gibt mit $L' = L^w$.

Minimalität des Quotientenautomaten

Die Residualsprache von L bzgl. $w \in \Sigma^*$ ist die Menge

$$L^w := \{z \in \Sigma^* \mid wz \in L\} .$$

$L' \subseteq \Sigma^*$ ist Residualsprache von L wenn es w gibt mit $L' = L^w$.

Definition 3.55 (Äquivalenz von Wörtern bzgl. L)

$$u \equiv_L v \iff L^u = L^v$$

(Intuition: Zwei Wörter sind äquivalent wenn sie die gleiche Residualsprache haben.)

Minimalität des Quotientenautomaten

Die Residualsprache von L bzgl. $w \in \Sigma^*$ ist die Menge

$$L^w := \{z \in \Sigma^* \mid wz \in L\} .$$

$L' \subseteq \Sigma^*$ ist Residualsprache von L wenn es w gibt mit $L' = L^w$.

Definition 3.55 (Äquivalenz von Wörtern bzgl. L)

$$u \equiv_L v \iff L^u = L^v$$

(Intuition: *Zwei Wörter sind äquivalent wenn sie die gleiche Residualsprache haben.*)

Fakt: \equiv_L ist eine Äquivalenzrelation und die Anzahl der Äquivalenzklassen von \equiv_L (d.h. $|\Sigma^*/\equiv_L|$) ist die Anzahl der Residualsprachen von L (kann unendlich sein!).

Erste Beobachtung

Sei $p := \hat{\delta}(q_0, u)$ und $q := \hat{\delta}(q_0, v)$. Es gilt:

$$\begin{aligned} p \equiv_M q &\Leftrightarrow L_M(p) = L_M(q) \\ &\Leftrightarrow \forall w \in \Sigma^*. \hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F \\ &\Leftrightarrow \hat{\delta}(q_0, uw) \in F \Leftrightarrow \hat{\delta}(q_0, vw) \in F \\ &\Leftrightarrow \forall w \in \Sigma^*. uw \in L \Leftrightarrow vw \in L \\ &\Leftrightarrow \forall w \in \Sigma^*. w \in L^u \Leftrightarrow w \in L^v \\ &\Leftrightarrow L^u = L^v \\ &\Leftrightarrow u \equiv_L v \end{aligned}$$

Erste Beobachtung

Sei $p := \hat{\delta}(q_0, u)$ und $q := \hat{\delta}(q_0, v)$. Es gilt:

$$\begin{aligned} p \equiv_M q &\Leftrightarrow L_M(p) = L_M(q) \\ &\Leftrightarrow \forall w \in \Sigma^*. \hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F \\ &\Leftrightarrow \hat{\delta}(q_0, uw) \in F \Leftrightarrow \hat{\delta}(q_0, vw) \in F \\ &\Leftrightarrow \forall w \in \Sigma^*. uw \in L \Leftrightarrow vw \in L \\ &\Leftrightarrow \forall w \in \Sigma^*. w \in L^u \Leftrightarrow w \in L^v \\ &\Leftrightarrow L^u = L^v \\ &\Leftrightarrow u \equiv_L v \end{aligned}$$

Intuition:

- *Nicht-äquivalente Wörter führen zu nicht-äquivalenten Zuständen.*
- *Nicht-äquivalente Zustände werden von nicht-äquivalenten Wörtern erreicht.*

Zweite Beobachtung

Nicht-äquivalente Wörter führen zu nicht-äquivalenten Zuständen
und alle Zustände von M sind erreichbar $\implies |Q/\equiv_M| \geq |\Sigma^*/\equiv_L|$

Zweite Beobachtung

Nicht-äquivalente Wörter führen zu nicht-äquivalenten Zuständen
und alle Zustände von M sind erreichbar $\implies |Q/\equiv_M| \geq |\Sigma^*/\equiv_L|$

Nicht-äquivalente Zustände werden nur von nicht-äquivalenten
Wörtern erreicht $\implies |Q/\equiv_M| \leq |\Sigma^*/\equiv_L|$

Zweite Beobachtung

Nicht-äquivalente Wörter führen zu nicht-äquivalenten Zuständen
und alle Zustände von M sind erreichbar $\implies |Q/\equiv_M| \geq |\Sigma^*/\equiv_L|$

Nicht-äquivalente Zustände werden nur von nicht-äquivalenten
Wörtern erreicht $\implies |Q/\equiv_M| \leq |\Sigma^*/\equiv_L|$

Damit: $|Q/\equiv_M| = |\Sigma^*/\equiv_L|$

Zweite Beobachtung

Nicht-äquivalente Wörter führen zu nicht-äquivalenten Zuständen
und alle Zustände von M sind erreichbar $\implies |Q/\equiv_M| \geq |\Sigma^*/\equiv_L|$

Nicht-äquivalente Zustände werden nur von nicht-äquivalenten
Wörtern erreicht $\implies |Q/\equiv_M| \leq |\Sigma^*/\equiv_L|$

Damit: $|Q/\equiv_M| = |\Sigma^*/\equiv_L|$

Satz 3.56

*Sei M ein DFA ohne unerreichbare Zustände. Der
Quotientenautomat M/\equiv ist ein minimaler DFA für $L(M)$.*

Zweite Beobachtung

Nicht-äquivalente Wörter führen zu nicht-äquivalenten Zuständen und alle Zustände von M sind erreichbar $\implies |Q/\equiv_M| \geq |\Sigma^*/\equiv_L|$

Nicht-äquivalente Zustände werden nur von nicht-äquivalenten Wörtern erreicht $\implies |Q/\equiv_M| \leq |\Sigma^*/\equiv_L|$

Damit: $|Q/\equiv_M| = |\Sigma^*/\equiv_L|$

Satz 3.56

Sei M ein DFA ohne unerreichbare Zustände. Der Quotientenautomat M/\equiv ist ein minimaler DFA für $L(M)$.

Beweis:

Sei M' ein beliebiger DFA für $L(M)$. Es gilt:

$$|Q'| \geq |Q'/\equiv_{M'}| = |\Sigma^*/\equiv_L| = |Q/\equiv_M| = |\text{Zustände von } M/\equiv|$$



Sei \mathcal{R}_L die Menge der Residualsprachen von L .

Definition 3.57 (Kanonischer Minimalautomat)

$$M_L := (\mathcal{R}_L, \Sigma, \delta_L, L, F_L)$$

mit $\delta_L(R, a) := R^a$ und $F_L := \{R \in \mathcal{R}_L \mid \epsilon \in R\}$.

Sei \mathcal{R}_L die Menge der Residualsprachen von L .

Definition 3.57 (Kanonischer Minimalautomat)

$$M_L := (\mathcal{R}_L, \Sigma, \delta_L, L, F_L)$$

mit $\delta_L(R, a) := R^a$ und $F_L := \{R \in \mathcal{R}_L \mid \epsilon \in R\}$.

δ_L ist wohldefiniert und $\hat{\delta}_L(R, w) = R^w$. Jeder Zustand R erkennt die Sprache R und somit $L(M_L) = L$.

Sei \mathcal{R}_L die Menge der Residualsprachen von L .

Definition 3.57 (Kanonischer Minimalautomat)

$$M_L := (\mathcal{R}_L, \Sigma, \delta_L, L, F_L)$$

mit $\delta_L(R, a) := R^a$ und $F_L := \{R \in \mathcal{R}_L \mid \epsilon \in R\}$.

δ_L ist wohldefiniert und $\hat{\delta}_L(R, w) = R^w$. Jeder Zustand R erkennt die Sprache R und somit $L(M_L) = L$.

Satz 3.58

Jeder minimaler DFA für eine reguläre Sprache L unterscheidet sich vom kanonischen Minimalautomaten M_L nur durch eine Umbenennung der Zustände.

Beweis:

(Skizze.) Sei M minimaler DFA für L . Jeder Zustand von M erkennt eine andere Residualsprache von L . Das gleiche gilt für M_L . Die Bijektion, die jedem Zustand von M den Zustand von M_L mit der selben Sprache zuordnet, ist die gesuchte Umbenennung. □

Satz 3.59

Eine Sprache $L \subseteq \Sigma^$ ist genau dann regulär, wenn sie endlich viele Residualsprachen hat.*

Satz 3.59

Eine Sprache $L \subseteq \Sigma^$ ist genau dann regulär, wenn sie endlich viele Residualsprachen hat.*

Beweis:

„ \implies “: Ist L regulär, so wird L von einem DFA M akzeptiert. Daher hat L so viele Residualsprachen wie M/\equiv_M Zustände.

Satz 3.59

Eine Sprache $L \subseteq \Sigma^$ ist genau dann regulär, wenn sie endlich viele Residualsprachen hat.*

Beweis:

„ \implies “: Ist L regulär, so wird L von einem DFA M akzeptiert.

Daher hat L so viele Residualsprachen wie M/\equiv_M Zustände.

„ \impliedby “: Hat L endlich viele Residualsprachen, so ist M_L ein DFA mit $L(M_L) = L$. □

Wie sieht M_L aus wenn L *nicht* regulär ist?

Wie sieht M_L aus wenn L nicht regulär ist?

Beispiel 3.60

$L = \{w \in \{0, 1\}^* \mid \#_0(w) = \#_1(w)\}$ $\#_a(w) = \text{Anzahl der } a \text{ in } w$

Wie sieht M_L aus wenn L nicht regulär ist?

Beispiel 3.60

$L = \{w \in \{0, 1\}^* \mid \#_0(w) = \#_1(w)\}$ $\#_a(w) = \text{Anzahl der } a \text{ in } w$

↓

L^ϵ

Wie sieht M_L aus wenn L nicht regulär ist?

Beispiel 3.60

$L = \{w \in \{0, 1\}^* \mid \#_0(w) = \#_1(w)\}$ $\#_a(w) = \text{Anzahl der } a \text{ in } w$

$$\begin{array}{ccc} & \downarrow & \\ L^\epsilon & \xrightarrow{1} & L^1 \end{array}$$

Wie sieht M_L aus wenn L nicht regulär ist?

Beispiel 3.60

$L = \{w \in \{0, 1\}^* \mid \#_0(w) = \#_1(w)\}$ $\#_a(w) = \text{Anzahl der } a \text{ in } w$

$$\begin{array}{ccccc} & \downarrow & & & \\ L^\epsilon & \xrightarrow{1} & L^1 & \xrightarrow{1} & L^{11} \end{array}$$

Wie sieht M_L aus wenn L nicht regulär ist?

Beispiel 3.60

$L = \{w \in \{0, 1\}^* \mid \#_0(w) = \#_1(w)\}$ $\#_a(w) = \text{Anzahl der } a \text{ in } w$

$$\begin{array}{ccccccc} & \downarrow & & & & & \\ & L^\epsilon & \xrightarrow{1} & L^1 & \xrightarrow{1} & L^{11} & \xrightarrow{1} \dots \end{array}$$

Wie sieht M_L aus wenn L nicht regulär ist?

Beispiel 3.60

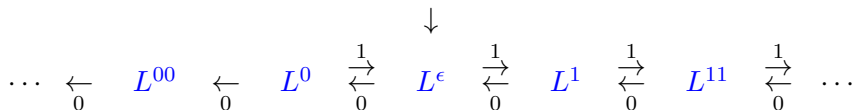
$L = \{w \in \{0, 1\}^* \mid \#_0(w) = \#_1(w)\}$ $\#_a(w) = \text{Anzahl der } a \text{ in } w$

$$\begin{array}{ccccccc} & & \downarrow & & & & \\ L^0 & \xleftarrow[0]{} & L^\epsilon & \xrightarrow{1} & L^1 & \xrightarrow{1} & L^{11} & \xrightarrow{1} & \dots \end{array}$$

Wie sieht M_L aus wenn L nicht regulär ist?

Beispiel 3.60

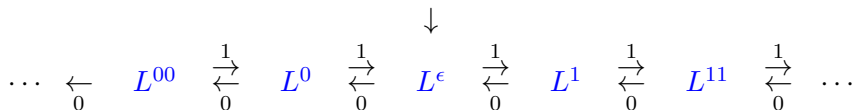
$L = \{w \in \{0, 1\} \mid \#_0(w) = \#_1(w)\}$ $\#_a(w) = \text{Anzahl der } a \text{ in } w$



Wie sieht M_L aus wenn L nicht regulär ist?

Beispiel 3.60

$L = \{w \in \{0, 1\} \mid \#_0(w) = \#_1(w)\}$ $\#_a(w) = \text{Anzahl der } a \text{ in } w$



Wie sieht M_L aus wenn L nicht regulär ist?

Beispiel 3.60

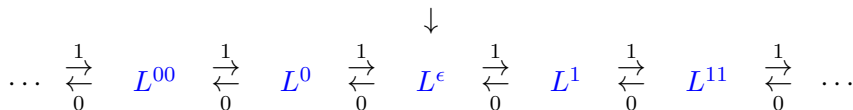
$L = \{w \in \{0, 1\}^* \mid \#_0(w) = \#_1(w)\}$ $\#_a(w) = \text{Anzahl der } a \text{ in } w$

$$\begin{array}{cccccccccccc} \dots & \begin{array}{c} \xrightarrow{1} \\ \xleftarrow{0} \end{array} & L^{00} & \begin{array}{c} \xrightarrow{1} \\ \xleftarrow{0} \end{array} & L^0 & \begin{array}{c} \xrightarrow{1} \\ \xleftarrow{0} \end{array} & L^\epsilon & \begin{array}{c} \xrightarrow{1} \\ \xleftarrow{0} \end{array} & L^1 & \begin{array}{c} \xrightarrow{1} \\ \xleftarrow{0} \end{array} & L^{11} & \begin{array}{c} \xrightarrow{1} \\ \xleftarrow{0} \end{array} & \dots \\ & & & & & & \downarrow & & & & & & \end{array}$$

Wie sieht M_L aus wenn L nicht regulär ist?

Beispiel 3.60

$L = \{w \in \{0, 1\} \mid \#_0(w) = \#_1(w)\}$ $\#_a(w) = \text{Anzahl der } a \text{ in } w$

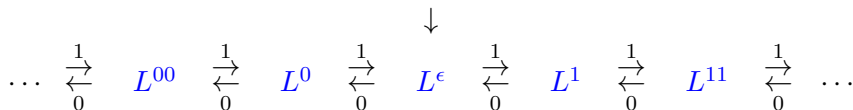


Endzustände?

Wie sieht M_L aus wenn L nicht regulär ist?

Beispiel 3.60

$L = \{w \in \{0, 1\}^* \mid \#_0(w) = \#_1(w)\}$ $\#_a(w) = \text{Anzahl der } a \text{ in } w$



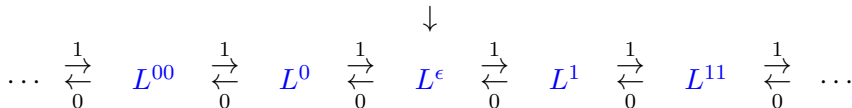
Endzustände?

$$L^{1^i} = \{w \mid$$

Wie sieht M_L aus wenn L nicht regulär ist?

Beispiel 3.60

$L = \{w \in \{0, 1\}^* \mid \#_0(w) = \#_1(w)\}$ $\#_a(w) = \text{Anzahl der } a \text{ in } w$



Endzustände?

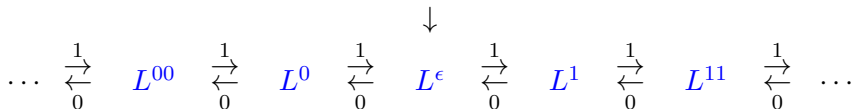
$$L^{1^i} = \{w \mid \#_1(w) = \#_0(w) + i\}$$

Warum $L^{1^i} \neq L^{1^j}$ falls $i \neq j$?

Wie sieht M_L aus wenn L nicht regulär ist?

Beispiel 3.60

$L = \{w \in \{0, 1\}^* \mid \#_0(w) = \#_1(w)\}$ $\#_a(w) = \text{Anzahl der } a \text{ in } w$



Endzustände?

$$L^{1^i} = \{w \mid \#_1(w) = \#_0(w) + i\}$$

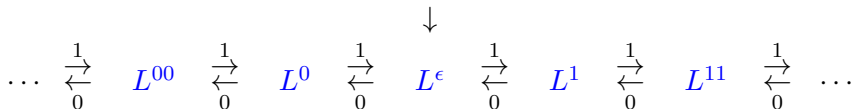
Warum $L^{1^i} \neq L^{1^j}$ falls $i \neq j$?

Vollständige Methode um **Nichtregularität** von L zu zeigen:

Wie sieht M_L aus wenn L nicht regulär ist?

Beispiel 3.60

$L = \{w \in \{0, 1\}^* \mid \#_0(w) = \#_1(w)\}$ $\#_a(w) = \text{Anzahl der } a \text{ in } w$



Endzustände?

$$L^{1^i} = \{w \mid \#_1(w) = \#_0(w) + i\}$$

Warum $L^{1^i} \neq L^{1^j}$ falls $i \neq j$?

Vollständige Methode um **Nichtregularität** von L zu zeigen:

Gib **unendliche** Menge w_1, w_2, \dots an mit $L^{w_i} \neq L^{w_j}$ falls $i \neq j$.

Bemerkung

Eindeutigkeit des minimalen Automaten (modulo Umbenennung der Zustände) gilt nur bei DFAs, nicht bei NFAs!

Bemerkung

Eindeutigkeit des minimalen Automaten (modulo Umbenennung der Zustände) gilt nur bei DFAs, nicht bei NFAs!

Aufgabe: Finde Gegenbeispiel für NFAs

Rückblick reguläre Sprachen

- DFA, NFA, ϵ -NFA und RE definieren die gleiche Sprachklasse.
 - Potenzmengenkonstruktion (exponentiell)
 - Strukturelle Übersetzung von RE nach ϵ -NFA
 - Übersetzung NFA nach RE, zB über Gleichungssysteme (exponentiell)
- Regularitätserhaltende Konstruktionen auf Sprachen bzw Automaten:
 - $\cup, \cap, \dots, R, \dots$
 - Für welche Darstellung wie teuer?
(Komplement, Produkt, ...)
 - NFA kompakt aber oft teuer; DFA oft billiger
- Entscheidungsprobleme auf Automaten und REs:
 - Direkt entscheidbar?
Auf Automat? (Oft mit Erreichbarkeit) Auf RE?
 - Reduzierbar auf anderes Problem?
ZB $L(A) \subseteq L(B)$ auf $L(A) \cap \overline{L(B)} = \emptyset$
 - Für welche Darstellung wie teuer?

- Pumping-Lemma
- Äquivalenzen \equiv zw REs:
 - Standardregeln: Kommutativität etc, Beweis mit $L(\cdot)$
 - $\alpha \equiv \beta$ entscheidbar da $L(\alpha) = L(\beta)$ über Automaten entscheidbar
 - Auch für REs mit Variablen entscheidbar: betrachte Variablen als Konstanten
 - Gleichungssysteme lösbar durch Variablenelimination mit Ardens Lemma
- Minimierung
 - Algorithmen zur Kollabierung
 - Relationen \equiv_M und \equiv_L
 - Quotientenautomat M/\equiv_M minimal, da gleich groß wie kanonischer minimaler Automat $M_{L(M)}$
 - L genau dann regulär wenn sie endlich viele Residualsprachen hat.

4. Kontextfreie Sprachen

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$

$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\langle \text{Factor} \rangle \rightarrow a$

$\langle \text{Factor} \rangle \rightarrow (\langle \text{Expr} \rangle)$

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$

$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\langle \text{Factor} \rangle \rightarrow a$

$\langle \text{Factor} \rangle \rightarrow (\langle \text{Expr} \rangle)$

Eine (Links)Ableitung:

$\langle \text{Expr} \rangle \rightarrow$

$\rightarrow a * (a + a)$

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$

$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\langle \text{Factor} \rangle \rightarrow a$

$\langle \text{Factor} \rangle \rightarrow (\langle \text{Expr} \rangle)$

Eine (Links)Ableitung:

$\langle \text{Expr} \rangle \rightarrow$

$\rightarrow a * (a + a)$

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$

$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\langle \text{Factor} \rangle \rightarrow a$

$\langle \text{Factor} \rangle \rightarrow (\langle \text{Expr} \rangle)$

Eine (Links)Ableitung:

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$

$\rightarrow a * (a + a)$

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$

$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\langle \text{Factor} \rangle \rightarrow a$

$\langle \text{Factor} \rangle \rightarrow (\langle \text{Expr} \rangle)$

Eine (Links)Ableitung:

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\rightarrow a * (a + a)$

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$

$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\langle \text{Factor} \rangle \rightarrow a$

$\langle \text{Factor} \rangle \rightarrow (\langle \text{Expr} \rangle)$

Eine (Links)Ableitung:

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\rightarrow a * (a + a)$

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$

$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\langle \text{Factor} \rangle \rightarrow a$

$\langle \text{Factor} \rangle \rightarrow (\langle \text{Expr} \rangle)$

Eine (Links)Ableitung:

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle$

$\rightarrow a * (a + a)$

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$

$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\langle \text{Factor} \rangle \rightarrow a$

$\langle \text{Factor} \rangle \rightarrow (\langle \text{Expr} \rangle)$

Eine (Links)Ableitung:

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle$

$\rightarrow a * (a + a)$

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$

$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\langle \text{Factor} \rangle \rightarrow a$

$\langle \text{Factor} \rangle \rightarrow (\langle \text{Expr} \rangle)$

Eine (Links)Ableitung:

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle$

$\rightarrow a * (a + a)$

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$

$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\langle \text{Factor} \rangle \rightarrow a$

$\langle \text{Factor} \rangle \rightarrow (\langle \text{Expr} \rangle)$

Eine (Links)Ableitung:

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle$

$\rightarrow a * (\langle \text{Expr} \rangle)$

$\rightarrow a * (a + a)$

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$

$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\langle \text{Factor} \rangle \rightarrow a$

$\langle \text{Factor} \rangle \rightarrow (\langle \text{Expr} \rangle)$

Eine (Links)Ableitung:

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle$

$\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle)$

$\rightarrow a * (a + a)$

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$

$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\langle \text{Factor} \rangle \rightarrow a$

$\langle \text{Factor} \rangle \rightarrow (\langle \text{Expr} \rangle)$

Eine (Links)Ableitung:

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle$

$\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle)$

$\rightarrow a * (a + a)$

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$

$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle$

$\langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\langle \text{Factor} \rangle \rightarrow a$

$\langle \text{Factor} \rangle \rightarrow (\langle \text{Expr} \rangle)$

Eine (Links)Ableitung:

$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$

$\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle$

$\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle)$

$\rightarrow a * (\langle \text{Term} \rangle + \langle \text{Term} \rangle)$

$\rightarrow a * (a + a)$

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

$$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$$
$$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle$$
$$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle$$
$$\langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$$
$$\langle \text{Factor} \rangle \rightarrow a$$
$$\langle \text{Factor} \rangle \rightarrow (\langle \text{Expr} \rangle)$$

Eine (Links)Ableitung:

$$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$$
$$\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle$$
$$\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle)$$
$$\rightarrow a * (\langle \text{Term} \rangle + \langle \text{Term} \rangle)$$
$$\rightarrow a * (a + a)$$

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

$$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$$

$$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle$$

$$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle$$

$$\langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$$

$$\langle \text{Factor} \rangle \rightarrow a$$

$$\langle \text{Factor} \rangle \rightarrow (\langle \text{Expr} \rangle)$$

Eine (Links)Ableitung:

$$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$$

$$\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle$$

$$\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle)$$

$$\rightarrow a * (\langle \text{Term} \rangle + \langle \text{Term} \rangle) \rightarrow a * (\langle \text{Factor} \rangle + \langle \text{Term} \rangle) \\ \rightarrow a * (a + a)$$

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

$$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$$
$$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle$$
$$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle$$
$$\langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$$
$$\langle \text{Factor} \rangle \rightarrow a$$
$$\langle \text{Factor} \rangle \rightarrow (\langle \text{Expr} \rangle)$$

Eine (Links)Ableitung:

$$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$$
$$\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle$$
$$\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle)$$
$$\rightarrow a * (\langle \text{Term} \rangle + \langle \text{Term} \rangle) \rightarrow a * (\langle \text{Factor} \rangle + \langle \text{Term} \rangle)$$
$$\rightarrow a * (a + a)$$

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

$$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$$

$$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle$$

$$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle$$

$$\langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$$

$$\langle \text{Factor} \rangle \rightarrow a$$

$$\langle \text{Factor} \rangle \rightarrow (\langle \text{Expr} \rangle)$$

Eine (Links)Ableitung:

$$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$$

$$\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle$$

$$\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle)$$

$$\rightarrow a * (\langle \text{Term} \rangle + \langle \text{Term} \rangle) \rightarrow a * (\langle \text{Factor} \rangle + \langle \text{Term} \rangle)$$

$$\rightarrow a * (a + \langle \text{Term} \rangle) \qquad \qquad \qquad \rightarrow a * (a + a)$$

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

$$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle$$
$$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle$$
$$\langle \text{Term} \rangle \rightarrow \langle \text{Factor} \rangle$$
$$\langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$$
$$\langle \text{Factor} \rangle \rightarrow a$$
$$\langle \text{Factor} \rangle \rightarrow (\langle \text{Expr} \rangle)$$

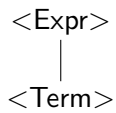
Eine (Links)Ableitung:

$$\langle \text{Expr} \rangle \rightarrow \langle \text{Term} \rangle \rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle$$
$$\rightarrow \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \rightarrow a * \langle \text{Factor} \rangle$$
$$\rightarrow a * (\langle \text{Expr} \rangle) \rightarrow a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle)$$
$$\rightarrow a * (\langle \text{Term} \rangle + \langle \text{Term} \rangle) \rightarrow a * (\langle \text{Factor} \rangle + \langle \text{Term} \rangle)$$
$$\rightarrow a * (a + \langle \text{Term} \rangle) \rightarrow a * (a + \langle \text{Factor} \rangle) \rightarrow a * (a + a)$$

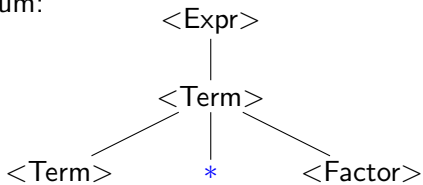
Der Syntaxbaum:

<Expr>

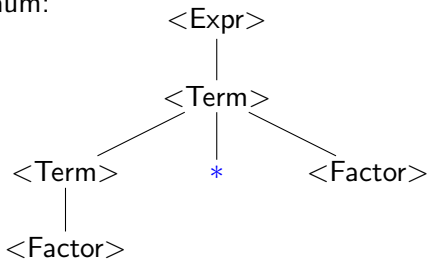
Der Syntaxbaum:



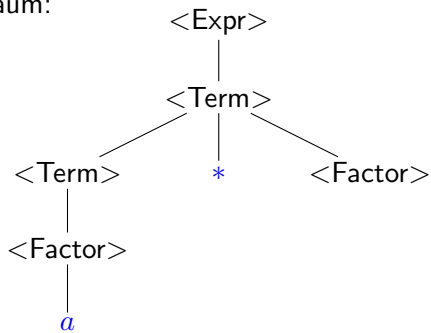
Der Syntaxbaum:



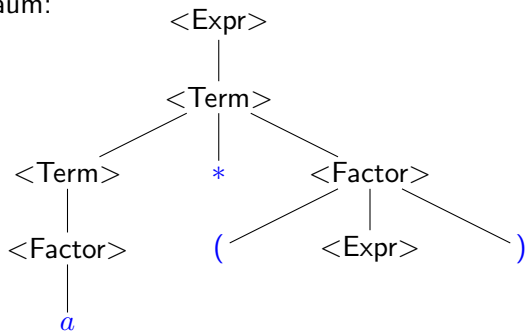
Der Syntaxbaum:



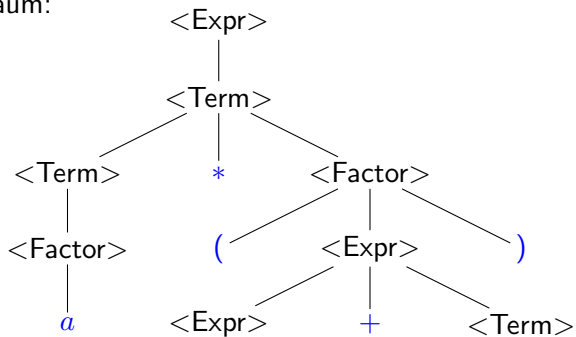
Der Syntaxbaum:



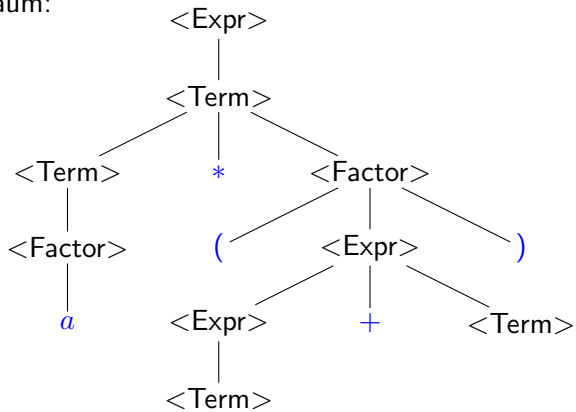
Der Syntaxbaum:



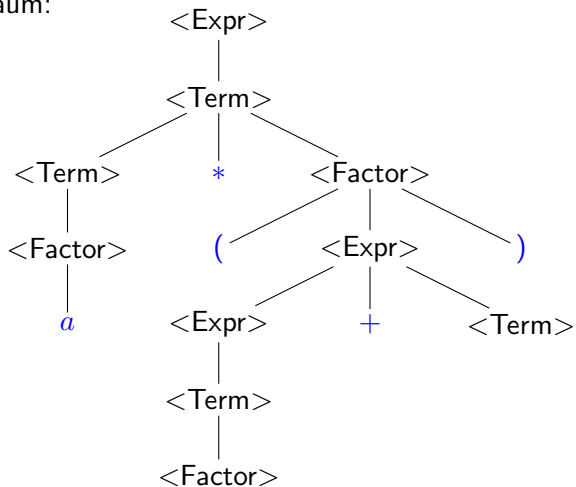
Der Syntaxbaum:



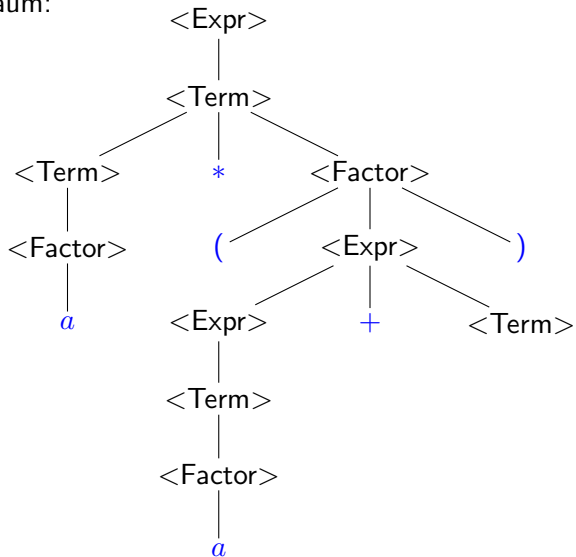
Der Syntaxbaum:



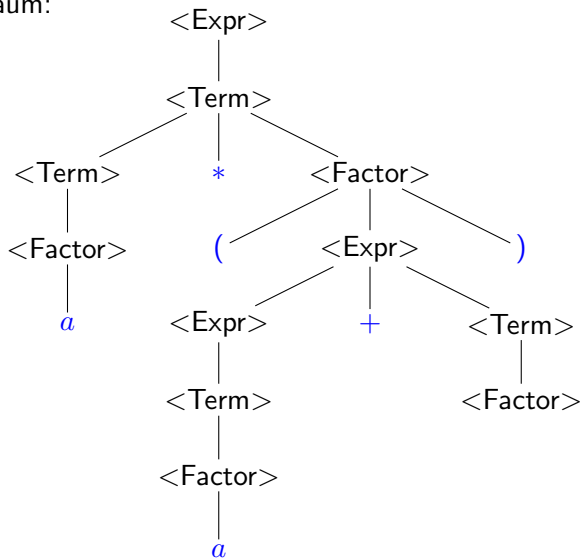
Der Syntaxbaum:



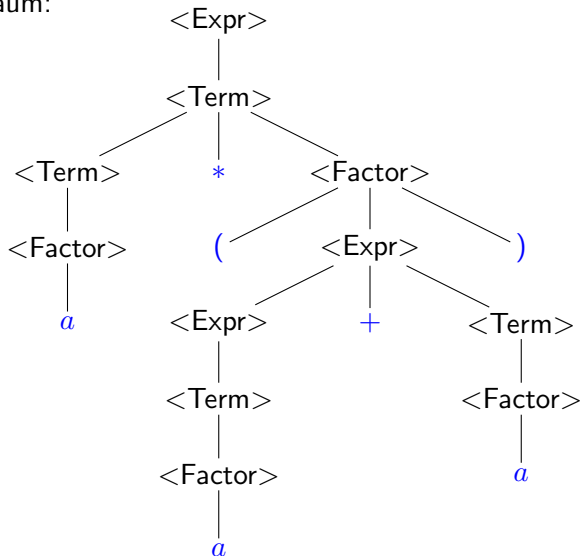
Der Syntaxbaum:



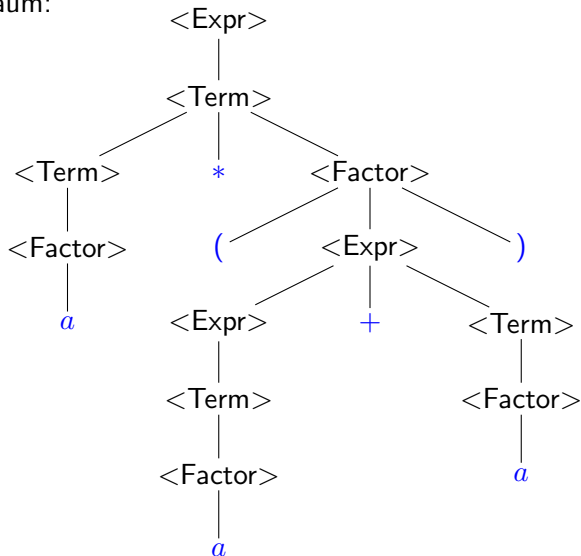
Der Syntaxbaum:



Der Syntaxbaum:



Der Syntaxbaum:



Die Blätter des Baums, von links nach rechts gelesen, ergeben das abgeleitete Wort

Bemerkungen

- Der vollständige Syntaxbaum enthält die gesamte Information über die Ableitung, bis auf die (irrelevante) Reihenfolge des Aufbaus.
- Kontextfreie Grammatiken dienen zur Spezifikation von Sprachen. Das **Parsen** ist:
 - Die Überprüfung, ob ein Wort von einer Grammatik abgeleitet werden kann, bzw
 - Die Erzeugung des Syntaxbaums (*parse tree*).

Parsen ist die Transformation eines Worts in einen Syntaxbaum.

Definition 4.2

Eine **kontextfreie Grammatik** $G = (V, \Sigma, P, S)$ ist ein 4-Tupel:

V ist eine endliche Menge von **Nichtterminalzeichen**
(oder **Variablen**),

Σ ist ein Alphabet von **Terminalzeichen**, disjunkt von V ,

$P \subseteq V \times (V \cup \Sigma)^*$ ist eine endliche Menge von **Produktionen** und

$S \in V$ ist das **Startsymbol**.

Definition 4.2

Eine **kontextfreie Grammatik** $G = (V, \Sigma, P, S)$ ist ein 4-Tupel:

V ist eine endliche Menge von **Nichtterminalzeichen**
(oder **Variablen**),

Σ ist ein Alphabet von **Terminalzeichen**, disjunkt von V ,

$P \subseteq V \times (V \cup \Sigma)^*$ ist eine endliche Menge von **Produktionen** und

$S \in V$ ist das **Startsymbol**.

Konventionen:

- A, B, C, \dots sind Nichtterminale,
- a, b, c, \dots (und Sonderzeichen wie $+, *, \dots$) sind Terminale,
- $\alpha, \beta, \gamma, \dots \in (V \cup \Sigma)^*$
- Produktionen schreiben wir $A \rightarrow \alpha$ statt $(A, \alpha) \in P$.
- Statt $A \rightarrow \alpha_1, A \rightarrow \alpha_2, A \rightarrow \alpha_3$ schreiben wir einfach

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3$$

Beispiel 4.3 (Arithmetische Ausdrücke)

$$V = \{E, T, F\}$$

$$\Sigma = \{a, +, *, (,)\}$$

$$P =$$

$$\left\{ \begin{array}{l} E \rightarrow T \mid E + T \\ T \rightarrow F \mid T * F \\ F \rightarrow a \mid (E) \end{array} \right\}$$

$$S = E$$

Definition 4.4

Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ induziert eine **Ableitungsrelation** \rightarrow_G auf Wörtern über $V \cup \Sigma$:

$$\alpha \rightarrow_G \beta$$

gdw es eine Regel $A \rightarrow \gamma$ in P gibt, und Wörter α_1, α_2 , so dass

$$\alpha = \alpha_1 A \alpha_2 \quad \text{und} \quad \beta = \alpha_1 \gamma \alpha_2$$

Beispiel:

$$a + T + a \quad \rightarrow_G \quad a + T * F + a$$

Definition 4.5 (Reflexive transitive Hülle)

$$\alpha \rightarrow_G^0 \alpha$$

$$\alpha \rightarrow_G^{n+1} \gamma \quad :\Leftrightarrow \quad \exists \beta. \alpha \rightarrow_G^n \beta \rightarrow_G \gamma$$

$$\alpha \rightarrow_G^* \beta \quad :\Leftrightarrow \quad \exists n. \alpha \rightarrow_G^n \beta$$

$$\alpha \rightarrow_G^+ \beta \quad :\Leftrightarrow \quad \exists n > 0. \alpha \rightarrow_G^n \beta$$

Beispiel: $E \rightarrow_G^{11} a * (a + a)$ und daher $E \rightarrow_G^* a * (a + a)$.

Definition 4.5 (Reflexive transitive Hülle)

$$\alpha \rightarrow_G^0 \alpha$$

$$\alpha \rightarrow_G^{n+1} \gamma \quad :\Leftrightarrow \quad \exists \beta. \alpha \rightarrow_G^n \beta \rightarrow_G \gamma$$

$$\alpha \rightarrow_G^* \beta \quad :\Leftrightarrow \quad \exists n. \alpha \rightarrow_G^n \beta$$

$$\alpha \rightarrow_G^+ \beta \quad :\Leftrightarrow \quad \exists n > 0. \alpha \rightarrow_G^n \beta$$

Beispiel: $E \rightarrow_G^{11} a * (a + a)$ und daher $E \rightarrow_G^* a * (a + a)$.

Wir nennen

$$\alpha_1 \rightarrow_G \alpha_2 \rightarrow_G \cdots \rightarrow_G \alpha_n$$

eine **Linksableitung** gdw in jedem Schritt das linkeste Nichtterminal in α_i ersetzt wird.

Definition 4.6 (Kontextfreie Sprache)

Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ erzeugt die Sprache

$$L(G) := \{w \in \Sigma^* \mid S \rightarrow_G^* w\}$$

Eine Sprache $L \subseteq \Sigma^*$ heißt **kontextfrei** gdw es eine kontextfreie Grammatik G gibt mit $L = L(G)$.

Definition 4.6 (Kontextfreie Sprache)

Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ erzeugt die Sprache

$$L(G) := \{w \in \Sigma^* \mid S \rightarrow_G^* w\}$$

Eine Sprache $L \subseteq \Sigma^*$ heißt **kontextfrei** gdw es eine kontextfreie Grammatik G gibt mit $L = L(G)$.

Abkürzungen:

CFG Kontextfreie Grammatik (*context-free grammar*)

CFL Kontextfreie Sprache (*context-free language*)

Konvention:

Ist G aus dem Kontext eindeutig ersichtlich, so schreibt man auch nur $\alpha \rightarrow \beta$ statt $\alpha \rightarrow_G \beta$.

Beispiel 4.7

Die nicht-reguläre Sprache $L = \{a^n b^n \mid n \in \mathbb{N}\}$ ist kontextfrei, da sie von der CFG

$$S \rightarrow aSb \mid \epsilon$$

erzeugt wird. Genauer: $L = L(G)$ wobei $G = (V, \Sigma, P, S)$ mit

$$V = \{S\}$$

$$\Sigma = \{a, b\}$$

$$P = \{S \rightarrow aSb \mid \epsilon\}$$

Der unendliche Baum aller möglichen (Links-)Ableitungen:

$$\begin{array}{ccccccc} S & \rightarrow & aSb & \rightarrow & a^2Sb^2 & \rightarrow & a^3Sb^3 & \rightarrow & \dots \\ & \searrow & & \searrow & & \searrow & & \searrow & \\ & & \epsilon & & ab & & a^2b^2 & & a^3b^3 \end{array}$$

Beispiel 4.8

Die nicht-reguläre Sprache der Palindrome ($w = w^R$) über $\{a, b\}$ wird von folgender CFG erzeugt:

$$S \rightarrow \epsilon \mid a \mid b \mid aSa \mid bSb$$

Der Anfang des unendlichen Baums aller (Links-)Ableitungen (Achtung, dies ist *kein* Syntaxbaum!):

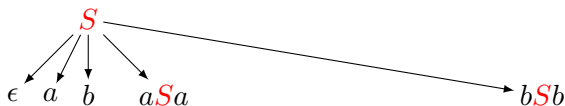
S

Beispiel 4.8

Die nicht-reguläre Sprache der Palindrome ($w = w^R$) über $\{a, b\}$ wird von folgender CFG erzeugt:

$$S \rightarrow \epsilon \mid a \mid b \mid aSa \mid bSb$$

Der Anfang des unendlichen Baums aller (Links-)Ableitungen (Achtung, dies ist *kein* Syntaxbaum!):

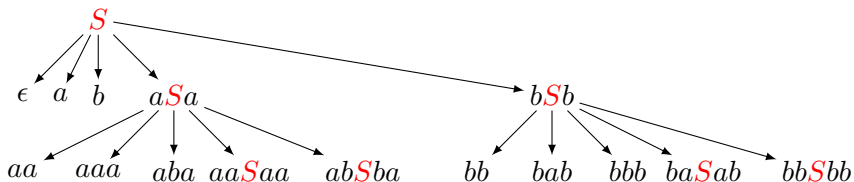


Beispiel 4.8

Die nicht-reguläre Sprache der Palindrome ($w = w^R$) über $\{a, b\}$ wird von folgender CFG erzeugt:

$$S \rightarrow \epsilon \mid a \mid b \mid aSa \mid bSb$$

Der Anfang des unendlichen Baums aller (Links-)Ableitungen (Achtung, dies ist *kein* Syntaxbaum!):



Lemma 4.9 (Dekompositionslemma)

$$\alpha_1 \alpha_2 \rightarrow_G^n \beta$$

$$\Leftrightarrow$$

$$\exists \beta_1, \beta_2, n_1, n_2. \beta = \beta_1 \beta_2 \wedge n = n_1 + n_2 \wedge \alpha_i \rightarrow_G^{n_i} \beta_i \quad (i = 1, 2)$$

Beweis:

Übung!



4.2 Induktive Definitionen, Syntaxbäume und Ableitungen

- 1 Beispiel: Balancierte Klammern
- 2 Induktive Definitionen und Syntaxbäume allgemein
- 3 Äquivalenz von Ableitung, Syntaxbaum und induktiver Erzeugung

Beispiel 4.10

$$S \rightarrow \epsilon \mid [S] \mid SS$$

Um zu zeigen, dass diese Grammatik die Menge aller balancierten Klammerwörter in $\{[,]\}^*$ erzeugt, betrachten wir die Grammatik als induktive Definition einer Sprache $L_G(S)$:

Beispiel 4.10

$$S \rightarrow \epsilon \mid [S] \mid SS$$

Um zu zeigen, dass diese Grammatik die Menge aller balancierten Klammerwörter in $\{[,]\}^*$ erzeugt, betrachten wir die Grammatik als induktive Definition einer Sprache $L_G(S)$:

Beispiel 4.10

$$S \rightarrow \epsilon \mid [S] \mid SS$$

Um zu zeigen, dass diese Grammatik die Menge aller balancierten Klammerwörter in $\{[,]\}^*$ erzeugt, betrachten wir die Grammatik als induktive Definition einer Sprache $L_G(S)$:

$$\epsilon \in L_G(S)$$

Beispiel 4.10

$$S \rightarrow \epsilon \mid [S] \mid SS$$

Um zu zeigen, dass diese Grammatik die Menge aller balancierten Klammerwörter in $\{[,]\}^*$ erzeugt, betrachten wir die Grammatik als induktive Definition einer Sprache $L_G(S)$:

$$u \in L_G(S) \implies [u] \in L_G(S)$$
$$\epsilon \in L_G(S)$$

Beispiel 4.10

$$S \rightarrow \epsilon \mid [S] \mid SS$$

Um zu zeigen, dass diese Grammatik die Menge aller balancierten Klammerwörter in $\{[,]\}^*$ erzeugt, betrachten wir die Grammatik als induktive Definition einer Sprache $L_G(S)$:

		$\epsilon \in L_G(S)$
$u \in L_G(S)$	\implies	$[u] \in L_G(S)$
$u \in L_G(S) \wedge v \in L_G(S)$	\implies	$uv \in L_G(S)$

Beispiel 4.10

$$S \rightarrow \epsilon \mid [S] \mid SS$$

Um zu zeigen, dass diese Grammatik die Menge aller balancierten Klammerwörter in $\{[,]\}^*$ erzeugt, betrachten wir die Grammatik als induktive Definition einer Sprache $L_G(S)$:

		$\epsilon \in L_G(S)$
$u \in L_G(S)$	\implies	$[u] \in L_G(S)$
$u \in L_G(S) \wedge v \in L_G(S)$	\implies	$uv \in L_G(S)$

Damit gilt zB:

$$\begin{aligned} \epsilon \in L_G(S) &\implies [] \in L_G(S) \implies [[]] \in L_G(S) \\ &\implies [[]] [] \in L_G(S) \end{aligned}$$

Bemerkungen

- Die Produktionen (\rightarrow) erzeugen Wörter *top-down*, d.h. von einem Nichtterminal zu einem Wort hin.
- Die induktive Definition (\Rightarrow) erzeugt Wörter *bottom-up*, d.h. sie setzt kleinere Wörter zu größeren zusammen.
- Die induktive Definition betrachtet nur Wörter aus Σ^* .

Zur induktiven Definition von $L_G(S)$ gehört ein Induktionsprinzip:

Um zu zeigen, dass für alle $u \in L_G(S)$ eine Eigenschaft $P(u)$ gilt, dh $\forall u \in L_G(S). P(u)$, zeige:



Zur induktiven Definition von $L_G(S)$ gehört ein Induktionsprinzip:

Um zu zeigen, dass für alle $u \in L_G(S)$ eine Eigenschaft $P(u)$ gilt, dh $\forall u \in L_G(S). P(u)$, zeige:

$P(\epsilon)$

Zur induktiven Definition von $L_G(S)$ gehört ein Induktionsprinzip:

Um zu zeigen, dass für alle $u \in L_G(S)$ eine Eigenschaft $P(u)$ gilt, dh $\forall u \in L_G(S). P(u)$, zeige:

$$P(u) \implies P([u])$$

$P(\epsilon)$

Zur induktiven Definition von $L_G(S)$ gehört ein Induktionsprinzip:

Um zu zeigen, dass für alle $u \in L_G(S)$ eine Eigenschaft $P(u)$ gilt, dh $\forall u \in L_G(S). P(u)$, zeige:

		$P(\epsilon)$
	$P(u)$	$\implies P([u])$
	$P(u) \wedge P(v)$	$\implies P(uv)$

Zur induktiven Definition von $L_G(S)$ gehört ein Induktionsprinzip:

Um zu zeigen, dass für alle $u \in L_G(S)$ eine Eigenschaft $P(u)$ gilt, dh $\forall u \in L_G(S). P(u)$, zeige:

		$P(\epsilon)$
	$P(u)$	$\implies P([u])$
	$P(u) \wedge P(v)$	$\implies P(uv)$

„Induktion über die Erzeugung von u “

Zur induktiven Definition von $L_G(S)$ gehört ein Induktionsprinzip:

Um zu zeigen, dass für alle $u \in L_G(S)$ eine Eigenschaft $P(u)$ gilt, dh $\forall u \in L_G(S). P(u)$, zeige:

		$P(\epsilon)$
$P(u)$	\implies	$P([u])$
$P(u) \wedge P(v)$	\implies	$P(uv)$

„Induktion über die Erzeugung von u “

Lemma 4.11

Alle $u \in L_G(S)$ enthalten gleich viele [wie].

Zur induktiven Definition von $L_G(S)$ gehört ein Induktionsprinzip:

Um zu zeigen, dass für alle $u \in L_G(S)$ eine Eigenschaft $P(u)$ gilt, dh $\forall u \in L_G(S). P(u)$, zeige:

$$\begin{array}{l} P(\epsilon) \\ P(u) \implies P([u]) \\ P(u) \wedge P(v) \implies P(uv) \end{array}$$

„Induktion über die Erzeugung von u “

Lemma 4.11

Alle $u \in L_G(S)$ enthalten gleich viele [wie].

Beweis:

Mit Induktion über die Erzeugung von u :

- ϵ enthält 0 [und].
- Enthält u gleich viele [wie], so auch $[u]$.
- Enthalten u und v gleich viele [wie], so auch uv .



Definition 4.12 (Balancierte Klammerausdrücke)

Definition 4.12 (Balancierte Klammerausdrücke)

Präfix:

$$u \preceq w \quad :\Leftrightarrow \quad \exists v. uv = w$$

Anzahl der Vorkommen:

$$\#_a(w) := \text{Anzahl der Vorkommen von } a \text{ in } w$$

Seien:

$$A(w) := \#_{[}(w) \quad B(w) := \#_{]}(w)$$

Wir nennen $w \in \{[,]\}^*$ **balanciert** gdw

Definition 4.12 (Balancierte Klammerausdrücke)

Präfix:

$$u \preceq w \quad :\Leftrightarrow \quad \exists v. uv = w$$

Anzahl der Vorkommen:

$$\#_a(w) := \text{Anzahl der Vorkommen von } a \text{ in } w$$

Seien:

$$A(w) := \#_{[}(w) \quad B(w) := \#_{]}(w)$$

Wir nennen $w \in \{[,]\}^*$ **balanciert** gdw

$$(1) \quad A(w) = B(w) \text{ und}$$

Definition 4.12 (Balancierte Klammerausdrücke)

Präfix:

$$u \preceq w \quad :\Leftrightarrow \quad \exists v. uv = w$$

Anzahl der Vorkommen:

$$\#_a(w) := \text{Anzahl der Vorkommen von } a \text{ in } w$$

Seien:

$$A(w) := \#_{[}(w) \quad B(w) := \#_{]}(w)$$

Wir nennen $w \in \{[,]\}^*$ **balanciert** gdw

- (1) $A(w) = B(w)$ und
- (2) für alle Präfixe u von w gilt $A(u) \geq B(u)$

Definition 4.12 (Balancierte Klammerausdrücke)

Präfix:

$$u \preceq w \quad :\Leftrightarrow \quad \exists v. uv = w$$

Anzahl der Vorkommen:

$$\#_a(w) := \text{Anzahl der Vorkommen von } a \text{ in } w$$

Seien:

$$A(w) := \#_{[}(w) \quad B(w) := \#_{]}(w)$$

Wir nennen $w \in \{[,]\}^*$ **balanciert** gdw

(1) $A(w) = B(w)$ und

(2) für alle Präfixe u von w gilt $A(u) \geq B(u)$

- Balanciert: $[], [()], [] [], [[] [] [] []$

Definition 4.12 (Balancierte Klammerausdrücke)

Präfix:

$$u \preceq w \quad :\Leftrightarrow \quad \exists v. uv = w$$

Anzahl der Vorkommen:

$$\#_a(w) := \text{Anzahl der Vorkommen von } a \text{ in } w$$

Seien:

$$A(w) := \#_[(w) \quad B(w) := \#_](w)$$

Wir nennen $w \in \{[,]\}^*$ **balanciert** gdw

(1) $A(w) = B(w)$ und

(2) für alle Präfixe u von w gilt $A(u) \geq B(u)$

- Balanciert: [], [[]], [] [], [[] []] []
- Nicht balanciert:] [, []] [[]

Satz 4.13

Die Grammatik $S \rightarrow \epsilon \mid [S] \mid SS$ erzeugt genau die Menge der balancierten Wörter.

Satz 4.13

Die Grammatik $S \rightarrow \epsilon \mid [S] \mid SS$ erzeugt genau die Menge der balancierten Wörter.

Beweis:

$u \in L_G(S) \implies u$ balanciert:

Mit Ind. über die Erzeugung von u . (1) bewiesen, wir zeigen (2).

ϵ : $p \preceq \epsilon \implies p = \epsilon \implies A(p) = B(p)$

Satz 4.13

Die Grammatik $S \rightarrow \epsilon \mid [S] \mid SS$ erzeugt genau die Menge der balancierten Wörter.

Beweis:

$u \in L_G(S) \implies u$ balanciert:

Mit Ind. über die Erzeugung von u . (1) bewiesen, wir zeigen (2).

ϵ : $p \preceq \epsilon \implies p = \epsilon \implies A(p) = B(p)$

$[u]$: Sei $p \preceq [u]$

Fall $p = \epsilon$: $A(p) = B(p)$

Fall $p = [u]$: $A(p) = A(u) + 1 = B(u) + 1 = B(p)$

Fall $p = [q$ mit $q \preceq u$:

$A(p) = A(q) + 1 \geq B(q) + 1 > B(q) = B(p)$

Satz 4.13

Die Grammatik $S \rightarrow \epsilon \mid [S] \mid SS$ erzeugt genau die Menge der balancierten Wörter.

Beweis:

$u \in L_G(S) \implies u$ balanciert:

Mit Ind. über die Erzeugung von u . (1) bewiesen, wir zeigen (2).

ϵ : $p \preceq \epsilon \implies p = \epsilon \implies A(p) = B(p)$

$[u]$: Sei $p \preceq [u]$

Fall $p = \epsilon$: $A(p) = B(p)$

Fall $p = [u]$: $A(p) = A(u) + 1 = B(u) + 1 = B(p)$

Fall $p = [q$ mit $q \preceq u$:

$A(p) = A(q) + 1 \geq B(q) + 1 > B(q) = B(p)$

uv : Sei $p \preceq uv$.

Fall $p \preceq u$: $A(p) \geq B(p)$ mit IA für u

Fall $p = uq$ und $q \preceq v$:

$A(p) = A(u) + A(q) = B(u) + A(q) \geq B(u) + B(q) = B(uq) = B(p)$

Beweis (Forts.)

u balanciert $\implies u \in L_G(S)$:

Mit vollständiger Induktion über $n := |u|$. (dh $u = a_1 \dots a_n$)

IA: Jedes balancierte Wort $< n$ liegt in $L_G(S)$.

Zz: $u \in L_G(S)$.

Beweis (Forts.)

u balanciert $\implies u \in L_G(S)$:

Mit vollständiger Induktion über $n := |u|$. (dh $u = a_1 \dots a_n$)

IA: Jedes balancierte Wort $< n$ liegt in $L_G(S)$.

Zz: $u \in L_G(S)$.

Falls $n = 0$, so $u = \epsilon \in L_G(S)$.

Beweis (Forts.)

u balanciert $\implies u \in L_G(S)$:

Mit vollständiger Induktion über $n := |u|$. (dh $u = a_1 \dots a_n$)

IA: Jedes balancierte Wort $< n$ liegt in $L_G(S)$.

Zz: $u \in L_G(S)$.

Falls $n = 0$, so $u = \epsilon \in L_G(S)$.

Sei $n > 0$.

Betrachte Werteverlauf von $h(w) := A(w) - B(w)$:

$h(\epsilon), h(a_1), h(a_1 a_2), \dots, h(a_1 \dots a_n)$ (alle ≥ 0 !).

Insb. gilt $a_1 = [$ und $a_n =]$. Wir betrachten zwei Fälle:

Beweis (Forts.)

u balanciert $\implies u \in L_G(S)$:

Mit vollständiger Induktion über $n := |u|$. (dh $u = a_1 \dots a_n$)

IA: Jedes balancierte Wort $< n$ liegt in $L_G(S)$.

Zz: $u \in L_G(S)$.

Falls $n = 0$, so $u = \epsilon \in L_G(S)$.

Sei $n > 0$.

Betrachte Werteverlauf von $h(w) := A(w) - B(w)$:

$h(\epsilon), h(a_1), h(a_1 a_2), \dots, h(a_1 \dots a_n)$ (alle ≥ 0 !).

Insb. gilt $a_1 = [$ und $a_n =]$. Wir betrachten zwei Fälle:

- Es gibt nur die Nullstellen $h(\epsilon)$ und $h(a_1 \dots a_n)$.

Beweis (Forts.)

u balanciert $\implies u \in L_G(S)$:

Mit vollständiger Induktion über $n := |u|$. (dh $u = a_1 \dots a_n$)

IA: Jedes balancierte Wort $< n$ liegt in $L_G(S)$.

Zz: $u \in L_G(S)$.

Falls $n = 0$, so $u = \epsilon \in L_G(S)$.

Sei $n > 0$.

Betrachte Werteverlauf von $h(w) := A(w) - B(w)$:

$h(\epsilon), h(a_1), h(a_1 a_2), \dots, h(a_1 \dots a_n)$ (alle $\geq 0!$).

Insb. gilt $a_1 = [$ und $a_n =]$. Wir betrachten zwei Fälle:

- Es gibt nur die Nullstellen $h(\epsilon)$ und $h(a_1 \dots a_n)$.

Dann ist $v := a_2 \dots a_{n-1}$ balanciert:

$$(1) A(v) = A([v]) - 1 = B([v]) - 1 = B(v)$$

$$(2) p \preceq v: h([p]) = A([p]) - B([p]) > 0 \implies$$

$$A(p) = A([p]) - 1 > B([p]) - 1 = B(p) - 1 \implies$$

$$A(p) \geq B(p)$$

$$\implies v \in L_G(S) \text{ (nach IA)} \implies u = [v] \in L_G(S)$$

Beweis (Forts.):

- Es gibt noch eine Nullstelle $h(a_1 \dots a_k)$.

Beweis (Forts.):

- Es gibt noch eine Nullstelle $h(a_1 \dots a_k)$.

Sei $u_1 := a_1 \dots a_k$, $u_2 := a_{k+1} \dots a_n$

Dann sind u_1 und u_2 balanciert:

(1) $A(u_1) = B(u_1)$ da $h(u_1) = 0$;

$$A(u_2) = A(u) - A(u_1) = B(u) - B(u_1) = B(u_2)$$

(2) $p \preceq u_1 \implies p \preceq u \implies A(p) \geq B(p)$

$$\begin{aligned} p \preceq u_2: A(p) &= A(u_1 p) - A(u_1) \geq B(u_1 p) - B(u_1) = B(p) \\ \implies u_1, u_2 &\in L_G(S) \text{ (nach IA, da } |u_i| < n) \\ \implies u &= u_1 u_2 \in L_G(S) \end{aligned}$$



Moral:

- „ $w \in L_G(S) \implies P(w)$ “ beweist man immer schematisch mit Induktion über die Erzeugung von w .
- „ $P(w) \implies w \in L_G(S)$ “ beweist man oft mit Induktion über $|w|$. Erfordert meist Kreativität.

Wir übertragen die Idee der induktiven Erzeugung auf beliebige CFGs $G = (V, \Sigma, P, S)$. Sei $V = \{A_1, \dots, A_k\}$. Damit ist jede Produktion von der Form

$$A_i \rightarrow w_0 A_{i_1} w_1 \dots w_{n-1} A_{i_n} w_n$$

Wir übertragen die Idee der induktiven Erzeugung auf beliebige CFGs $G = (V, \Sigma, P, S)$. Sei $V = \{A_1, \dots, A_k\}$. Damit ist jede Produktion von der Form

$$A_i \rightarrow w_0 A_{i_1} w_1 \dots w_{n-1} A_{i_n} w_n$$

Man kann G als eine (simultane) induktive Definition von Sprachen $L_G(A_i)$ für all $A_i \in V$ sehen. Jede Produktion korrespondiert zu einer Erzeugungsregel

$$u_1 \in L_G(A_{i_1}) \wedge \dots \wedge u_n \in L_G(A_{i_n}) \implies w_0 u_1 w_1 \dots u_n w_n \in L_G(A_i)$$

Wir übertragen die Idee der induktiven Erzeugung auf beliebige CFGs $G = (V, \Sigma, P, S)$. Sei $V = \{A_1, \dots, A_k\}$. Damit ist jede Produktion von der Form

$$A_i \rightarrow w_0 A_{i_1} w_1 \dots w_{n-1} A_{i_n} w_n$$

Man kann G als eine (simultane) induktive Definition von Sprachen $L_G(A_i)$ für all $A_i \in V$ sehen. Jede Produktion korrespondiert zu einer Erzeugungsregel

$$u_1 \in L_G(A_{i_1}) \wedge \dots \wedge u_n \in L_G(A_{i_n}) \implies w_0 u_1 w_1 \dots u_n w_n \in L_G(A_i)$$

Aussagen der Form

$$(u \in L_G(A_1) \implies P_1(u)) \wedge \dots \wedge (u \in L_G(A_k) \implies P_k(u))$$

werden durch **simultane Induktion über die Erzeugung von u** bewiesen, indem man für jede Produktion zeigt:

$$P_{i_1}(u_1) \wedge \dots \wedge P_{i_n}(u_n) \implies P_i(w_0 u_1 w_1 \dots w_{n-1} u_n w_n)$$

Beispiel 4.14

Grammatik:

$$A \rightarrow \epsilon \mid aB \quad B \rightarrow Aa$$

Induktive Definition:

- $\epsilon \in L_G(A)$
- $w \in L_G(B) \implies aw \in L_G(A)$
- $w \in L_G(A) \implies wa \in L_G(B)$

Beispiel 4.14

Grammatik:

$$A \rightarrow \epsilon \mid aB \quad B \rightarrow Aa$$

Induktive Definition:

- $\epsilon \in L_G(A)$
- $w \in L_G(B) \implies aw \in L_G(A)$
- $w \in L_G(A) \implies wa \in L_G(B)$

Beim induktiven Beweis von

$$\begin{aligned} & (w \in L_G(A) \implies \#_a(w) \text{ ist gerade}) \wedge \\ & (w \in L_G(B) \implies \#_a(w) \text{ ist ungerade}) \end{aligned}$$

muss man zeigen

- $\#_a(\epsilon)$ ist gerade
- $\#_a(w)$ ist ungerade $\implies \#_a(aw)$ ist gerade
- $\#_a(w)$ ist gerade $\implies \#_a(wa)$ ist ungerade

Definition 4.15 (Syntaxbaum)

Ein **Syntaxbaum** für eine Ableitung mit einer Grammatik $G = (V, \Sigma, P, S)$ ist ein Baum, so dass

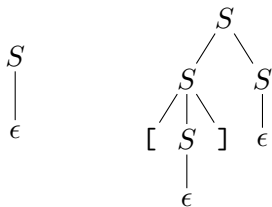
- jedes Blatt mit einem Zeichen aus $\Sigma \cup \{\epsilon\}$ beschriftet ist,
- jeder innere Knoten mit einem $A \in V$ beschriftet ist, und falls die Nachfolger (von links nach rechts) mit $X_1, \dots, X_n \in V \cup \Sigma \cup \{\epsilon\}$ beschriftet sind, dann ist $A \rightarrow X_1 \dots X_n$ eine Produktion in P ,
- ein Blatt ϵ der einzige Nachfolger seines Vorgängers ist.

Definition 4.15 (Syntaxbaum)

Ein **Syntaxbaum** für eine Ableitung mit einer Grammatik $G = (V, \Sigma, P, S)$ ist ein Baum, so dass

- jedes Blatt mit einem Zeichen aus $\Sigma \cup \{\epsilon\}$ beschriftet ist,
- jeder innere Knoten mit einem $A \in V$ beschriftet ist, und falls die Nachfolger (von links nach rechts) mit $X_1, \dots, X_n \in V \cup \Sigma \cup \{\epsilon\}$ beschriftet sind, dann ist $A \rightarrow X_1 \dots X_n$ eine Produktion in P ,
- ein Blatt ϵ der einzige Nachfolger seines Vorgängers ist.

Beispiel 4.16 (Syntaxbäume für $S \rightarrow \epsilon \mid [S] \mid SS$)

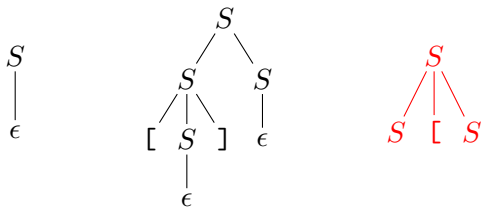


Definition 4.15 (Syntaxbaum)

Ein **Syntaxbaum** für eine Ableitung mit einer Grammatik $G = (V, \Sigma, P, S)$ ist ein Baum, so dass

- jedes Blatt mit einem Zeichen aus $\Sigma \cup \{\epsilon\}$ beschriftet ist,
- jeder innere Knoten mit einem $A \in V$ beschriftet ist, und falls die Nachfolger (von links nach rechts) mit $X_1, \dots, X_n \in V \cup \Sigma \cup \{\epsilon\}$ beschriftet sind, dann ist $A \rightarrow X_1 \dots X_n$ eine Produktion in P ,
- ein Blatt ϵ der einzige Nachfolger seines Vorgängers ist.

Beispiel 4.16 (Syntaxbäume für $S \rightarrow \epsilon \mid [S] \mid SS$)



Satz 4.17

Für eine CFG und ein $w \in \Sigma^*$ sind folgende Bedingungen äquivalent:

- 1 $A \rightarrow_G^* w$
- 2 $w \in L_G(A)$ (gemäß der induktiven Definition)
- 3 Es gibt einen Syntaxbaum mit Wurzel A , dessen **Rand** (Blätter von links nach rechts gelesen) das Wort w ist.

Beweis:

Skizze (Details: [HMU]) $1 \Rightarrow 2$: Sei $A \rightarrow_G^n w$. Wir konstruieren dazu einen Beweis für $w \in L_G(A)$ mit Induktion über n . Die Ableitung muss von folgender Form sein:

$$A \rightarrow_G w_0 A_1 w_1 \dots A_m w_m \rightarrow_G^{n-1} w$$

Nach dem Dekompositionslemma muss es u_i und $n_i \leq n - 1$ geben mit $A_i \rightarrow_G^{n_i} u_i$ und $w = w_0 u_1 w_1 \dots u_m w_m$. Nach IA (da $n_i < n$) gilt $u_i \in L_G(A_i)$ und daher (mit einem weiteren Erzeugungsschritt) auch $w \in L_G(A)$.

2 \Rightarrow 3: Parallel zur induktiven Erzeugung von $w \in L_G(A)$ kann man einen Syntaxbaum mit Rand w erzeugen. Formal: Induktion über die Erzeugung von w .

3 \Rightarrow 1: Jeder Baum lässt sich in eine Ableitung, sogar eine Linksableitung(!) umformen. Induktion über die Höhe des Baums: Transformiere die Unterbäume t_i mit Beschriftung A_i in Ableitungen $A_i \rightarrow_G^* u_i$ und füge diese mit dem Dekompositionslemma zu einer grossen Ableitung $A \rightarrow_G w_0 A_1 w_1 \dots A_m w_m \rightarrow_G^* w_0 u_1 w_1 \dots u_m w_m$ zusammen.

Definition 4.18

- Eine CFG G heißt **mehrdeutig** gdw es ein $w \in L(G)$ gibt, das zwei verschiedene Syntaxbäume hat, also zwei verschiedene Syntaxbäume mit Wurzel S und Rand w .
- Eine CFL L heißt **inhärent mehrdeutig** gdw jede CFG G mit $L(G) = L$ mehrdeutig ist.

Definition 4.18

- Eine CFG G heißt **mehrdeutig** gdw es ein $w \in L(G)$ gibt, das zwei verschiedene Syntaxbäume hat, also zwei verschiedene Syntaxbäume mit Wurzel S und Rand w .
- Eine CFL L heißt **inhärent mehrdeutig** gdw jede CFG G mit $L(G) = L$ mehrdeutig ist.

Beispiel 4.19

Die Grammatik $E \rightarrow E + E \mid E * E \mid (E) \mid a$ ist mehrdeutig — betrachte $a + a * a$. Die erzeugte Sprache ist aber nicht inhärent mehrdeutig (siehe Beispiel Arithmetische Ausdrücke).

Definition 4.18

- Eine CFG G heißt **mehrdeutig** gdw es ein $w \in L(G)$ gibt, das zwei verschiedene Syntaxbäume hat, also zwei verschiedene Syntaxbäume mit Wurzel S und Rand w .
- Eine CFL L heißt **inhärent mehrdeutig** gdw jede CFG G mit $L(G) = L$ mehrdeutig ist.

Beispiel 4.19

Die Grammatik $E \rightarrow E + E \mid E * E \mid (E) \mid a$ ist mehrdeutig — betrachte $a + a * a$. Die erzeugte Sprache ist aber nicht inhärent mehrdeutig (siehe Beispiel Arithmetische Ausdrücke).

Satz 4.20

Die Sprache $\{a^i b^j c^k \mid i = j \vee j = k\}$ ist inhärent mehrdeutig.

4.3 Die Chomsky-Normalform

Definition 4.21

Eine kontextfreie Grammatik G ist in **Chomsky-Normalform** gdw alle Produktionen eine der Formen

$$A \rightarrow a \quad \text{oder} \quad A \rightarrow BC$$

haben.

4.3 Die Chomsky-Normalform

Definition 4.21

Eine kontextfreie Grammatik G ist in **Chomsky-Normalform** gdw alle Produktionen eine der Formen

$$A \rightarrow a \quad \text{oder} \quad A \rightarrow BC$$

haben.

Wir konstruieren und beweisen nun schrittweise:

Satz 4.22

Zu jeder CFG G kann man eine CFG G' in Chomsky-Normalform konstruieren mit $L(G') = L(G) \setminus \{\epsilon\}$.

4.3 Die Chomsky-Normalform

Definition 4.21

Eine kontextfreie Grammatik G ist in **Chomsky-Normalform** gdw alle Produktionen eine der Formen

$$A \rightarrow a \quad \text{oder} \quad A \rightarrow BC$$

haben.

Wir konstruieren und beweisen nun schrittweise:

Satz 4.22

Zu jeder CFG G kann man eine CFG G' in Chomsky-Normalform konstruieren mit $L(G') = L(G) \setminus \{\epsilon\}$.

Wer auf $\epsilon \in L(G')$ nicht verzichten möchte:

Füge am Ende $S' \rightarrow S, S' \rightarrow \epsilon$ hinzu und setze S' als Startsymbol.

Wir nennen $A \rightarrow \epsilon$ eine ϵ -Produktion.

Wir nennen $A \rightarrow \epsilon$ eine ϵ -Produktion.

Lemma 4.23

Zu jeder CFG $G = (V, \Sigma, P, S)$ kann man eine CFG G' konstruieren, die keine ϵ -Produktionen enthält, so dass gilt

$$L(G') = L(G) \setminus \{\epsilon\}$$

Wir nennen $A \rightarrow \epsilon$ eine ϵ -Produktion.

Lemma 4.23

Zu jeder CFG $G = (V, \Sigma, P, S)$ kann man eine CFG G' konstruieren, die keine ϵ -Produktionen enthält, so dass gilt $L(G') = L(G) \setminus \{\epsilon\}$

Beweis:

Wir erweitern P induktiv zu eine Obermenge \hat{P} :

- 1 Jede Produktion aus P ist in \hat{P}
- 2 Sind $B \rightarrow \epsilon$ und $A \rightarrow \alpha B \beta$ in \hat{P} , so füge auch $A \rightarrow \alpha \beta$ hinzu.

Offensichtlich gilt $L(\hat{G}) = L(G)$: Jede neue Produktion kann von 2 alten simuliert werden.

Wir nennen $A \rightarrow \epsilon$ eine ϵ -Produktion.

Lemma 4.23

Zu jeder CFG $G = (V, \Sigma, P, S)$ kann man eine CFG G' konstruieren, die keine ϵ -Produktionen enthält, so dass gilt $L(G') = L(G) \setminus \{\epsilon\}$

Beweis:

Wir erweitern P induktiv zu eine Obermenge \hat{P} :

- 1 Jede Produktion aus P ist in \hat{P}
- 2 Sind $B \rightarrow \epsilon$ und $A \rightarrow \alpha B \beta$ in \hat{P} , so füge auch $A \rightarrow \alpha \beta$ hinzu.

Offensichtlich gilt $L(\hat{G}) = L(G)$: Jede neue Produktion kann von 2 alten simuliert werden.

Wir definieren G' als \hat{G} ohne die ϵ -Produktionen.

Denn diese sind nun überflüssig:

Beweis (Forts.):

Sei $S \xrightarrow{\hat{G}}^* w$, $w \neq \epsilon$, eine Ableitung minimaler Länge.

Käme darin $B \rightarrow \epsilon$ vor, also

$$S \xrightarrow{\hat{G}}^* \gamma B \delta \xrightarrow{\hat{G}} \gamma \delta \xrightarrow{\hat{G}}^* w$$

so wäre γ oder δ nichtleer, dh B muss durch eine Produktion $A \rightarrow \alpha B \beta$ eingeführt worden sein:

$$S \xrightarrow{\hat{G}}^k \alpha' A \beta' \xrightarrow{\hat{G}} \alpha' \alpha B \beta \beta' \xrightarrow{\hat{G}}^m \gamma B \delta \xrightarrow{\hat{G}} \gamma \delta \xrightarrow{\hat{G}}^n w$$

Beweis (Forts.):

Sei $S \xrightarrow{\hat{G}}^* w$, $w \neq \epsilon$, eine Ableitung minimaler Länge.

Käme darin $B \rightarrow \epsilon$ vor, also

$$S \xrightarrow{\hat{G}}^* \gamma B \delta \xrightarrow{\hat{G}} \gamma \delta \xrightarrow{\hat{G}}^* w$$

so wäre γ oder δ nichtleer, dh B muss durch eine Produktion $A \rightarrow \alpha B \beta$ eingeführt worden sein:

$$S \xrightarrow{\hat{G}}^k \alpha' A \beta' \xrightarrow{\hat{G}} \alpha' \alpha B \beta \beta' \xrightarrow{\hat{G}}^m \gamma B \delta \xrightarrow{\hat{G}} \gamma \delta \xrightarrow{\hat{G}}^n w$$

Dann wäre aber folgende Ableitung mit $(A \rightarrow \alpha \beta) \in \hat{P}$ kürzer:

$$S \xrightarrow{\hat{G}}^k \alpha' A \beta' \xrightarrow{\hat{G}} \alpha' \alpha \beta \beta' \xrightarrow{\hat{G}}^m \gamma \delta \xrightarrow{\hat{G}}^n w$$

Beweis (Forts.):

Sei $S \xrightarrow{\hat{G}}^* w$, $w \neq \epsilon$, eine Ableitung minimaler Länge.
Käme darin $B \rightarrow \epsilon$ vor, also

$$S \xrightarrow{\hat{G}}^* \gamma B \delta \xrightarrow{\hat{G}} \gamma \delta \xrightarrow{\hat{G}}^* w$$

so wäre γ oder δ nichtleer, dh B muss durch eine Produktion
 $A \rightarrow \alpha B \beta$ eingeführt worden sein:

$$S \xrightarrow{\hat{G}}^k \alpha' A \beta' \xrightarrow{\hat{G}} \alpha' \alpha B \beta \beta' \xrightarrow{\hat{G}}^m \gamma B \delta \xrightarrow{\hat{G}} \gamma \delta \xrightarrow{\hat{G}}^n w$$

Dann wäre aber folgende Ableitung mit $(A \rightarrow \alpha \beta) \in \hat{P}$ kürzer:

$$S \xrightarrow{\hat{G}}^k \alpha' A \beta' \xrightarrow{\hat{G}} \alpha' \alpha \beta \beta' \xrightarrow{\hat{G}}^m \gamma \delta \xrightarrow{\hat{G}}^n w$$

Also kommen in Ableitungen minimaler Länge keine
 ϵ -Produktionen vor. Letztere sind also überflüssig. □

Beispiel 4.24

$$S \rightarrow AB, \quad A \rightarrow aAA \mid B, \quad B \rightarrow bBS \mid \epsilon$$

Wir nennen $A \rightarrow B$ eine Kettenproduktion.

Wir nennen $A \rightarrow B$ eine Kettenproduktion.

Lemma 4.25

Zu jeder CFG $G = (V, \Sigma, P, S)$ kann man eine CFG G' konstruieren, die keine Kettenproduktionen enthält, so dass gilt $L(G') = L(G)$.

Wir nennen $A \rightarrow B$ eine Kettenproduktion.

Lemma 4.25

Zu jeder CFG $G = (V, \Sigma, P, S)$ kann man eine CFG G' konstruieren, die keine Kettenproduktionen enthält, so dass gilt $L(G') = L(G)$.

Beweis:

Wir erweitern P induktiv zu eine Obermenge \hat{P} :

- 1 Jede Produktion aus P ist in \hat{P}
- 2 Sind $A \rightarrow B$ und $B \rightarrow \alpha$ in \hat{P} mit $\alpha \neq A$,
so füge auch $A \rightarrow \alpha$ hinzu.

Das Ergebnis G' ist \hat{G} ohne die (nun überflüssigen) Kettenproduktionen.

Rest des Beweises analog zur Elimination von ϵ -Produktionen. □

Beispiel 4.26

$$A \rightarrow a \mid B, \quad B \rightarrow b \mid C \mid aBB, \quad C \rightarrow A \mid AB$$

Konstruktion einer Chomsky-Normalform

Eingabe: Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$

Konstruktion einer Chomsky-Normalform

Eingabe: Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$

- 1 Füge für jedes $a \in \Sigma$, das in einer rechten Seite der Länge ≥ 2 vorkommt, ein neues Nichtterminal A_a zu V hinzu, ersetze a in allen rechten Seiten der Länge ≥ 2 durch A_a , und füge $A_a \rightarrow a$ zu P hinzu.
- 2 Ersetze jede Produktion der Form

$$A \rightarrow B_1 B_2 \cdots B_k \quad (k \geq 3)$$

durch

$$A \rightarrow B_1 C_2, \quad C_2 \rightarrow B_2 C_3, \quad \dots, \quad C_{k-1} \rightarrow B_{k-1} B_k$$

wobei C_2, \dots, C_{k-1} neue Nichtterminale sind.

- 3 Eliminiere alle ϵ -Produktionen.
- 4 Eliminiere alle Kettenproduktionen.

Aufgabe:

- Zeige: Mit diesem Algorithmus ist das Ergebnis höchstens quadratisch größer als G .
- Zeige: die Reihenfolge der Schritte ist wichtig!
 - Finde eine andere Reihenfolge, die inkorrekt ist: Das resultierende Verfahren ergibt nicht immer eine Grammatik in CNF.
 - Finde eine andere Reihenfolge, die zwar zu einem korrekten Verfahren führt, aber eine Grammatik in CNF produzieren kann, die exponentiell größer als G ist.

Definition 4.27

Eine CFG ist in **Greibach-Normalform** (benannt nach Sheila Greibach, UCLA), falls jede Produktion von der Form

$$A \rightarrow aA_1 \dots A_n$$

ist.

Definition 4.27

Eine CFG ist in **Greibach-Normalform** (benannt nach Sheila Greibach, UCLA), falls jede Produktion von der Form

$$A \rightarrow aA_1 \dots A_n$$

ist.

Satz 4.28

Zu jeder CFG G gibt es eine CFG G' in Greibach-Normalform mit $L(G') = L(G) \setminus \{\epsilon\}$.

4.4 Das Pumping-Lemma für kontextfreie Sprachen

Zur Erinnerung: Das Pumping-Lemma für reguläre Sprachen: Für jede reguläre Sprache L gibt es ein $n \in \mathbb{N}$, so dass sich jedes Wort $z \in L$ mit $|z| \geq n$ zerlegen lässt in $z = uvw$ mit $|uv| \leq n$, $v \neq \epsilon$ und $uv^*w \subseteq L$.

4.4 Das Pumping-Lemma für kontextfreie Sprachen

Zur Erinnerung: Das Pumping-Lemma für reguläre Sprachen: Für jede reguläre Sprache L gibt es ein $n \in \mathbb{N}$, so dass sich jedes Wort $z \in L$ mit $|z| \geq n$ zerlegen lässt in $z = uvw$ mit $|uv| \leq n$, $v \neq \epsilon$ und $uv^*w \subseteq L$.

Zum Beweis haben wir $n = |Q|$ gewählt, wobei Q die Zustandsmenge eines L erkennenden DFA war. Das Argument war dann, dass beim Erkennen von z (mindestens) ein Zustand zweimal besucht werden muss und damit der dazwischen liegende Weg im Automaten beliebig oft wiederholt werden kann.

4.4 Das Pumping-Lemma für kontextfreie Sprachen

Zur Erinnerung: Das Pumping-Lemma für reguläre Sprachen: Für jede reguläre Sprache L gibt es ein $n \in \mathbb{N}$, so dass sich jedes Wort $z \in L$ mit $|z| \geq n$ zerlegen lässt in $z = uvw$ mit $|uv| \leq n$, $v \neq \epsilon$ und $uv^*w \subseteq L$.

Zum Beweis haben wir $n = |Q|$ gewählt, wobei Q die Zustandsmenge eines L erkennenden DFA war. Das Argument war dann, dass beim Erkennen von z (mindestens) ein Zustand zweimal besucht werden muss und damit der dazwischen liegende Weg im Automaten beliebig oft wiederholt werden kann.

Ein ähnliches Argument kann auch auf kontextfreie Grammatiken angewendet werden.

Satz 4.29 (Pumping-Lemma)

Für jede kontextfreie Sprache L gibt es ein $n \geq 1$, so dass sich jedes Wort $z \in L$ mit $|z| \geq n$ zerlegen lässt in

$$z = uvwxy,$$

mit

Satz 4.29 (Pumping-Lemma)

Für jede kontextfreie Sprache L gibt es ein $n \geq 1$, so dass sich jedes Wort $z \in L$ mit $|z| \geq n$ zerlegen lässt in

$$z = uvwxy,$$

mit

- $vx \neq \epsilon$,

Satz 4.29 (Pumping-Lemma)

Für jede kontextfreie Sprache L gibt es ein $n \geq 1$, so dass sich jedes Wort $z \in L$ mit $|z| \geq n$ zerlegen lässt in

$$z = uvwxy,$$

mit

- $vx \neq \epsilon$,
- $|vwx| \leq n$, und

Satz 4.29 (Pumping-Lemma)

Für jede kontextfreie Sprache L gibt es ein $n \geq 1$, so dass sich jedes Wort $z \in L$ mit $|z| \geq n$ zerlegen lässt in

$$z = uvwxy,$$

mit

- $vx \neq \epsilon$,
- $|vwx| \leq n$, und
- $\forall i \in \mathbb{N}. uv^iwx^iy \in L$.

Satz 4.29 (Pumping-Lemma)

Für jede kontextfreie Sprache L gibt es ein $n \geq 1$, so dass sich jedes Wort $z \in L$ mit $|z| \geq n$ zerlegen lässt in

$$z = uvwxy,$$

mit

- $vx \neq \epsilon$,
- $|vwx| \leq n$, und
- $\forall i \in \mathbb{N}. uv^iwx^iy \in L$.

Beweis:

Sei $G = (V, \Sigma, P, S)$ eine CFG in Chomsky-Normalform mit $L(G) = L \setminus \{\epsilon\}$. Wähle $n = 2^{|V|}$. Sei $z \in L(G)$ mit $|z| \geq n$.

Beweis:

Sei $G = (V, \Sigma, P, S)$ eine CFG in Chomsky-Normalform mit $L(G) = L \setminus \{\epsilon\}$. Wähle $n = 2^{|V|}$. Sei $z \in L(G)$ mit $|z| \geq n$.

Jeder Binärbaum mit $\geq 2^k$ Blättern hat die Höhe $\geq k$

Beweis:

Sei $G = (V, \Sigma, P, S)$ eine CFG in Chomsky-Normalform mit $L(G) = L \setminus \{\epsilon\}$. Wähle $n = 2^{|V|}$. Sei $z \in L(G)$ mit $|z| \geq n$.

Jeder Binärbaum mit $\geq 2^k$ Blättern hat die Höhe $\geq k$

Dann hat der Syntaxbaum für z (ohne die letzte Stufe für die Terminale) mindestens einen Pfad der Länge $\geq |V|$, da er wegen der Chomsky-Normalform ein Binärbaum ist: Wir betrachten einen solchen Pfad maximaler Länge.

Beweis:

Sei $G = (V, \Sigma, P, S)$ eine CFG in Chomsky-Normalform mit $L(G) = L \setminus \{\epsilon\}$. Wähle $n = 2^{|V|}$. Sei $z \in L(G)$ mit $|z| \geq n$.

Jeder Binärbaum mit $\geq 2^k$ Blättern hat die Höhe $\geq k$

Dann hat der Syntaxbaum für z (ohne die letzte Stufe für die Terminale) mindestens einen Pfad der Länge $\geq |V|$, da er wegen der Chomsky-Normalform ein Binärbaum ist: Wir betrachten einen solchen Pfad maximaler Länge.

Auf diesem Pfad der Länge $\geq |V|$ kommen $\geq |V| + 1$ Nichtterminale vor, also mindestens eins doppelt. Nennen wir es A .

Beweis:

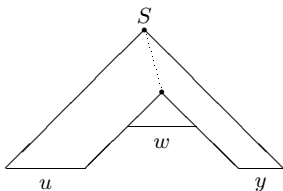
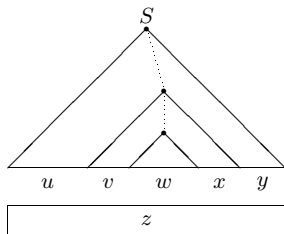
Sei $G = (V, \Sigma, P, S)$ eine CFG in Chomsky-Normalform mit $L(G) = L \setminus \{\epsilon\}$. Wähle $n = 2^{|V|}$. Sei $z \in L(G)$ mit $|z| \geq n$.

Jeder Binärbaum mit $\geq 2^k$ Blättern hat die Höhe $\geq k$

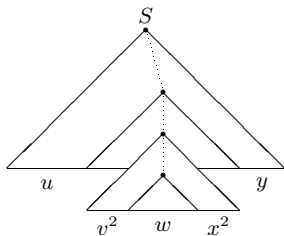
Dann hat der Syntaxbaum für z (ohne die letzte Stufe für die Terminale) mindestens einen Pfad der Länge $\geq |V|$, da er wegen der Chomsky-Normalform ein Binärbaum ist: Wir betrachten einen solchen Pfad maximaler Länge.

Auf diesem Pfad der Länge $\geq |V|$ kommen $\geq |V| + 1$ Nichtterminale vor, also mindestens eins doppelt. Nennen wir es A .

Die zwischen zwei Vorkommen von A liegende Teibleitung kann nun beliebig oft wiederholt werden.



Dieser Ableitungsbaum zeigt
 $uw y \in L$



Dieser Ableitungsbaum zeigt
 $uv^2wx^2y \in L$

Beweis (Forts.):

Die Bedingung $vx \neq \epsilon$ folgt, da das obere der beiden A 's mit $A \rightarrow BC$ abgeleitet werden muss.

Beweis (Forts.):

Die Bedingung $vx \neq \epsilon$ folgt, da das obere der beiden A 's mit $A \rightarrow BC$ abgeleitet werden muss.

Um $|vwx| \leq n$ zu erreichen, darf das obere A maximal $|V| + 1$ Schritte von jedem Blatt entfernt sein. Da der ausgewählte Pfad maximale Länge hat, genügt es, dass das obere A vom Blatt dieses Pfades $|V| + 1$ Schritte entfernt ist. Da es nur $|V|$ Nichtterminale gibt, muss ein solches doppeltes A existieren. □

Beispiel 4.30

Wir zeigen, dass die Sprache

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

nicht kontextfrei ist.

Beispiel 4.30

Wir zeigen, dass die Sprache

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

nicht kontextfrei ist.

Wäre L kontextfrei, so gäbe es eine dazugehörige Pumping-Lemma Zahl n und wir könnten jedes $z \in L$ mit $|z| \geq n$, insb. $z = a^n b^n c^n$, zerlegen und aufpumpen, ohne aus der Sprache herauszufallen.

Beispiel 4.30

Wir zeigen, dass die Sprache

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

nicht kontextfrei ist.

Wäre L kontextfrei, so gäbe es eine dazugehörige Pumping-Lemma Zahl n und wir könnten jedes $z \in L$ mit $|z| \geq n$, insb. $z = a^n b^n c^n$, zerlegen und aufpumpen, ohne aus der Sprache herauszufallen.

Eine Zerlegung $z = uvwxy$ mit $|vwx| \leq n$ bedeutet aber, dass vwx nicht a 's und c 's enthalten kann. OE nehmen wir an, vwx enthält nur a 's und b 's. Da $vx \neq \epsilon$, muss vx mindestens ein a oder ein b enthalten.

Beispiel 4.30

Wir zeigen, dass die Sprache

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

nicht kontextfrei ist.

Wäre L kontextfrei, so gäbe es eine dazugehörige Pumping-Lemma Zahl n und wir könnten jedes $z \in L$ mit $|z| \geq n$, insb. $z = a^n b^n c^n$, zerlegen und aufpumpen, ohne aus der Sprache herauszufallen.

Eine Zerlegung $z = uvwxy$ mit $|vwx| \leq n$ bedeutet aber, dass vwx nicht a 's und c 's enthalten kann. OE nehmen wir an, vwx enthält nur a 's und b 's. Da $vx \neq \epsilon$, muss vx mindestens ein a oder ein b enthalten.

Damit gilt aber $\#_a(uv^2wx^2y) > \#_c(uv^2wx^2y)$ oder $\#_b(uv^2wx^2y) > \#_c(uv^2wx^2y)$. Also $uv^2wx^2y \notin L$. ⚡

Beispiel 4.31

Mit dem Pumping-Lemma kann man zeigen, dass die Sprache $\{ww \mid w \in \{a, b\}^*\}$ nicht kontextfrei ist.

Beispiel 4.31

Mit dem Pumping-Lemma kann man zeigen, dass die Sprache $\{ww \mid w \in \{a, b\}^*\}$ nicht kontextfrei ist.

Dies zeigt, dass Kontextbedingungen in Programmiersprachen, etwa „*Deklaration vor Benutzung*“, nicht durch kontextfreie Grammatiken ausgedrückt werden können.

4.5 Abschlusseigenschaften

Satz 4.32

Seien kontextfreie Grammatiken $G_1 = (V_1, \Sigma_1, P_1, S_1)$ und $G_2 = (V_2, \Sigma_2, P_2, S_2)$ gegeben. Dann kann man in linearer Zeit kontextfreie Grammatiken für

- 1 $L(G_1) \cup L(G_2)$
- 2 $L(G_1)L(G_2)$
- 3 $(L(G_1))^*$
- 4 $(L(G_1))^R$

konstruieren.

Die Klasse der kontextfreien Sprachen ist also unter *Vereinigung*, *Konkatenation*, *Stern* und *Spiegelung* abgeschlossen.

Beweis:

OE nehmen wir an, dass $V_1 \cap V_2 = \emptyset$.

- 1 $V = V_1 \cup V_2 \cup \{S\}$; S neu
 $P = P_1 \cup P_2 \cup \{S \rightarrow S_1 \mid S_2\}$
- 2 $V = V_1 \cup V_2 \cup \{S\}$; S neu
 $P = P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\}$
- 3 $V = V_1 \cup \{S\}$; S neu
 $P = P_1 \cup \{S \rightarrow \epsilon \mid S_1 S\}$
- 4 $P = \{A \rightarrow \alpha^R \mid (A \rightarrow \alpha) \in P_1\}$



Beweis:

OE nehmen wir an, dass $V_1 \cap V_2 = \emptyset$.

- 1 $V = V_1 \cup V_2 \cup \{S\}$; S neu
 $P = P_1 \cup P_2 \cup \{S \rightarrow S_1 \mid S_2\}$
- 2 $V = V_1 \cup V_2 \cup \{S\}$; S neu
 $P = P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\}$
- 3 $V = V_1 \cup \{S\}$; S neu
 $P = P_1 \cup \{S \rightarrow \epsilon \mid S_1 S\}$
- 4 $P = \{A \rightarrow \alpha^R \mid (A \rightarrow \alpha) \in P_1\}$



Es folgt: Verallgemeinerte kontextfreie Grammatiken mit Produktionen der Gestalt $X \rightarrow r$, wobei r ein regulärer Ausdruck über $(V \cup T)$ ist, erzeugen kontextfreie Sprachen.

Satz 4.33

*Die Menge der kontextfreien Sprachen ist **nicht** abgeschlossen unter Durchschnitt oder Komplement.*

Satz 4.33

Die Menge der kontextfreien Sprachen ist **nicht** abgeschlossen unter Durchschnitt oder Komplement.

Beweis:

$L_1 := \{a^i b^i c^j \mid i, j \in \mathbb{N}\}$ ist kontextfrei

$L_2 := \{a^i b^j c^j \mid i, j \in \mathbb{N}\}$ ist kontextfrei

$L_1 \cap L_2 = \{a^i b^i c^i \mid i \in \mathbb{N}\}$ ist **nicht** kontextfrei

Satz 4.33

Die Menge der kontextfreien Sprachen ist **nicht** abgeschlossen unter Durchschnitt oder Komplement.

Beweis:

$L_1 := \{a^i b^i c^j \mid i, j \in \mathbb{N}\}$ ist kontextfrei

$L_2 := \{a^i b^j c^j \mid i, j \in \mathbb{N}\}$ ist kontextfrei

$L_1 \cap L_2 = \{a^i b^i c^i \mid i \in \mathbb{N}\}$ ist **nicht** kontextfrei

Wegen **de Morgan** (1806–1871) können die CFLs daher auch nicht unter Komplement abgeschlossen sein. □

4.6 Algorithmen für kontextfreie Grammatiken

Wie üblich: $G = (V, \Sigma, P, S)$ ist eine CFG.

4.6 Algorithmen für kontextfreie Grammatiken

Wie üblich: $G = (V, \Sigma, P, S)$ ist eine CFG.

Ein Symbol $X \in V \cup \Sigma$ ist

nützlich gdw es eine Ableitung $S \rightarrow_G^* w \in \Sigma^*$ gibt, in der X vorkommt.

erzeugend gdw es eine Ableitung $X \rightarrow_G^* w \in \Sigma^*$ gibt.

erreichbar gdw es eine Ableitung $S \rightarrow_G^* \alpha X \beta$ gibt.

4.6 Algorithmen für kontextfreie Grammatiken

Wie üblich: $G = (V, \Sigma, P, S)$ ist eine CFG.

Ein Symbol $X \in V \cup \Sigma$ ist

nützlich gdw es eine Ableitung $S \rightarrow_G^* w \in \Sigma^*$ gibt, in der X vorkommt.

erzeugend gdw es eine Ableitung $X \rightarrow_G^* w \in \Sigma^*$ gibt.

erreichbar gdw es eine Ableitung $S \rightarrow_G^* \alpha X \beta$ gibt.

Fakt 4.34

Nützliche Symbole sind erzeugend und erreichbar.

Aber nicht notwendigerweise umgekehrt:

$$S \rightarrow AB \mid a, \quad A \rightarrow b$$

4.6 Algorithmen für kontextfreie Grammatiken

Wie üblich: $G = (V, \Sigma, P, S)$ ist eine CFG.

Ein Symbol $X \in V \cup \Sigma$ ist

nützlich gdw es eine Ableitung $S \rightarrow_G^* w \in \Sigma^*$ gibt, in der X vorkommt.

erzeugend gdw es eine Ableitung $X \rightarrow_G^* w \in \Sigma^*$ gibt.

erreichbar gdw es eine Ableitung $S \rightarrow_G^* \alpha X \beta$ gibt.

Fakt 4.34

Nützliche Symbole sind erzeugend und erreichbar.

Aber nicht notwendigerweise umgekehrt:

$$S \rightarrow AB \mid a, \quad A \rightarrow b$$

Ziel: Elimination der unnützen Symbole und der Produktionen, in denen sie vorkommen.

Beispiel 4.35

$$S \rightarrow AB \mid a, \quad A \rightarrow b$$

Beispiel 4.35

$$S \rightarrow AB \mid a, \quad A \rightarrow b$$

- 1 Elimination nicht erzeugender Symbole:

$$S \rightarrow a, \quad A \rightarrow b$$

- 2 Elimination unerreichbarer Symbole:

$$S \rightarrow a$$

Beispiel 4.35

$$S \rightarrow AB \mid a, \quad A \rightarrow b$$

- 1 Elimination nicht erzeugender Symbole:

$$S \rightarrow a, \quad A \rightarrow b$$

- 2 Elimination unerreichbarer Symbole:

$$S \rightarrow a$$

Umgekehrte Reihenfolge:

- 1 Elimination unerreichbarer Symbole:

$$S \rightarrow AB \mid a, \quad A \rightarrow b$$

- 2 Elimination nicht erzeugender Symbole:

$$S \rightarrow a, \quad A \rightarrow b$$

Satz 4.36

Eliminiert man aus einer Grammatik G

- ① *alle nicht erzeugenden Symbole, mit Resultat G_1 , und*
- ② *aus G_1 alle unerreichbaren Symbole, mit Resultat G_2 ,*
dann enthält G_2 nur noch nützliche Symbole und $L(G_2) = L(G)$.

Satz 4.36

Eliminiert man aus einer Grammatik G

- 1 *alle nicht erzeugenden Symbole, mit Resultat G_1 , und*
- 2 *aus G_1 alle unerreichbaren Symbole, mit Resultat G_2 ,*
dann enthält G_2 nur noch nützliche Symbole und $L(G_2) = L(G)$.

Beweis:

Wir zeigen zuerst $L(G_2) = L(G)$.

Da $P_2 \subseteq P$ gilt $L(G_2) \subseteq L(G)$.

Umgekehrt, sei $w \in L(G)$, dh $S \rightarrow_G^* w$.

Jedes Symbol in dieser Ableitung ist erreichbar und erzeugend.

Also gilt auch $S \rightarrow_{G_2}^* w$, dh $w \in L(G_2)$.

Beweis (Forts.): [G_1 : erzeugend in G , G_2 : erreichbar in G_1]

Beweis (Forts.): [G_1 : erzeugend in G , G_2 : erreichbar in G_1]

Wir zeigen: Alle $X \in V_2 \cup \Sigma_2$ sind nützlich in G_2 , dh

$$S \xrightarrow{*}_{G_2} \dots X \dots \xrightarrow{*}_{G_2} \dots \in \Sigma_2^*$$

Beweis (Forts.): [G_1 : erzeugend in G , G_2 : erreichbar in G_1]

Wir zeigen: Alle $X \in V_2 \cup \Sigma_2$ sind nützlich in G_2 , dh

$$S \rightarrow_{G_2}^* \dots X \dots \rightarrow_{G_2}^* \dots \in \Sigma_2^*$$

X muss in G_1 erreichbar sein, dh $S \rightarrow_{G_1}^* \alpha X \beta$.

Da alle Symbole in der Ableitung erreichbar sind: $S \rightarrow_{G_2}^* \alpha X \beta$.

Beweis (Forts.): [G_1 : erzeugend in G , G_2 : erreichbar in G_1]

Wir zeigen: Alle $X \in V_2 \cup \Sigma_2$ sind nützlich in G_2 , dh

$$S \rightarrow_{G_2}^* \dots X \dots \rightarrow_{G_2}^* \dots \in \Sigma_2^*$$

X muss in G_1 erreichbar sein, dh $S \rightarrow_{G_1}^* \alpha X \beta$.

Da alle Symbole in der Ableitung erreichbar sind: $S \rightarrow_{G_2}^* \alpha X \beta$.

Alle Symbole in $\alpha X \beta$ müssen in G erzeugend sein:

$$\forall Y \in \alpha X \beta. \exists u \in \Sigma^*. Y \rightarrow_G^* u$$

Da alle Symbole in den Ableitungen $Y \rightarrow_G^* u$ erzeugend sind:

$$\forall Y \in \alpha X \beta. \exists u \in \Sigma_1^*. Y \rightarrow_{G_1}^* u$$

Also gibt es $w \in \Sigma_1^*$ mit $\alpha X \beta \rightarrow_{G_1}^* w$.

Beweis (Forts.): [G_1 : erzeugend in G , G_2 : erreichbar in G_1]

Wir zeigen: Alle $X \in V_2 \cup \Sigma_2$ sind nützlich in G_2 , dh

$$S \rightarrow_{G_2}^* \dots X \dots \rightarrow_{G_2}^* \dots \in \Sigma_2^*$$

X muss in G_1 erreichbar sein, dh $S \rightarrow_{G_1}^* \alpha X \beta$.

Da alle Symbole in der Ableitung erreichbar sind: $S \rightarrow_{G_2}^* \alpha X \beta$.

Alle Symbole in $\alpha X \beta$ müssen in G erzeugend sein:

$$\forall Y \in \alpha X \beta. \exists u \in \Sigma^*. Y \rightarrow_G^* u$$

Da alle Symbole in den Ableitungen $Y \rightarrow_G^* u$ erzeugend sind:

$$\forall Y \in \alpha X \beta. \exists u \in \Sigma_1^*. Y \rightarrow_{G_1}^* u$$

Also gibt es $w \in \Sigma_1^*$ mit $\alpha X \beta \rightarrow_{G_1}^* w$.

Die Ableitung $\alpha X \beta \rightarrow_{G_1}^* w$ enthält nur Symbole, die in G_1 erreichbar sind. Daher auch $\alpha X \beta \rightarrow_{G_2}^* w$.

$$S \rightarrow_{G_2}^* \alpha X \beta \rightarrow_{G_2}^* w$$



Satz 4.37

Die Menge der erzeugenden Symbole einer CFG sind berechenbar.

Satz 4.37

Die Menge der erzeugenden Symbole einer CFG sind berechenbar.

Beweis:

Wir berechnen die erzeugenden Symbole induktiv:

- Alle Symbole in Σ sind erzeugend.
- Falls $(A \rightarrow \alpha) \in P$ und alle Symbole in α sind erzeugend, dann ist auch A erzeugend. □

Satz 4.37

Die Menge der erzeugenden Symbole einer CFG sind berechenbar.

Beweis:

Wir berechnen die erzeugenden Symbole induktiv:

- Alle Symbole in Σ sind erzeugend.
- Falls $(A \rightarrow \alpha) \in P$ und alle Symbole in α sind erzeugend, dann ist auch A erzeugend. □

Beispiel 4.38

$$S \rightarrow SAB, \quad A \rightarrow BC, \quad B \rightarrow C, \quad C \rightarrow c$$

Erzeugend: $\{c, C, B, A\}$.

Satz 4.37

Die Menge der erzeugenden Symbole einer CFG sind berechenbar.

Beweis:

Wir berechnen die erzeugenden Symbole induktiv:

- Alle Symbole in Σ sind erzeugend.
- Falls $(A \rightarrow \alpha) \in P$ und alle Symbole in α sind erzeugend, dann ist auch A erzeugend. □

Beispiel 4.38

$$S \rightarrow SAB, \quad A \rightarrow BC, \quad B \rightarrow C, \quad C \rightarrow c$$

Erzeugend: $\{c, C, B, A\}$.

Korollar 4.39

Für eine kontextfreie Grammatik G ist entscheidbar, ob $L(G) = \emptyset$.

Beweis:

$L(G) = \emptyset \iff S$ ist nicht erzeugend. □

Satz 4.40

Die Menge der erreichbaren Symbole einer CFG ist berechenbar.

Beweis:

Satz 4.40

Die Menge der erreichbaren Symbole einer CFG ist berechenbar.

Beweis:

Wir berechnen die erreichbaren Symbole induktiv:

- S ist erreichbar.
- Ist A erreichbar und $(A \rightarrow \alpha X \beta) \in P$,
so ist auch X erreichbar.



Satz 4.41

Das Wortproblem ($w \in L(G)?$) ist für eine CFG G entscheidbar.

Satz 4.41

Das Wortproblem ($w \in L(G)?$) ist für eine CFG G entscheidbar.

Beweis:

OE sei $w \neq \epsilon$. Wir eliminieren zuerst alle ϵ -Produktionen aus G (wie in Lemma 4.23).

Dann berechnen wir induktiv die Menge R aller von S ableitbaren Wörter $\in (V \cup \Sigma)^*$, die nicht länger als w sind:

- $S \in R$
- Wenn $\alpha B \gamma \in R$ und $(B \rightarrow \beta) \in P$ und $|\alpha \beta \gamma| \leq |w|$, dann auch $\alpha \beta \gamma \in R$.

Satz 4.41

Das Wortproblem ($w \in L(G)?$) ist für eine CFG G entscheidbar.

Beweis:

OE sei $w \neq \epsilon$. Wir eliminieren zuerst alle ϵ -Produktionen aus G (wie in Lemma 4.23).

Dann berechnen wir induktiv die Menge R aller von S ableitbaren Wörter $\in (V \cup \Sigma)^*$, die nicht länger als w sind:

- $S \in R$
- Wenn $\alpha B \gamma \in R$ und $(B \rightarrow \beta) \in P$ und $|\alpha \beta \gamma| \leq |w|$, dann auch $\alpha \beta \gamma \in R$.

Es gilt:

$$w \in L_V(G) \quad \Leftrightarrow \quad w \in R$$

wobei $L_V(G) := \{w \in (V \cup \Sigma)^* \mid S \rightarrow_G^* w\}$.

Da R endlich ist ($|R| \leq |V \cup \Sigma|^{|w|}$), ist $w \in R$ entscheidbar, und damit auch $w \in L_V(G)$, und damit auch $w \in L(G)$. □

4.7 Der Cocke-Younger-Kasami-Algorithmus

Der CYK-Algorithmus entscheidet das Wortproblem für kontextfreie Grammatiken in Chomsky-Normalform.

4.7 Der Cocke-Younger-Kasami-Algorithmus

Der CYK-Algorithmus entscheidet das Wortproblem für kontextfreie Grammatiken in Chomsky-Normalform.

Eingabe: Grammatik $G = (V, \Sigma, P, S)$ in Chomsky-Normalform,
 $w = a_1 \dots a_n \in \Sigma^*$.

4.7 Der Cocke-Younger-Kasami-Algorithmus

Der CYK-Algorithmus entscheidet das Wortproblem für kontextfreie Grammatiken in Chomsky-Normalform.

Eingabe: Grammatik $G = (V, \Sigma, P, S)$ in Chomsky-Normalform, $w = a_1 \dots a_n \in \Sigma^*$.

Definition 4.42

$$V_{ij} := \{A \in V \mid A \rightarrow_G^* a_i \dots a_j\} \quad \text{für } i \leq j$$

Damit gilt:

$$w \in L(G) \quad \Leftrightarrow \quad S \in V_{1n}$$

Der CYK-Algorithmus berechnet die V_{ij} rekursiv nach wachsendem $j - i$:

$$V_{ii} = \{A \in V \mid (A \rightarrow a_i) \in P\}$$

$$V_{ij} = \left\{ A \in V \mid \begin{array}{l} \exists i \leq k < j, B \in V_{ik}, C \in V_{k+1,j}. \\ (A \rightarrow BC) \in P \end{array} \right\} \quad \text{für } i < j$$

Der CYK-Algorithmus berechnet die V_{ij} rekursiv nach wachsendem $j - i$:

$$V_{ii} = \{A \in V \mid (A \rightarrow a_i) \in P\}$$

$$V_{ij} = \left\{ A \in V \mid \begin{array}{l} \exists i \leq k < j, B \in V_{ik}, C \in V_{k+1,j}. \\ (A \rightarrow BC) \in P \end{array} \right\} \quad \text{für } i < j$$

Korrektheitsbeweis: Induktion nach $j - i$.

Der CYK-Algorithmus berechnet die V_{ij} rekursiv nach wachsendem $j - i$:

$$V_{ii} = \{A \in V \mid (A \rightarrow a_i) \in P\}$$

$$V_{ij} = \left\{ A \in V \mid \begin{array}{l} \exists i \leq k < j, B \in V_{ik}, C \in V_{k+1,j}. \\ (A \rightarrow BC) \in P \end{array} \right\} \quad \text{für } i < j$$

Korrektheitsbeweis: Induktion nach $j - i$.

Die V_{ij} als Tabelle (mit ij statt V_{ij}):

14			
13	24		
12	23	34	
11	22	33	44
a_1	a_2	a_3	a_4

Beispiel 4.43

$S \rightarrow AB \mid BC$

$A \rightarrow BA \mid a$

$B \rightarrow CC \mid b$

$C \rightarrow AB \mid a$

15					
14	25				
13	24	35			
12	23	34	45		
11	22	33	44	55	
	b	a	a	b	a

Satz 4.44

Der CYK-Algorithmus entscheidet das Wortproblem $w \in L(G)$ für eine fixe CFG G in Chomsky-Normalform in Zeit $O(|w|^3)$.

Beweis:

Sei $n := |w|$. Es werden $\frac{n(n-1)}{2} \in O(n^2)$ Mengen V_{ij} berechnet.

$$V_{ij} = \left\{ A \in V \mid \begin{array}{l} \exists i \leq k < j, B \in V_{ik}, C \in V_{k+1,j} \\ (A \rightarrow BC) \in P \end{array} \right\} \quad (i < j)$$

Satz 4.44

Der CYK-Algorithmus entscheidet das Wortproblem $w \in L(G)$ für eine fixe CFG G in Chomsky-Normalform in Zeit $O(|w|^3)$.

Beweis:

Sei $n := |w|$. Es werden $\frac{n(n-1)}{2} \in O(n^2)$ Mengen V_{ij} berechnet.

$$V_{ij} = \left\{ A \in V \mid \begin{array}{l} \exists i \leq k < j, B \in V_{ik}, C \in V_{k+1,j}. \\ (A \rightarrow BC) \in P \end{array} \right\} \quad (i < j)$$

Für jede dieser Mengen werden

- $j - i < n$ Werte für k betrachtet,
- für jedes k wird für alle Produktionen $A \rightarrow BC$ untersucht, ob $B \in V_{ik}$ und $C \in V_{k+1,j}$, wobei $|V_{ik}|, |V_{k+1,j}| \leq |V|$.

Satz 4.44

Der CYK-Algorithmus entscheidet das Wortproblem $w \in L(G)$ für eine fixe CFG G in Chomsky-Normalform in Zeit $O(|w|^3)$.

Beweis:

Sei $n := |w|$. Es werden $\frac{n(n-1)}{2} \in O(n^2)$ Mengen V_{ij} berechnet.

$$V_{ij} = \left\{ A \in V \mid \begin{array}{l} \exists i \leq k < j, B \in V_{ik}, C \in V_{k+1,j} \\ (A \rightarrow BC) \in P \end{array} \right\} \quad (i < j)$$

Für jede dieser Mengen werden

- $j - i < n$ Werte für k betrachtet,
- für jedes k wird für alle Produktionen $A \rightarrow BC$ untersucht, ob $B \in V_{ik}$ und $C \in V_{k+1,j}$, wobei $|V_{ik}|, |V_{k+1,j}| \leq |V|$.

Gesamtzeit: $O(n^3)$, denn $|P|$ und $|V|$ sind Konstanten unabhängig von n . [Konstruktion jeder Menge V_{ii} : $O(1)$. Für alle V_{ii} also $O(n)$.] □

Erweiterung Der CYK-Algorithmus kann so erweitert werden, dass er nicht nur das Wortproblem entscheidet, sondern auch die Menge der Syntaxbäume für die Eingabe berechnet.

Erweiterung Der CYK-Algorithmus kann so erweitert werden, dass er nicht nur das Wortproblem entscheidet, sondern auch die Menge der Syntaxbäume für die Eingabe berechnet.

Realisierung:

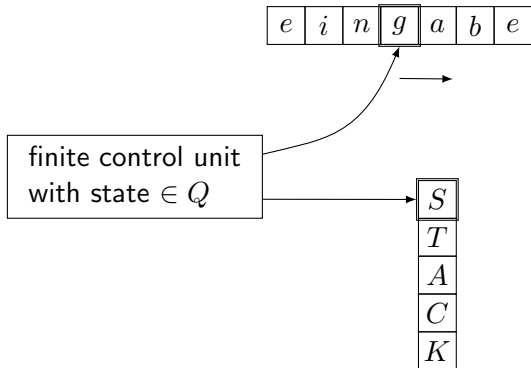
- V_{ij} ist die Menge der Syntaxbäume mit Rand $a_i \dots a_j$.
- Statt A enthält V_{ij} die Syntaxbäume, dessen Wurzel mit A beschriftet ist.

Vorschau

Für CFGs sind folgende Probleme nicht entscheidbar:

- Äquivalenz: $L(G_1) = L(G_2)$?
- Schnittproblem: $L(G_1) \cap L(G_2) = \emptyset$?
- Regularität: $L(G)$ regulär?
- Mehrdeutigkeit: Ist G mehrdeutig?

4.8 Kellerautomaten



Definition 4.45

Ein (nichtdeterministischer) **Kellerautomat**

$M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$ besteht aus

- einer endlichen Menge von **Zuständen** Q ,
- einem endlichen **Eingabealphabet** Σ ,
- einem endlichen **Kelleralphabet** Γ ,
- einem **Anfangszustand** q_0 ,
- einem **untersten Kellerzeichen** Z_0 ,
- einer **Übergangsfunktion** $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow \mathcal{P}_e(Q \times \Gamma^*)$,
(Hierbei bedeutet \mathcal{P}_e die Menge aller endlichen Teilmengen)
- einer Menge $F \subseteq Q$ von **Endzuständen**.

Definition 4.45

Ein (nichtdeterministischer) **Kellerautomat**

$M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$ besteht aus

- einer endlichen Menge von **Zuständen** Q ,
- einem endlichen **Eingabealphabet** Σ ,
- einem endlichen **Kelleralphabet** Γ ,
- einem **Anfangszustand** q_0 ,
- einem **untersten Kellerzeichen** Z_0 ,
- einer **Übergangsfunktion** $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow \mathcal{P}_e(Q \times \Gamma^*)$,
(Hierbei bedeutet \mathcal{P}_e die Menge aller endlichen Teilmengen)
- einer Menge $F \subseteq Q$ von **Endzuständen**.

Intuitive Bedeutung von $(q', \alpha) \in \delta(q, a, Z)$:

Wenn sich M in Zustand q befindet, das Eingabezeichen a liest, und Z das oberste Kellerzeichen ist, so kann M im nächsten Schritt in q' übergehen und Z durch α ersetzen.

Intuitive Bedeutung von $(q', \alpha) \in \delta(q, a, Z)$:

Wenn sich M in Zustand q befindet, das Eingabezeichen a liest, und Z das oberste Kellerzeichen ist, so kann M im nächsten Schritt in q' übergehen und Z durch α ersetzen.

Intuitive Bedeutung von $(q', \alpha) \in \delta(q, a, Z)$:

*Wenn sich M in Zustand q befindet, das Eingabezeichen a liest, und Z das oberste Kellerzeichen ist,
so kann M im nächsten Schritt in q' übergehen und Z
durch α ersetzen.*

Spezialfälle:

POP-Operation: $\alpha = \epsilon$.

Das oberste kellerzeichen Z wird entfernt.

PUSH-Operation: $\alpha = Z'Z$.

Z' wird als neues oberstes Kellerzeichen gePUSHt.

ϵ -Übergang $a = \epsilon$.

Ohne Lesen eines Eingabezeichens.

Intuitive Bedeutung von $(q', \alpha) \in \delta(q, a, Z)$:

Wenn sich M in Zustand q befindet, das Eingabezeichen a liest, und Z das oberste Kellerzeichen ist, so kann M im nächsten Schritt in q' übergehen und Z durch α ersetzen.

Spezialfälle:

POP-Operation: $\alpha = \epsilon$.

Das oberste kellerzeichen Z wird entfernt.

PUSH-Operation: $\alpha = Z'Z$.

Z' wird als neues oberstes Kellerzeichen gePUSHT.

ϵ -Übergang $a = \epsilon$.

Ohne Lesen eines Eingabezeichens.

Aufgabe: Jeder Schritt kann durch eine Sequenz von diesen drei Operationen simuliert werden. (Hilfszustände verwenden.)

Definition 4.46

Eine **Konfiguration** eines Kellerautomaten M ist ein Tripel (q, w, α) mit $q \in Q$, $w \in \Sigma^*$ und $\alpha \in \Gamma^*$.

Die **Anfangskonfiguration** von M für die Eingabe $w \in \Sigma^*$ ist (q_0, w, Z_0) .

Definition 4.46

Eine **Konfiguration** eines Kellerautomaten M ist ein Tripel (q, w, α) mit $q \in Q$, $w \in \Sigma^*$ und $\alpha \in \Gamma^*$.

Die **Anfangskonfiguration** von M für die Eingabe $w \in \Sigma^*$ ist (q_0, w, Z_0) .

Intuitiv stellt eine Konfiguration (q, w, α) eine “Momentaufnahme” des Kellerautomaten dar:

- Der momentane Zustand ist q .
- Der noch zu lesende Teil der Eingabe ist w .
- Der aktuelle Kellerinhalt ist α (das oberste Kellerzeichen ganz links stehend).

Definition 4.47

Auf der Menge aller Konfigurationen definieren wir eine binäre Relation \rightarrow_M wie folgt:

$$(q, aw, Z\alpha) \rightarrow_M \begin{cases} (q', w, \beta\alpha) & \text{falls } (q', \beta) \in \delta(q, a, Z) \\ (q', aw, \beta\alpha) & \text{falls } (q', \beta) \in \delta(q, \epsilon, Z) \end{cases}$$

Die reflexive und transitive Hülle von \rightarrow_M wird mit \rightarrow_M^* bezeichnet.

Definition 4.47

Auf der Menge aller Konfigurationen definieren wir eine binäre Relation \rightarrow_M wie folgt:

$$(q, aw, Z\alpha) \rightarrow_M \begin{cases} (q', w, \beta\alpha) & \text{falls } (q', \beta) \in \delta(q, a, Z) \\ (q', aw, \beta\alpha) & \text{falls } (q', \beta) \in \delta(q, \epsilon, Z) \end{cases}$$

Die reflexive und transitive Hülle von \rightarrow_M wird mit \rightarrow_M^* bezeichnet.

Intuitive Bedeutung von $(q, w, \alpha) \rightarrow_M (q', w', \alpha')$:

Wenn M sich in der Konfiguration (q, w, α) befindet, dann kann er in einen Schritt in die Nachfolgerkonfiguration (q', w', α') übergehen.

Achtung: eine Konfiguration kann mehrere Nachfolger haben (Nichtdeterminismus!)

Definition 4.48

- Ein PDA M akzeptiert $w \in \Sigma^*$ mit Endzustand gdw

$$(q_0, w, Z_0) \rightarrow_M^* (f, \epsilon, \gamma) \text{ für ein } f \in F, \gamma \in \Gamma^*.$$

$$L_F(M) := \{w \mid \exists f \in F, \gamma \in \Gamma^*. (q_0, w, Z_0) \rightarrow_M^* (f, \epsilon, \gamma)\}$$

Definition 4.48

- Ein PDA M akzeptiert $w \in \Sigma^*$ mit Endzustand gdw

$$(q_0, w, Z_0) \rightarrow_M^* (f, \epsilon, \gamma) \text{ f\"ur ein } f \in F, \gamma \in \Gamma^*.$$

$$L_F(M) := \{w \mid \exists f \in F, \gamma \in \Gamma^*. (q_0, w, Z_0) \rightarrow_M^* (f, \epsilon, \gamma)\}$$

- Ein PDA M akzeptiert $w \in \Sigma^*$ mit leeren Keller gdw

$$(q_0, w, Z_0) \rightarrow_M^* (q, \epsilon, \epsilon) \text{ f\"ur ein } q \in Q.$$

$$L_\epsilon(M) := \{w \mid \exists q \in Q. (q_0, w, Z_0) \rightarrow_M^* (q, \epsilon, \epsilon)\}$$

Definition 4.48

- Ein PDA M akzeptiert $w \in \Sigma^*$ mit Endzustand gdw

$$(q_0, w, Z_0) \rightarrow_M^* (f, \epsilon, \gamma) \text{ f\"ur ein } f \in F, \gamma \in \Gamma^*.$$

$$L_F(M) := \{w \mid \exists f \in F, \gamma \in \Gamma^*. (q_0, w, Z_0) \rightarrow_M^* (f, \epsilon, \gamma)\}$$

- Ein PDA M akzeptiert $w \in \Sigma^*$ mit leeren Keller gdw

$$(q_0, w, Z_0) \rightarrow_M^* (q, \epsilon, \epsilon) \text{ f\"ur ein } q \in Q.$$

$$L_\epsilon(M) := \{w \mid \exists q \in Q. (q_0, w, Z_0) \rightarrow_M^* (q, \epsilon, \epsilon)\}$$

Konvention: Wir blenden die F -Komponente von M aus, wenn wir nur an $L_\epsilon(M)$ interessiert sind.

Beispiel 4.49

Die Sprache $L = \{ww^R \mid w \in \{0, 1\}^*\}$ wird vom PDA

$$M = (\{p, q, r\}, \{0, 1\}, \{0, 1, Z_0\}, p, Z_0, \delta, \{r\})$$

$$\delta(p, a, Z) = \{(p, aZ)\} \quad \text{für } a \in \{0, 1\}, Z \in \{0, 1, Z_0\}$$

$$\delta(p, \epsilon, Z) = \{(q, Z)\} \quad \text{für } Z \in \{0, 1, Z_0\}$$

$$\delta(q, a, a) = \{(q, \epsilon)\} \quad \text{für } a \in \{0, 1\}$$

$$\delta(q, \epsilon, Z_0) = \{(r, \epsilon)\}$$

sowohl mit Endzustand als auch mit leerem Keller akzeptiert.

Bsp: $(p, 110011, Z_0) \rightarrow_M^* (r, \epsilon, \epsilon)$.

Ziel:

Akzeptanz durch Endzustände und leeren Keller gleich mächtig.

Ziel:

Akzeptanz durch Endzustände und leeren Keller gleich mächtig.

Satz 4.50 (Endzustand \rightarrow leerer Keller)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$ kann man in linearer Zeit einen PDA $M' = (Q', \Sigma, \Gamma', q'_0, Z'_0, \delta')$ konstruieren mit $L_F(M) = L_\epsilon(M')$.

Ziel:

Akzeptanz durch Endzustände und leeren Keller gleich mächtig.

Satz 4.50 (Endzustand \rightarrow leerer Keller)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$ kann man in linearer Zeit einen PDA $M' = (Q', \Sigma, \Gamma', q'_0, Z'_0, \delta')$ konstruieren mit $L_F(M) = L_\epsilon(M')$.

Ziel:

$$(q_0, w, Z_0) \rightarrow_M^* (f, \epsilon, \gamma) \quad \Leftrightarrow \quad (q'_0, w, Z'_0) \rightarrow_{M'}^* (q, \epsilon, \epsilon)$$

Beweisskizze:

M' (mit leerem Keller) simuliert M (mit Endzustand). Zusätzlich:

- Sobald M einen Endzustand erreicht, darf M' den Keller leeren (im neuen Zustand \bar{q}).
- Um zu verhindern, dass der Keller von M' leer wird, ohne dass M in einem Endzustand ist, führen wir ein neues Kellersymbol Z'_0 ein.

Beweisskizze:

M' (mit leerem Keller) simuliert M (mit Endzustand). Zusätzlich:

- Sobald M einen Endzustand erreicht, darf M' den Keller leeren (im neuen Zustand \bar{q}).
- Um zu verhindern, dass der Keller von M' leer wird, ohne dass M in einem Endzustand ist, führen wir ein neues Kellersymbol Z'_0 ein.

$$\begin{aligned}Q' &:= Q \uplus \{\bar{q}, q'_0\} \\ \Gamma' &:= \Gamma \uplus \{Z'_0\}\end{aligned}$$

Wir **erweitern** δ zu δ' :

$$\delta'(q'_0, \epsilon, Z'_0) = \{(q_0, Z_0 Z'_0)\}$$

$$\delta'(q, b, Z) = \delta(q, b, Z)$$

$$\delta'(f, a, Z) = \delta(f, a, Z)$$

$$\delta'(f, \epsilon, Z) = \delta(f, \epsilon, Z) \cup \{(\bar{q}, \epsilon)\} \quad \text{für } f \in F, Z \in \Gamma'$$

$$\delta'(\bar{q}, \epsilon, Z) = \{(\bar{q}, \epsilon)\}$$

$$\text{für } q \in Q \setminus F, b \in \Sigma \cup \{\epsilon\}, Z \in \Gamma$$

$$\text{für } f \in F, a \in \Sigma, Z \in \Gamma$$

$$\text{für } f \in F, Z \in \Gamma'$$

$$\text{für } Z \in \Gamma'$$

□

Satz 4.51 (Leerer Keller \rightarrow Endzustand)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$ kann man in linearer Zeit einen PDA $M' = (Q', \Sigma, \Gamma', q'_0, Z'_0, \delta', F)$ konstruieren mit $L_\epsilon(M) = L_F(M')$.

Satz 4.51 (Leerer Keller \rightarrow Endzustand)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$ kann man in linearer Zeit einen PDA $M' = (Q', \Sigma, \Gamma', q'_0, Z'_0, \delta', F)$ konstruieren mit $L_\epsilon(M) = L_F(M')$.

Beweisskizze:

M' (mit Endzustand) simuliert M (mit leerem Keller). Zusätzlich:

- M' schreibt am Anfang Z_0 (das unterste Kellerzeichen von M) auf den Keller, $'$ über Z'_0 .
- Sobald M' auf dem Keller Z'_0 wieder findet, ist der Keller von M leer, und M' geht in den (neuen) Endzustand f .

Satz 4.51 (Leerer Keller \rightarrow Endzustand)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$ kann man in linearer Zeit einen PDA $M' = (Q', \Sigma, \Gamma', q'_0, Z'_0, \delta', F)$ konstruieren mit $L_\epsilon(M) = L_F(M')$.

Beweisskizze:

M' (mit Endzustand) simuliert M (mit leerem Keller). Zusätzlich:

- M' schreibt am Anfang Z_0 (das unterste Kellerzeichen von M) auf den Keller, f über Z'_0 .
- Sobald M' auf dem Keller Z'_0 wieder findet, ist der Keller von M leer, und M' geht in den (neuen) Endzustand f .

$$Q' := Q \uplus \{q'_0, f\}$$

$$\Gamma' := \Gamma \uplus \{Z'_0\}$$

$$F := \{f\}$$

$$\delta'(q'_0, \epsilon, Z'_0) = \{(q_0, Z_0 Z'_0)\}$$

$$\delta'(q, b, Z) = \delta(q, b, Z) \quad \text{für } q \in Q, b \in \Sigma \cup \{\epsilon\}, Z \in \Gamma$$

$$\delta'(q, \epsilon, Z'_0) = \{(f, Z'_0)\} \quad \text{für } q \in Q \quad \square$$

Die folgenden Sätze erleichtern Beweise über Berechnungen von PDAs.

Die folgenden Sätze erleichtern Beweise über Berechnungen von PDAs.

Lemma 4.52 (Erweiterungslemma)

$$(q, u, \alpha) \rightarrow_M^n (q', u', \alpha') \implies (q, uv, \alpha\beta) \rightarrow_M^n (q', u'v, \alpha'\beta)$$

Die folgenden Sätze erleichtern Beweise über Berechnungen von PDAs.

Lemma 4.52 (Erweiterungslemma)

$$(q, u, \alpha) \rightarrow_M^n (q', u', \alpha') \implies (q, uv, \alpha\beta) \rightarrow_M^n (q', u'v, \alpha'\beta)$$

Beweis:

Nach Definition von \rightarrow_M gilt

$$\begin{aligned} (q_i, u_i, \alpha_i) \rightarrow_M (q_{i+1}, u_{i+1}, \alpha_{i+1}) &\implies \\ (q_i, u_i v, \alpha_i \beta) \rightarrow_M (q_{i+1}, u_{i+1} v, \alpha_{i+1} \beta) \end{aligned}$$

Mit Induktion über n folgt die Behauptung. □

Die folgenden Sätze erleichtern Beweise über Berechnungen von PDAs.

Lemma 4.52 (Erweiterungslemma)

$$(q, u, \alpha) \rightarrow_M^n (q', u', \alpha') \implies (q, uv, \alpha\beta) \rightarrow_M^n (q', u'v, \alpha'\beta)$$

Beweis:

Nach Definition von \rightarrow_M gilt

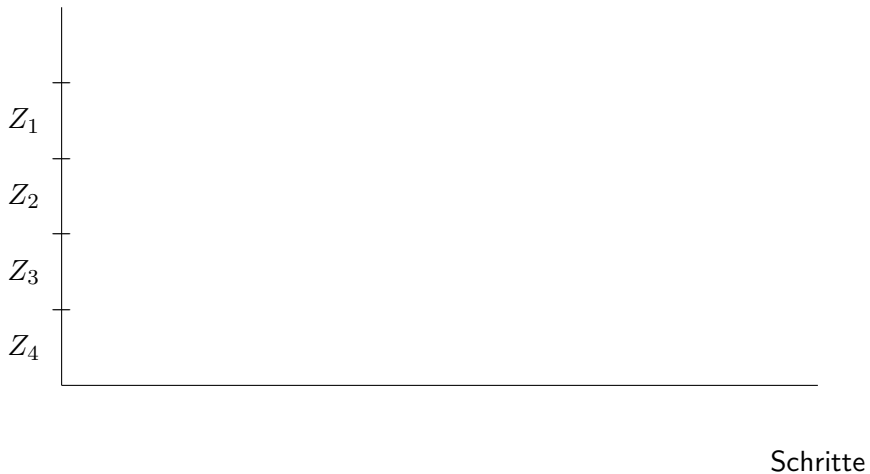
$$\begin{aligned} (q_i, u_i, \alpha_i) \rightarrow_M (q_{i+1}, u_{i+1}, \alpha_{i+1}) &\implies \\ (q_i, u_i v, \alpha_i \beta) \rightarrow_M (q_{i+1}, u_{i+1} v, \alpha_{i+1} \beta) \end{aligned}$$

Mit Induktion über n folgt die Behauptung. □

Gilt die Umkehrung, zB

$$(q_1, aa, XZ) \rightarrow_M^* (q_3, \epsilon, YZ) \implies (q_1, aa, X) \rightarrow_M^* (q_3, \epsilon, Y) ?$$

Keller



Satz 4.53 (Zerlegungssatz)

Wenn $(q, w, Z_{1\dots k}) \rightarrow_M^n (q', \epsilon, \epsilon)$

dann gibt es u_i, p_i, n_i , so dass

$$(p_{i-1}, u_i, Z_i) \rightarrow_M^{n_i} (p_i, \epsilon, \epsilon) \quad (i = 1, \dots, k)$$

und $w = u_1 \dots u_k$, $p_0 = q$, $p_k = q'$, $\sum n_i = n$.

Satz 4.53 (Zerlegungssatz)

Wenn $(q, w, Z_{1\dots k}) \rightarrow_M^n (q', \epsilon, \epsilon)$

dann gibt es u_i, p_i, n_i , so dass

$$(p_{i-1}, u_i, Z_i) \rightarrow_M^{n_i} (p_i, \epsilon, \epsilon) \quad (i = 1, \dots, k)$$

und $w = u_1 \dots u_k$, $p_0 = q$, $p_k = q'$, $\sum n_i = n$.

Beweis: Mit Induktion über n . Basis trivial.

Satz 4.53 (Zerlegungssatz)

Wenn $(q, w, Z_{1\dots k}) \rightarrow_M^n (q', \epsilon, \epsilon)$

dann gibt es u_i, p_i, n_i , so dass

$$(p_{i-1}, u_i, Z_i) \rightarrow_M^{n_i} (p_i, \epsilon, \epsilon) \quad (i = 1, \dots, k)$$

und $w = u_1 \dots u_k$, $p_0 = q$, $p_k = q'$, $\sum n_i = n$.

Beweis: Mit Induktion über n . Basis trivial.

Schritt: Eine Berechnung der Länge $n + 1$ hat die Form

$$(q, bw', Z_{1\dots k}) \rightarrow_M (p, w', Y_{1\dots l} Z_{2\dots k}) \rightarrow_M^n (q', \epsilon, \epsilon)$$

Satz 4.53 (Zerlegungssatz)

Wenn $(q, w, Z_{1\dots k}) \rightarrow_M^n (q', \epsilon, \epsilon)$
dann gibt es u_i, p_i, n_i , so dass

$$(p_{i-1}, u_i, Z_i) \rightarrow_M^{n_i} (p_i, \epsilon, \epsilon) \quad (i = 1, \dots, k)$$

und $w = u_1 \dots u_k$, $p_0 = q$, $p_k = q'$, $\sum n_i = n$.

Beweis: Mit Induktion über n . Basis trivial.

Schritt: Eine Berechnung der Länge $n + 1$ hat die Form

$$(q, bw', Z_{1\dots k}) \rightarrow_M (p, w', Y_{1\dots l} Z_{2\dots k}) \rightarrow_M^n (q', \epsilon, \epsilon)$$

IA $\Rightarrow \exists v_j, r_j, m_j$ ($j = 1, \dots, l$), $\exists u_i, p_i, n_i$ ($i = 2, \dots, k$) mit

$$(r_{j-1}, v_j, Y_j) \rightarrow_M^{m_j} (r_j, \epsilon, \epsilon) \quad (p_{i-1}, u_i, Z_i) \rightarrow_M^{n_i} (p_i, \epsilon, \epsilon)$$

und $w' = v_{1\dots l} u_{2\dots k}$, $r_0 = p$, $r_l = p_1$, $p_k = q'$, $\sum m_j + \sum n_i = n$.

Satz 4.53 (Zerlegungssatz)

Wenn $(q, w, Z_{1\dots k}) \rightarrow_M^n (q', \epsilon, \epsilon)$
dann gibt es u_i, p_i, n_i , so dass

$$(p_{i-1}, u_i, Z_i) \rightarrow_M^{n_i} (p_i, \epsilon, \epsilon) \quad (i = 1, \dots, k)$$

und $w = u_1 \dots u_k$, $p_0 = q$, $p_k = q'$, $\sum n_i = n$.

Beweis: Mit Induktion über n . Basis trivial.

Schritt: Eine Berechnung der Länge $n + 1$ hat die Form

$$(q, bw', Z_{1\dots k}) \rightarrow_M (p, w', Y_{1\dots l} Z_{2\dots k}) \rightarrow_M^n (q', \epsilon, \epsilon)$$

IA $\Rightarrow \exists v_j, r_j, m_j$ ($j = 1, \dots, l$), $\exists u_i, p_i, n_i$ ($i = 2, \dots, k$) mit

$$(r_{j-1}, v_j, Y_j) \rightarrow_M^{m_j} (r_j, \epsilon, \epsilon) \quad (p_{i-1}, u_i, Z_i) \rightarrow_M^{n_i} (p_i, \epsilon, \epsilon)$$

und $w' = v_{1\dots l} u_{2\dots k}$, $r_0 = p$, $r_l = p_1$, $p_k = q'$, $\sum m_j + \sum n_i = n$.

Mit Erweiterungslemma: $(r_{j-1}, v_{j\dots l}, Y_{j\dots l}) \rightarrow_M^{m_j} (r_j, v_{j+1\dots l}, Y_{j+1\dots l})$
 $\Rightarrow (r_0, v_{1\dots l}, Y_{1\dots l}) \rightarrow_M^{n_1-1} (r_l, \epsilon, \epsilon)$ wobei $n_1 := 1 + \sum m_j$.

Satz 4.53 (Zerlegungssatz)

Wenn $(q, w, Z_{1\dots k}) \rightarrow_M^n (q', \epsilon, \epsilon)$

dann gibt es u_i, p_i, n_i , so dass

$$(p_{i-1}, u_i, Z_i) \rightarrow_M^{n_i} (p_i, \epsilon, \epsilon) \quad (i = 1, \dots, k)$$

und $w = u_1 \dots u_k$, $p_0 = q$, $p_k = q'$, $\sum n_i = n$.

Beweis: Mit Induktion über n . Basis trivial.

Schritt: Eine Berechnung der Länge $n + 1$ hat die Form

$$(q, bw', Z_{1\dots k}) \rightarrow_M (p, w', Y_{1\dots l} Z_{2\dots k}) \rightarrow_M^n (q', \epsilon, \epsilon)$$

IA $\Rightarrow \exists v_j, r_j, m_j$ ($j = 1, \dots, l$), $\exists u_i, p_i, n_i$ ($i = 2, \dots, k$) mit

$$(r_{j-1}, v_j, Y_j) \rightarrow_M^{m_j} (r_j, \epsilon, \epsilon) \quad (p_{i-1}, u_i, Z_i) \rightarrow_M^{n_i} (p_i, \epsilon, \epsilon)$$

und $w' = v_{1\dots l} u_{2\dots k}$, $r_0 = p$, $r_l = p_1$, $p_k = q'$, $\sum m_j + \sum n_i = n$.

Mit Erweiterungslemma: $(r_{j-1}, v_{j\dots l}, Y_{j\dots l}) \rightarrow_M^{m_j} (r_j, v_{j+1\dots l}, Y_{j+1\dots l})$

$\Rightarrow (r_0, v_{1\dots l}, Y_{1\dots l}) \rightarrow_M^{n_1-1} (r_l, \epsilon, \epsilon)$ wobei $n_1 := 1 + \sum m_j$.

Aus $(q, bv_{1\dots l}, Z_1) \rightarrow_M (p, v_{1\dots l}, Y_{1\dots l})$ folgt $(p_0, u_1, Z_1) \rightarrow_M^{n_1} (p_1, \epsilon, \epsilon)$

wobei $p_0 := q$, $u_1 := bv_{1\dots l}$, $p_1 := r_l$.

Satz 4.53 (Zerlegungssatz)

Wenn $(q, w, Z_{1\dots k}) \rightarrow_M^n (q', \epsilon, \epsilon)$
dann gibt es u_i, p_i, n_i , so dass

$$(p_{i-1}, u_i, Z_i) \rightarrow_M^{n_i} (p_i, \epsilon, \epsilon) \quad (i = 1, \dots, k)$$

und $w = u_1 \dots u_k$, $p_0 = q$, $p_k = q'$, $\sum n_i = n$.

Beweis: Mit Induktion über n . Basis trivial.

Schritt: Eine Berechnung der Länge $n + 1$ hat die Form

$$(q, bw', Z_{1\dots k}) \rightarrow_M (p, w', Y_{1\dots l} Z_{2\dots k}) \rightarrow_M^n (q', \epsilon, \epsilon)$$

IA $\Rightarrow \exists v_j, r_j, m_j$ ($j = 1, \dots, l$), $\exists u_i, p_i, n_i$ ($i = 2, \dots, k$) mit

$$(r_{j-1}, v_j, Y_j) \rightarrow_M^{m_j} (r_j, \epsilon, \epsilon) \quad (p_{i-1}, u_i, Z_i) \rightarrow_M^{n_i} (p_i, \epsilon, \epsilon)$$

und $w' = v_{1\dots l} u_{2\dots k}$, $r_0 = p$, $r_l = p_1$, $p_k = q'$, $\sum m_j + \sum n_i = n$.

Mit Erweiterungslemma: $(r_{j-1}, v_{j\dots l}, Y_{j\dots l}) \rightarrow_M^{m_j} (r_j, v_{j+1\dots l}, Y_{j+1\dots l})$

$\Rightarrow (r_0, v_{1\dots l}, Y_{1\dots l}) \rightarrow_M^{n_1-1} (r_l, \epsilon, \epsilon)$ wobei $n_1 := 1 + \sum m_j$.

Aus $(q, bv_{1\dots l}, Z_1) \rightarrow_M (p, v_{1\dots l}, Y_{1\dots l})$ folgt $(p_0, u_1, Z_1) \rightarrow_M^{n_1} (p_1, \epsilon, \epsilon)$
wobei $p_0 := q$, $u_1 := bv_{1\dots l}$, $p_1 := r_l$.

Damit auch $w = bw' = bv_{1\dots l} u_{2\dots k} = u_{1\dots k}$ und $\sum n_i = n + 1$. \square

4.9 Äquivalenz von PDAs und CFGs

Satz 4.54 (CFG \rightarrow PDA)

Zu jeder CFG G kann man einen PDA M konstruieren, der mit leerem Stack akzeptiert, so dass $L_\epsilon(M) = L(G)$.

4.9 Äquivalenz von PDAs und CFGs

Satz 4.54 (CFG \rightarrow PDA)

Zu jeder CFG G kann man einen PDA M konstruieren, der mit leerem Stack akzeptiert, so dass $L_\epsilon(M) = L(G)$.

Konstruktion:

Zuerst bringen wir alle Produktionen von G in die Form

$$A \rightarrow bB_1 \dots B_k$$

wobei $b \in \Sigma \cup \{\epsilon\}$.

4.9 Äquivalenz von PDAs und CFGs

Satz 4.54 (CFG \rightarrow PDA)

Zu jeder CFG G kann man einen PDA M konstruieren, der mit leerem Stack akzeptiert, so dass $L_\epsilon(M) = L(G)$.

Konstruktion:

Zuerst bringen wir alle Produktionen von G in die Form

$$A \rightarrow bB_1 \dots B_k$$

wobei $b \in \Sigma \cup \{\epsilon\}$.

Methode: Für jedes $a \in \Sigma$

- 1 füge ein neues A_a zu V hinzu,
- 2 ersetze a rechts in P durch A_a (außer am Kopfende),
- 3 füge eine neue Produktion $A_a \rightarrow a$ hinzu.

Alle Produktionen in $G = (V, \Sigma, P, S)$ haben jetzt die Form

$$A \rightarrow bB_1 \dots B_k$$

Der PDA wird wie folgt definiert:

$$M := (\{q\}, \Sigma, V, q, S, \delta)$$

wobei

$$(A \rightarrow b\beta) \in P \implies \delta(q, b, A) \ni (q, \beta)$$

also für alle $b \in \Sigma \cup \{\epsilon\}$ und $A \in V$:

$$\delta(q, b, A) := \{(q, \beta) \mid (A \rightarrow b\beta) \in P\}$$

Lemma 4.55

Für alle $u, v \in \Sigma^*$ und $\gamma \in V^*$ und $A \in V$ gilt:

$$A \rightarrow_G^n u\gamma \text{ mit Linksableitung gdw } (q, uv, A) \rightarrow_M^n (q, v, \gamma)$$

Lemma 4.55

Für alle $u, v \in \Sigma^*$ und $\gamma \in V^*$ und $A \in V$ gilt:

$$A \rightarrow_G^n u\gamma \text{ mit Linksableitung gdw } (q, uv, A) \rightarrow_M^n (q, v, \gamma)$$

Beweis:

Mit Induktion über n . Basis $n = 0$ trivial.

Lemma 4.55

Für alle $u, v \in \Sigma^*$ und $\gamma \in V^*$ und $A \in V$ gilt:

$$A \rightarrow_G^n u\gamma \text{ mit Linksableitung gdw } (q, uv, A) \rightarrow_M^n (q, v, \gamma)$$

Beweis:

Mit Induktion über n . Basis $n = 0$ trivial. Schritt:

$$\begin{aligned} & A \rightarrow_G^{n+1} u\gamma \\ & \Leftrightarrow \\ & \exists (B \rightarrow b\beta) \in P, w, \alpha. A \rightarrow_G^n wB\alpha \rightarrow_G wb\beta\alpha = u\gamma \\ & \hspace{15em} (\text{dh } wb = u, \beta\alpha = \gamma) \\ & \Leftrightarrow \\ & (q, wbv, A) \rightarrow_M^n (q, bv, B\alpha) \wedge (q, bv, B\alpha) \rightarrow_M (q, v, \beta\alpha) \\ & \Leftrightarrow \\ & (q, uv, A) \rightarrow_M^{n+1} (q, v, \gamma) \end{aligned}$$



$A \rightarrow_G^n u\gamma$ mit Linksableitung gdw $(q, uv, A) \rightarrow_M^n (q, v, \gamma)$

Satz 4.56

$$L(G) = L_\epsilon(M)$$

$A \rightarrow_G^n u\gamma$ mit Linksableitung gdw $(q, uv, A) \rightarrow_M^n (q, v, \gamma)$

Satz 4.56

$$L(G) = L_\epsilon(M)$$

Beweis:

$$u \in L(G)$$

$$\Leftrightarrow S \rightarrow_G^* u \text{ mit Linksableitung}$$

$$\Leftrightarrow (q, u, S) \rightarrow_M^* (q, \epsilon, \epsilon)$$

$$\Leftrightarrow u \in L_\epsilon(M)$$



Satz 4.57 (PDA \rightarrow CFG)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$, der mit leerem Keller akzeptiert, kann man eine CFG G konstruieren mit $L(G) = L_\epsilon(M)$.

Satz 4.57 (PDA \rightarrow CFG)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$, der mit leerem Keller akzeptiert, kann man eine CFG G konstruieren mit $L(G) = L_\epsilon(M)$.

Konstruktion: $G := (V, \Sigma, P, S)$ mit

$V := Q \times \Gamma \times Q \cup \{S\}$ wobei wir die Tripel mit $[\cdot, \cdot, \cdot]$ notieren

und P die folgenden Produktionen enthält:

Satz 4.57 (PDA \rightarrow CFG)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$, der mit leerem Keller akzeptiert, kann man eine CFG G konstruieren mit $L(G) = L_\epsilon(M)$.

Konstruktion: $G := (V, \Sigma, P, S)$ mit

$V := Q \times \Gamma \times Q \cup \{S\}$ wobei wir die Tripel mit $[\cdot, \cdot, \cdot]$ notieren

und P die folgenden Produktionen enthält:

- $S \rightarrow [q_0, Z_0, q]$ für alle $q \in Q$
- Für alle $\delta(q, b, Z) \ni (r_0, Z_1 \dots Z_k)$ und für alle $r_1, \dots, r_k \in Q$:

$$[q, Z, r_k] \rightarrow b[r_0, Z_1, r_1][r_1, Z_2, r_2] \dots [r_{k-1}, Z_k, r_k]$$

Satz 4.57 (PDA \rightarrow CFG)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$, der mit leerem Keller akzeptiert, kann man eine CFG G konstruieren mit $L(G) = L_\epsilon(M)$.

Konstruktion: $G := (V, \Sigma, P, S)$ mit

$V := Q \times \Gamma \times Q \cup \{S\}$ wobei wir die Tripel mit $[\cdot, \cdot, \cdot]$ notieren

und P die folgenden Produktionen enthält:

- $S \rightarrow [q_0, Z_0, q]$ für alle $q \in Q$
- Für alle $\delta(q, b, Z) \ni (r_0, Z_1 \dots Z_k)$ und für alle $r_1, \dots, r_k \in Q$:

$$[q, Z, r_k] \rightarrow b[r_0, Z_1, r_1][r_1, Z_2, r_2] \dots [r_{k-1}, Z_k, r_k]$$

Idee: $[q, Z, r_k] \xrightarrow{*}_G w$ gdw $(q, w, Z) \xrightarrow{*}_M (r_k, \epsilon, \epsilon)$

Die r_1, \dots, r_k sind potenzielle Zwischenzustände beim Akzeptieren der Teilwörter von $bu_1 \dots u_k = w$, die zu Z_1, \dots, Z_k gehören.
(Zerlegungssatz!)

Lemma 4.58

$$[q, Z, p] \rightarrow_G^n w \text{ } gdw (q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$$

Lemma 4.58

$[q, Z, p] \rightarrow_G^n w \text{ gdw } (q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$

Beweis:

(\Rightarrow): Wenn $[q, Z, p] \rightarrow_G^n w$ dann $(q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$.

Mit Induktion über n .

IA: Beh. gilt für alle Werte $< n$. Zz: Beh. gilt für n .

Lemma 4.58

$$[q, Z, p] \rightarrow_G^n w \text{ gdw } (q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$$

Beweis:

(\Rightarrow): Wenn $[q, Z, p] \rightarrow_G^n w$ dann $(q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$.

Mit Induktion über n .

IA: Beh. gilt für alle Werte $< n$. Zz: Beh. gilt für n .

Die Ableitung $[q, Z, p] \rightarrow_G^n w$ muss von folgender Form sein:

$$\begin{array}{l} [q, Z, r_k] \quad \rightarrow_G \quad b[r_0, Z_1, r_1] \dots [r_{k-1}, Z_k, r_k] \\ \quad \quad \quad \rightarrow_G^{n-1} \quad bu_1 \dots u_k = w \end{array}$$

mit $r_k = p$, $(r_0, Z_1 \dots Z_k) \in \delta(q, b, Z)$,

$[r_{i-1}, Z_i, r_i] \rightarrow_G^{n_i} u_i$ ($i = 1, \dots, k$) und $\sum n_i = n - 1$.

Lemma 4.58

$$[q, Z, p] \rightarrow_G^n w \text{ gdw } (q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$$

Beweis:

(\Rightarrow): Wenn $[q, Z, p] \rightarrow_G^n w$ dann $(q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$.

Mit Induktion über n .

IA: Beh. gilt für alle Werte $< n$. Zz: Beh. gilt für n .

Die Ableitung $[q, Z, p] \rightarrow_G^n w$ muss von folgender Form sein:

$$\begin{array}{l} [q, Z, r_k] \quad \rightarrow_G \quad b[r_0, Z_1, r_1] \dots [r_{k-1}, Z_k, r_k] \\ \quad \rightarrow_G^{n-1} \quad bu_1 \dots u_k = w \end{array}$$

mit $r_k = p$, $(r_0, Z_1 \dots Z_k) \in \delta(q, b, Z)$,

$[r_{i-1}, Z_i, r_i] \rightarrow_G^{n_i} u_i$ ($i = 1, \dots, k$) und $\sum n_i = n - 1$.

IA $\Rightarrow (r_{i-1}, u_i, Z_i) \rightarrow_M^{n_i} (r_i, \epsilon, \epsilon)$

Lemma 4.58

$$[q, Z, p] \rightarrow_G^n w \text{ gdw } (q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$$

Beweis:

(\Rightarrow): Wenn $[q, Z, p] \rightarrow_G^n w$ dann $(q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$.

Mit Induktion über n .

IA: Beh. gilt für alle Werte $< n$. Zz: Beh. gilt für n .

Die Ableitung $[q, Z, p] \rightarrow_G^n w$ muss von folgender Form sein:

$$\begin{array}{l} [q, Z, r_k] \quad \rightarrow_G \quad b[r_0, Z_1, r_1] \dots [r_{k-1}, Z_k, r_k] \\ \quad \rightarrow_G^{n-1} \quad bu_1 \dots u_k = w \end{array}$$

mit $r_k = p$, $(r_0, Z_1 \dots Z_k) \in \delta(q, b, Z)$,

$[r_{i-1}, Z_i, r_i] \rightarrow_G^{n_i} u_i$ ($i = 1, \dots, k$) und $\sum n_i = n - 1$.

IA $\Rightarrow (r_{i-1}, u_i, Z_i) \rightarrow_M^{n_i} (r_i, \epsilon, \epsilon)$

Erweiterungslemma

$\Rightarrow (r_{i-1}, u_i \dots u_k, Z_i \dots Z_k) \rightarrow_M^{n_i} (r_i, u_{i+1} \dots u_k, Z_{i+1} \dots Z_k)$

$\Rightarrow (q, w, Z) \rightarrow_M (r_0, u_1 \dots u_k, Z_1 \dots Z_k) \rightarrow_M^{n-1} (r_k, \epsilon, \epsilon)$

□

Beweis (Forts.):

(\Leftarrow): Wenn $(q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$ dann $[q, Z, p] \rightarrow_G^n w$.

Mit Induktion über n .

Beweis (Forts.):

(\Leftarrow): Wenn $(q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$ dann $[q, Z, p] \rightarrow_G^n w$.

Mit Induktion über n .

IA: Beh. gilt für alle Werte $< n$. Zz: Beh. gilt für n .

Beweis (Forts.):

(\Leftarrow): Wenn $(q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$ dann $[q, Z, p] \rightarrow_G^n w$.

Mit Induktion über n .

IA: Beh. gilt für alle Werte $< n$. Zz: Beh. gilt für n .

Die Berechnung $(q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$ muss von dieser Form sein:

$$(q, w, Z) \rightarrow_M (r_0, w', Z_1 \dots Z_k) \rightarrow_M^{n-1} (p, \epsilon, \epsilon)$$

mit $w = bw'$, $(r_0, Z_1 \dots Z_k) \in \delta(q, b, Z)$.

Beweis (Forts.):

(\Leftarrow): Wenn $(q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$ dann $[q, Z, p] \rightarrow_G^n w$.

Mit Induktion über n .

IA: Beh. gilt für alle Werte $< n$. Zz: Beh. gilt für n .

Die Berechnung $(q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$ muss von dieser Form sein:

$$(q, w, Z) \rightarrow_M (r_0, w', Z_1 \dots Z_k) \rightarrow_M^{n-1} (p, \epsilon, \epsilon)$$

mit $w = bw'$, $(r_0, Z_1 \dots Z_k) \in \delta(q, b, Z)$.

Nach dem Zerlegungssatz muss es r_i , u_i und n_i geben mit

$$(r_{i-1}, u_i, Z_i) \rightarrow_M^{n_i} (r_i, \epsilon, \epsilon) \quad (i = 1, \dots, k)$$

und $w' = u_1 \dots u_k$, $\sum n_i = n - 1$.

Beweis (Forts.):

(\Leftarrow): Wenn $(q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$ dann $[q, Z, p] \rightarrow_G^n w$.

Mit Induktion über n .

IA: Beh. gilt für alle Werte $< n$. Zz: Beh. gilt für n .

Die Berechnung $(q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$ muss von dieser Form sein:

$$(q, w, Z) \rightarrow_M (r_0, w', Z_1 \dots Z_k) \rightarrow_M^{n-1} (p, \epsilon, \epsilon)$$

mit $w = bw'$, $(r_0, Z_1 \dots Z_k) \in \delta(q, b, Z)$.

Nach dem Zerlegungssatz muss es r_i , u_i und n_i geben mit

$$(r_{i-1}, u_i, Z_i) \rightarrow_M^{n_i} (r_i, \epsilon, \epsilon) \quad (i = 1, \dots, k)$$

und $w' = u_1 \dots u_k$, $\sum n_i = n - 1$.

$$\text{IA} \Rightarrow [r_{i-1}, Z_i, r_i] \rightarrow_G^{n_i} u_i$$

Beweis (Forts.):

(\Leftarrow): Wenn $(q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$ dann $[q, Z, p] \rightarrow_G^n w$.

Mit Induktion über n .

IA: Beh. gilt für alle Werte $< n$. Zz: Beh. gilt für n .

Die Berechnung $(q, w, Z) \rightarrow_M^n (p, \epsilon, \epsilon)$ muss von dieser Form sein:

$$(q, w, Z) \rightarrow_M (r_0, w', Z_1 \dots Z_k) \rightarrow_M^{n-1} (p, \epsilon, \epsilon)$$

mit $w = bw'$, $(r_0, Z_1 \dots Z_k) \in \delta(q, b, Z)$.

Nach dem Zerlegungssatz muss es r_i , u_i und n_i geben mit

$$(r_{i-1}, u_i, Z_i) \rightarrow_M^{n_i} (r_i, \epsilon, \epsilon) \quad (i = 1, \dots, k)$$

und $w' = u_1 \dots u_k$, $\sum n_i = n - 1$.

$$\text{IA} \Rightarrow [r_{i-1}, Z_i, r_i] \rightarrow_G^{n_i} u_i$$

$$\Rightarrow [q, Z, p] \rightarrow_G b[r_0, Z_1, r_1] \dots [r_{k-1}, Z_k, r_k]$$

$$\rightarrow_G^{n-1} bu_1 \dots u_k = w$$



Damit haben wir bewiesen:

Satz 4.59

Eine Sprache ist kontextfrei gdw sie von einem Kellerautomaten akzeptiert wird.

4.10 Deterministische Kellerautomaten

Definition 4.60

Ein PDA heißt **deterministisch (DPDA)** gdw für alle $q \in Q$, $a \in \Sigma$ und $Z \in \Gamma$

$$|\delta(q, a, Z)| + |\delta(q, \epsilon, Z)| \leq 1$$

4.10 Deterministische Kellerautomaten

Definition 4.60

Ein PDA heißt **deterministisch (DPDA)** gdw für alle $q \in Q$, $a \in \Sigma$ und $Z \in \Gamma$

$$|\delta(q, a, Z)| + |\delta(q, \epsilon, Z)| \leq 1$$

Beispiel 4.61

Der folgende DPDA mit $\Sigma = \{0, 1, \$\}$, $\Gamma = \{Z_0, 0, 1\}$ akzeptiert die Sprache $\{w\$w^R \mid w \in \{0, 1\}^*\}$.

Definition 4.62

Eine CFL heißt **deterministisch (DCFL)** gdw sie von einem DPDA akzeptiert wird.

Man kann zeigen: Die CFL $\{ww^R \mid w \in \{0,1\}^*\}$ ist nicht deterministisch.

Definition 4.62

Eine CFL heißt **deterministisch (DCFL)** gdw sie von einem DPDA akzeptiert wird.

Man kann zeigen: Die CFL $\{ww^R \mid w \in \{0,1\}^*\}$ ist nicht deterministisch.

Da man jeden DFA leicht mit einem DPDA simulieren kann:

Fakt 4.63

Jede reguläre Sprache ist eine DCFL.

Eine Sprache erfüllt die **Präfix Bedingung** gdw wenn sie keine zwei Wörter enthält, so dass das eine ein *echtes* Präfix des anderen ist.

Beispiel 4.64

- Die Sprache a^* erfüllt die Präfix Bedingung nicht.
- Die Sprache $\{w\$w^R \mid w \in \{0,1\}^*\}$ erfüllt die Präfix Bedingung.

Eine Sprache erfüllt die **Präfix Bedingung** gdw wenn sie keine zwei Wörter enthält, so dass das eine ein *echtes* Präfix des anderen ist.

Beispiel 4.64

- Die Sprache a^* erfüllt die Präfix Bedingung nicht.
- Die Sprache $\{w\$w^R \mid w \in \{0, 1\}^*\}$ erfüllt die Präfix Bedingung.

Lemma 4.65

$\exists DPDA M. L = L_\epsilon(M) \iff$

$\exists DPDA M. L = L_F(M)$ und L erfüllt die Präfix Bedingung

Beweis: Übung!

Eine Sprache erfüllt die **Präfix Bedingung** gdw wenn sie keine zwei Wörter enthält, so dass das eine ein *echtes* Präfix des anderen ist.

Beispiel 4.64

- Die Sprache a^* erfüllt die Präfix Bedingung nicht.
- Die Sprache $\{w\$w^R \mid w \in \{0,1\}^*\}$ erfüllt die Präfix Bedingung.

Lemma 4.65

$\exists \text{DPDA } M. L = L_\epsilon(M) \iff$
 $\exists \text{DPDA } M. L = L_F(M) \text{ und } L \text{ erfüllt die Präfix Bedingung}$

Beweis: Übung!

Es gibt also einen DPDA M mit $L_F(M) = L(a^*)$ aber keinen DPDA mit $L_\epsilon(M) = L(a^*)$.

Im Folgenden: Akzeptanz durch Endzustand.

Satz 4.66

Die Klasse der DCFLs ist unter Komplement abgeschlossen.

Beweis: Komplex. Siehe zB Erk&Prieese oder Kozen.

Da die CFLs *nicht* unter Komplement abgeschlossen sind:

Korollar 4.67

Die DCFLs sind eine echte Teilklasse der CFLs.

Satz 4.66

Die Klasse der DCFLs ist unter Komplement abgeschlossen.

Beweis: Komplex. Siehe zB Erk&Prieese oder Kozen.

Da die CFLs *nicht* unter Komplement abgeschlossen sind:

Korollar 4.67

Die DCFLs sind eine echte Teilklasse der CFLs.

Hier ist eine konkrete Sprache in $\text{CFL} \setminus \text{DCFL}$:

Sei $L_{ww} := \{ww \mid w \in \{a, b\}^*\}$.

Satz 4.66

Die Klasse der DCFLs ist unter Komplement abgeschlossen.

Beweis: Komplex. Siehe zB Erk&Prieese oder Kozen.

Da die CFLs *nicht* unter Komplement abgeschlossen sind:

Korollar 4.67

Die DCFLs sind eine echte Teilklasse der CFLs.

Hier ist eine konkrete Sprache in $CFL \setminus DCFL$:

Sei $L_{ww} := \{ww \mid w \in \{a, b\}^*\}$.

Dann ist $L := \{a, b\}^* \setminus L_{ww}$ eine CFL aber keine DCFL.

L wird von folgender CFG erzeugt (ohne Beweis):

$$\begin{aligned} S &\rightarrow AB \mid BA \mid A \mid B \\ A &\rightarrow CAC \mid a \quad B \rightarrow CBC \mid b \quad C \rightarrow a \mid b \end{aligned}$$

Wäre L eine DCFL, dann auch $\bar{L} = L_{ww}$ (wegen Satz 4.66).

Aber mit dem Pumping Lemma kann man zeigen, dass L_{ww} keine CFL ist, und daher erst recht keine DCFL.

Lemma 4.68

Die Klasse der DCFLs ist weder unter Schnitt noch unter Vereinigung abgeschlossen.

Lemma 4.68

Die Klasse der DCFLs ist weder unter Schnitt noch unter Vereinigung abgeschlossen.

Beweis:

Erinnerung: CFLs nicht unter Schnitt abgeschlossen:

$L_1 := \{a^i b^i c^j \mid i, j \in \mathbb{N}\}$ ist kontextfrei

$L_2 := \{a^i b^j c^j \mid i, j \in \mathbb{N}\}$ ist kontextfrei

Lemma 4.68

Die Klasse der DCFLs ist weder unter Schnitt noch unter Vereinigung abgeschlossen.

Beweis:

Erinnerung: CFLs nicht unter Schnitt abgeschlossen:

$L_1 := \{a^i b^i c^j \mid i, j \in \mathbb{N}\}$ ist kontextfrei

$L_2 := \{a^i b^j c^j \mid i, j \in \mathbb{N}\}$ ist kontextfrei

$L_1 \cap L_2 = \{a^i b^i c^i \mid i \in \mathbb{N}\}$ ist nicht kontextfrei

Aber L_1 und L_2 sind sogar DCFLs.

Lemma 4.68

Die Klasse der DCFLs ist weder unter Schnitt noch unter Vereinigung abgeschlossen.

Beweis:

Erinnerung: CFLs nicht unter Schnitt abgeschlossen:

$L_1 := \{a^i b^i c^j \mid i, j \in \mathbb{N}\}$ ist kontextfrei

$L_2 := \{a^i b^j c^j \mid i, j \in \mathbb{N}\}$ ist kontextfrei

$L_1 \cap L_2 = \{a^i b^i c^i \mid i \in \mathbb{N}\}$ ist nicht kontextfrei

Aber L_1 und L_2 sind sogar DCFLs.

Da $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ können die DCFLs auch nicht unter Vereinigung abgeschlossen sein. □

Lemma 4.69

Jede DCFL ist nicht inhärent mehrdeutig, dh sie wird von einer nicht-mehrdeutigen Grammatik erzeugt.

Beweisidee: Die Konversion $PDA \rightarrow CFG$ erzeugt aus einem DPDA eine nicht-mehrdeutige CFG. [HMU]

Lemma 4.69

Jede DCFL ist nicht inhärent mehrdeutig, dh sie wird von einer nicht-mehrdeutigen Grammatik erzeugt.

Beweisidee: Die Konversion $PDA \rightarrow CFG$ erzeugt aus einem DPDA eine nicht-mehrdeutige CFG. [HMU]

Satz 4.70

Das Wortproblem für DCFLs ist in linearer Zeit lösbar.

Lemma 4.69

Jede DCFL ist nicht inhärent mehrdeutig, dh sie wird von einer nicht-mehrdeutigen Grammatik erzeugt.

Beweisidee: Die Konversion $PDA \rightarrow CFG$ erzeugt aus einem DPDA eine nicht-mehrdeutige CFG. [HMU]

Satz 4.70

Das Wortproblem für DCFLs ist in linearer Zeit lösbar.

Mehr Information: Vorlesungen zum Übersetzerbau

4.11 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern

4.11 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Regulär	ja	ja	ja	ja	ja

4.11 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Regulär	ja	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein

4.11 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Regulär	ja	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein
CFL	nein	ja	nein	ja	ja

4.11 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Regulär	ja	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein
CFL	nein	ja	nein	ja	ja

Entscheidbarkeit

	Wortproblem			
DFA				
DPDA				
CFG				

4.11 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Regulär	ja	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein
CFL	nein	ja	nein	ja	ja

Entscheidbarkeit

	Wortproblem			
DFA	$O(n)$			
DPDA	$O(n)$			
CFG	$O(n^3)$			

4.11 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Regulär	ja	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein
CFL	nein	ja	nein	ja	ja

Entscheidbarkeit

	Wortproblem	Leerheit		
DFA	$O(n)$			
DPDA	$O(n)$			
CFG	$O(n^3)$			

4.11 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Regulär	ja	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein
CFL	nein	ja	nein	ja	ja

Entscheidbarkeit

	Wortproblem	Leerheit		
DFA	$O(n)$	ja		
DPDA	$O(n)$	ja		
CFG	$O(n^3)$	ja		

4.11 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Regulär	ja	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein
CFL	nein	ja	nein	ja	ja

Entscheidbarkeit

	Wortproblem	Leerheit	Äquivalenz	
DFA	$O(n)$	ja		
DPDA	$O(n)$	ja		
CFG	$O(n^3)$	ja		

4.11 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Regulär	ja	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein
CFL	nein	ja	nein	ja	ja

Entscheidbarkeit

	Wortproblem	Leerheit	Äquivalenz	
DFA	$O(n)$	ja	ja	
DPDA	$O(n)$	ja	ja	
CFG	$O(n^3)$	ja	nein(*)	

Sénizergues (1997), Stirling (2001)

(*) Vorschau

4.11 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Regulär	ja	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein
CFL	nein	ja	nein	ja	ja

Entscheidbarkeit

	Wortproblem	Leerheit	Äquivalenz	Schnittproblem
DFA	$O(n)$	ja	ja	
DPDA	$O(n)$	ja	ja	
CFG	$O(n^3)$	ja	nein(*)	

Sénizergues (1997), Stirling (2001)

(*) Vorschau

4.11 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Regulär	ja	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein
CFL	nein	ja	nein	ja	ja

Entscheidbarkeit

	Wortproblem	Leerheit	Äquivalenz	Schnittproblem
DFA	$O(n)$	ja	ja	ja
DPDA	$O(n)$	ja	ja	nein(*)
CFG	$O(n^3)$	ja	nein(*)	nein(*)

Sénizergues (1997), Stirling (2001)

(*) Vorschau

5. Berechenbarkeit, Entscheidbarkeit

5. Berechenbarkeit, Entscheidbarkeit

Überblick:

- Was kann man berechnen? Dh:
 - Welche Funktionen kann man in endlicher Zeit berechnen?
 - Welche Eigenschaften von Objekten können in endlicher Zeit entschieden werden?
- Mit welchen Sprachen/Maschinen?
- Was kann man in polynomieller Zeit berechnen?

5.1 Der Begriff der Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist **intuitiv berechenbar**, wenn es einen Algorithmus gibt, der bei Eingabe $(n_1, \dots, n_k) \in \mathbb{N}^k$

- nach endlich vielen Schritten mit Ergebnis $f(n_1, \dots, n_k)$ hält, falls $f(n_1, \dots, n_k)$ definiert ist,
- und nicht terminiert, falls $f(n_1, \dots, n_k)$ nicht definiert ist.

Was bedeutet „Algorithmus“? Assembler? C? Java? OCaml?
Macht es einen Unterschied?

Terminologie: Eine Funktion $f : A \rightarrow B$ ist

total gdw $f(a)$ für alle $a \in A$ definiert ist.

partiell gdw $f(a)$ auch undefiniert sein kann.

echt partiell gdw sie nicht total ist.

Terminologie: Eine Funktion $f : A \rightarrow B$ ist

total gdw $f(a)$ für alle $a \in A$ definiert ist.

partiell gdw $f(a)$ auch undefiniert sein kann.

echt partiell gdw sie nicht total ist.

Beispiel 5.1

Jeder Algorithmus berechnet eine partielle Funktion.

Der Algorithmus

```
input(n);  
while true do ;
```

berechnet die überall undefinierte Funktion, dh $\emptyset \subset \mathbb{N} \rightarrow \mathbb{N}$.

Beispiel 5.2

Ist die Funktion

$$f_1(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge Anfangsstück der} \\ & \text{Dezimalbruchentwicklung von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_1(31415) = 1$ aber $f_1(315) = 0$)

Beispiel 5.2

Ist die Funktion

$$f_1(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge Anfangsstück der} \\ & \text{Dezimalbruchentwicklung von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_1(31415) = 1$ aber $f_1(315) = 0$)

Ja, denn es Algorithmen gibt, die π iterativ auf beliebig viele Dezimalstellen genau berechnet werden.

Beispiel 5.3

Ist die Funktion

$$f_2(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge irgendwo in der} \\ & \text{Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_2(415) = 1$)

Beispiel 5.3

Ist die Funktion

$$f_2(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge irgendwo in der} \\ & \text{Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_2(415) = 1$)

Unbekannt!

Durch schrittweise Approximation und Suche in der Dezimalbruchentwicklung von π kann man feststellen, dass n vorkommt. Aber wie stellt man fest, dass n *nicht* vorkommt? Nichttermination statt 0!

Beispiel 5.3

Ist die Funktion

$$f_2(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge irgendwo in der} \\ & \text{Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_2(415) = 1$)

Unbekannt!

Durch schrittweise Approximation und Suche in der Dezimalbruchentwicklung von π kann man feststellen, dass n vorkommt. Aber wie stellt man fest, dass n *nicht* vorkommt? Nichttermination statt 0!

Vielleicht gibt es aber einen (noch zu findenden) mathematischen Satz, der genaue Aussagen über die in π vorkommenden Ziffernfolgen macht.

Beispiel 5.4

Ist die Funktion

$$f_3(n) := \begin{cases} 1 & \text{falls } \underbrace{00 \dots 00}_{10^9 \text{ Nullen}} \\ & \text{in der Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Beispiel 5.4

Ist die Funktion

$$f_3(n) := \begin{cases} 1 & \text{falls } \underbrace{00 \dots 00}_{10^9 \text{ Nullen}} \\ & \text{in der Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn f_3 ist entweder die konstante Funktion, die 1 für alle $n \geq 0$ zurückgibt, oder die konstante Funktion, die 0 für alle $n \geq 0$ zurückgibt. Beide sind berechenbar.

Beispiel 5.4

Ist die Funktion

$$f_3(n) := \begin{cases} 1 & \text{falls } \underbrace{00 \dots 00}_{10^9 \text{ Nullen}} \\ & \text{in der Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn f_3 ist entweder die konstante Funktion, die 1 für alle $n \geq 0$ zurückgibt, oder die konstante Funktion, die 0 für alle $n \geq 0$ zurückgibt. Beide sind berechenbar.

Dies ist ein **nicht-konstruktiver Beweis**:

Wir wissen, es gibt einen Algorithmus, der f_3 berechnet, aber wir wissen nicht, welcher.

Beispiel 5.5

Ist die Funktion

$$f_4(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der} \\ & \text{Dezimalbruchentwicklung von } \pi \text{ eine } 0 \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Beispiel 5.5

Ist die Funktion

$$f_4(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der} \\ & \text{Dezimalbruchentwicklung von } \pi \text{ eine } 0 \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn

- entweder kommt 0^n für beliebig große n vor, dann ist $f_3(n) = 1$ für alle n ,
- oder es gibt eine längste vorkommende Sequenz 0^m , dann ist $f_3(n) = 1$ für $n \leq m$ und $f_3(n) = 0$ sonst.

Beide Funktionen sind berechenbar.

Beispiel 5.5

Ist die Funktion

$$f_4(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der} \\ & \text{Dezimalbruchentwicklung von } \pi \text{ eine } 0 \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn

- entweder kommt 0^n für beliebig große n vor, dann ist $f_3(n) = 1$ für alle n ,
- oder es gibt eine längste vorkommende Sequenz 0^m , dann ist $f_3(n) = 1$ für $n \leq m$ und $f_3(n) = 0$ sonst.

Beide Funktionen sind berechenbar.

Wieder ein **nicht-konstruktiver Beweis**.

Satz 5.6

Es gibt nicht-berechenbare Funktionen $\mathbb{N} \rightarrow \{0, 1\}$.

Satz 5.6

Es gibt nicht-berechenbare Funktionen $\mathbb{N} \rightarrow \{0, 1\}$.

Beweis:

Es gibt nur **abzählbar** viele Algorithmen, aber **überabzählbar** viele Funktionen in $\mathbb{N} \rightarrow \{0, 1\}$. □

Erinnerung: Eine Menge M ist **abzählbar** falls es eine injektive Funktion $M \rightarrow \mathbb{N}$ gibt.

Äquivalente Definitionen :

- Entweder gibt es eine Bijektion $M \rightarrow \{0, \dots, n\}$ für ein $n \in \mathbb{N}$, oder eine Bijektion $M \rightarrow \mathbb{N}$.
- Es gibt eine Nummerierung der Elemente von M .

Eine Menge ist **überabzählbar** wenn sie nicht abzählbar ist.

Lemma 5.7

Σ^* ist abzählbar (falls Σ endlich).

Lemma 5.7

Σ^* ist abzählbar (falls Σ endlich).

Beweis:

Durch Beispiel:

$$\begin{aligned} \{0, 1\}^* &= \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \dots\} \\ \mathbb{N} &= \{0, 1, 2, 3, 4, 5, 6, 7, \dots\} \end{aligned}$$



Lemma 5.7

Σ^* ist abzählbar (falls Σ endlich).

Beweis:

Durch Beispiel:

$$\begin{aligned} \{0, 1\}^* &= \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \dots\} \\ \mathbb{N} &= \{0, 1, 2, 3, 4, 5, 6, 7, \dots\} \end{aligned}$$



Annahme:

Ein Algorithmus ist ein Wort über einem endlichen Alphabet.

Gilt für alle bekannten Programmiersprachen.

Lemma 5.7

Σ^* ist abzählbar (falls Σ endlich).

Beweis:

Durch Beispiel:

$$\begin{aligned} \{0, 1\}^* &= \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \dots\} \\ \mathbb{N} &= \{0, 1, 2, 3, 4, 5, 6, 7, \dots\} \end{aligned}$$



Annahme:

Ein Algorithmus ist ein Wort über einem endlichen Alphabet.

Gilt für alle bekannten Programmiersprachen.

\implies Die Menge der Algorithmen ist abzählbar.

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \rightarrow \{0, 1\}$ ist überabzählbar.

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \rightarrow \{0, 1\}$ ist überabzählbar.

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

f_0	
f_1	
f_2	
f_3	
\vdots	

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \rightarrow \{0, 1\}$ ist überabzählbar.

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	...
f_0					
f_1					
f_2					
f_3					
\vdots					

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \rightarrow \{0, 1\}$ ist überabzählbar.

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	...
f_0	0	1	1	0	...
f_1					
f_2					
f_3					
\vdots					

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \rightarrow \{0, 1\}$ ist überabzählbar.

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	...
f_0	0	1	1	0	...
f_1	1	0	0	0	...
f_2					
f_3					
\vdots					

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \rightarrow \{0, 1\}$ ist überabzählbar.

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	...
f_0	0	1	1	0	...
f_1	1	0	0	0	...
f_2	0	0	1	0	...
f_3					
\vdots					

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \rightarrow \{0, 1\}$ ist überabzählbar.

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	...
f_0	0	1	1	0	...
f_1	1	0	0	0	...
f_2	0	0	1	0	...
f_3	0	0	1	0	...
\vdots					

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \rightarrow \{0, 1\}$ ist überabzählbar.

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	...
f_0	0	1	1	0	...
f_1	1	0	0	0	...
f_2	0	0	1	0	...
f_3	0	0	1	0	...
\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \rightarrow \{0, 1\}$ ist überabzählbar.

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	...
f_0	0	1	1	0	...
f_1	1	0	0	0	...
f_2	0	0	1	0	...
f_3	0	0	1	0	...
\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

Eine Funktion f , die nicht in der Tabelle ist: \neg Diagonale!

f | 1 1 0 0 ...

Widerspruch!

Satz 5.8

Die Menge aller Funktionen $\mathbb{N} \rightarrow \{0, 1\}$ ist überabzählbar.

Beweis:

Indirekt. Nimm an, die Menge der Funktionen sei abzählbar.

	0	1	2	3	...
f_0	0	1	1	0	...
f_1	1	0	0	0	...
f_2	0	0	1	0	...
f_3	0	0	1	0	...
\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

Eine Funktion f , die nicht in der Tabelle ist: \neg Diagonale!

f | 1 1 0 0 ...

Widerspruch!

Formal: Sei $f(i) := 1 - f_i(i)$. Dann $f \neq f_i$ für alle i , da $f(i) \neq f_i(i)$.



Satz 5.9

Wenn Algorithmen als endliche Wörter kodiert werden können, dann gibt es unberechenbare Funktionen $\mathbb{N} \rightarrow \{0, 1\}$.

Satz 5.9

Wenn Algorithmen als endliche Wörter kodiert werden können, dann gibt es unberechenbare Funktionen $\mathbb{N} \rightarrow \{0, 1\}$.

Beweis:

Sei \mathcal{F} die Menge aller Funktionen $\mathbb{N} \rightarrow \{0, 1\}$.

Sei \mathcal{A} die Menge der Wörter, die Algorithmen kodieren.

Beweis durch Widerspruch.

Satz 5.9

Wenn Algorithmen als endliche Wörter kodiert werden können, dann gibt es unberechenbare Funktionen $\mathbb{N} \rightarrow \{0, 1\}$.

Beweis:

Sei \mathcal{F} die Menge aller Funktionen $\mathbb{N} \rightarrow \{0, 1\}$.

Sei \mathcal{A} die Menge der Wörter, die Algorithmen kodieren.

Beweis durch Widerspruch.

Annahme: Alle Funktionen von \mathcal{F} sind berechenbar.

Satz 5.9

Wenn Algorithmen als endliche Wörter kodiert werden können, dann gibt es unberechenbare Funktionen $\mathbb{N} \rightarrow \{0, 1\}$.

Beweis:

Sei \mathcal{F} die Menge aller Funktionen $\mathbb{N} \rightarrow \{0, 1\}$.

Sei \mathcal{A} die Menge der Wörter, die Algorithmen kodieren.

Beweis durch Widerspruch.

Annahme: Alle Funktionen von \mathcal{F} sind berechenbar.

Da \mathcal{A} abzählbar ist (Lemma 5.7), gibt es eine injektive Funktion $anum: \mathcal{A} \rightarrow \mathbb{N}$.

Definiere $fnum: \mathcal{F} \rightarrow \mathbb{N}$ durch

$$fnum(f) = \min\{anum(a) \mid a \text{ berechnet } f\}$$

Satz 5.9

Wenn Algorithmen als endliche Wörter kodiert werden können, dann gibt es unberechenbare Funktionen $\mathbb{N} \rightarrow \{0, 1\}$.

Beweis:

Sei \mathcal{F} die Menge aller Funktionen $\mathbb{N} \rightarrow \{0, 1\}$.

Sei \mathcal{A} die Menge der Wörter, die Algorithmen kodieren.

Beweis durch Widerspruch.

Annahme: Alle Funktionen von \mathcal{F} sind berechenbar.

Da \mathcal{A} abzählbar ist (Lemma 5.7), gibt es eine injektive Funktion $anum: \mathcal{A} \rightarrow \mathbb{N}$.

Definiere $fnum: \mathcal{F} \rightarrow \mathbb{N}$ durch

$$fnum(f) = \min\{anum(a) \mid a \text{ berechnet } f\}$$

$fnum$ is injektiv: Wenn $fnum(f_1) = fnum(f_2)$ dann gibt es einen Algorithmus der sowohl f_1 wie auch f_2 berechnet und so $f_1 = f_2$.

Aber dann ist \mathcal{F} abzählbar. **Widerspruch zu Satz 5.8** □

Verschiedene Formalisierungen des Begriffs der Berechenbarkeit:

- Turing-Maschinen (Turing 1936)
- λ -Kalkül (Church 1936)
- μ -rekursive Funktionen
- Markov-Algorithmen
- Registermaschinen
- Awk, Basic, C, Dylan, Eiffel, Fortran, Java, Lisp, Modula, Oberon, Pascal, Python, Ruby, Simula, T_EX, ...-Programme
- while-Programme
- goto-Programme
- DNA-Algorithmen
- Quantenalgorithmen
- ...

Es wurde bewiesen: Alle diese Beschreibungsmethoden sind in ihrer Mächtigkeit äquivalent.

Churchsche These / Church-Turing These

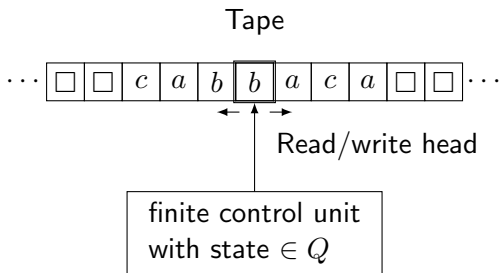
Der formale Begriff der Berechenbarkeit mit Turing-Maschinen (bzw λ -Kalkül etc) stimmt mit dem intuitiven Berechenbarkeitsbegriff überein.

Es wurde bewiesen: Alle diese Beschreibungsmethoden sind in ihrer Mächtigkeit äquivalent.

Churchsche These / Church-Turing These

Der formale Begriff der Berechenbarkeit mit Turing-Maschinen (bzw λ -Kalkül etc) stimmt mit dem intuitiven Berechenbarkeitsbegriff überein.

Die Church-Turing-These ist keine formale Aussage und so nicht beweisbar. Sie wird jedoch allgemein akzeptiert.



Definition 5.10

Eine **Turingmaschine (TM)** ist ein 7-Tupel

$M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ so dass

- Q ist eine endliche Menge von **Zuständen**.
- Σ ist eine endliche Menge, das **Eingabealphabet**.
- Γ ist eine endliche Menge, das **Bandalphabet**, mit $\Sigma \subset \Gamma$
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, N\}$ ist die **Übergangsfunktion**.
 δ darf partiell sein.
- $q_0 \in Q$ ist der **Startzustand**.
- $\square \in \Gamma \setminus \Sigma$ ist das **Leerzeichen**.
- $F \subseteq Q$ ist die Menge der **akzeptierenden** oder **Endzustände**.

Definition 5.10

Eine **Turingmaschine (TM)** ist ein 7-Tupel

$M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ so dass

- Q ist eine endliche Menge von **Zuständen**.
- Σ ist eine endliche Menge, das **Eingabealphabet**.
- Γ ist eine endliche Menge, das **Bandalphabet**, mit $\Sigma \subset \Gamma$
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, N\}$ ist die **Übergangsfunktion**.
 δ darf partiell sein.
- $q_0 \in Q$ ist der **Startzustand**.
- $\square \in \Gamma \setminus \Sigma$ ist das **Leerzeichen**.
- $F \subseteq Q$ ist die Menge der **akzeptierenden** oder **Endzustände**.

Annahme: $\delta(q, a)$ ist nicht definiert für alle $q \in F$ und $a \in \Gamma$.

Definition 5.10

Eine **Turingmaschine (TM)** ist ein 7-Tupel

$M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ so dass

- Q ist eine endliche Menge von **Zuständen**.
- Σ ist eine endliche Menge, das **Eingabealphabet**.
- Γ ist eine endliche Menge, das **Bandalphabet**, mit $\Sigma \subset \Gamma$
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, N\}$ ist die **Übergangsfunktion**.
 δ darf partiell sein.
- $q_0 \in Q$ ist der **Startzustand**.
- $\square \in \Gamma \setminus \Sigma$ ist das **Leerzeichen**.
- $F \subseteq Q$ ist die Menge der **akzeptierenden** oder **Endzustände**.

Annahme: $\delta(q, a)$ ist nicht definiert für alle $q \in F$ und $a \in \Gamma$.

Eine **nichtdeterministische** Turingmaschine hat eine Übergangsfunktion $\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R, N\})$.

Intuitiv bedeutet $\delta(q, a) = (q', b, D)$:

- Wenn sich M im Zustand q befindet,
- und auf dem Band a liest,
- so geht M im nächsten Schritt in den Zustand q' über,
- überschreibt a mit b ,
- und bewegt danach den Schreib-/Lesekopf nach **rechts** (falls $D = R$), nach **links** (falls $D = L$), oder **nicht** (falls $D = N$).

Definition 5.11

Eine **Konfiguration** einer Turingmaschine ist ein Tripel $(\alpha, q, \beta) \in \Gamma^* \times Q \times \Gamma^*$.

Dies modelliert

- Bandinhalt: $\dots \square \alpha \beta \square \dots$
- Zustand: q
- Kopf auf dem ersten Zeichen von $\beta \square$

Definition 5.11

Eine **Konfiguration** einer Turingmaschine ist ein Tripel $(\alpha, q, \beta) \in \Gamma^* \times Q \times \Gamma^*$.

Dies modelliert

- Bandinhalt: $\dots \square \alpha \beta \square \dots$
- Zustand: q
- Kopf auf dem ersten Zeichen von $\beta \square$

Die **Startkonfiguration** der Turingmaschine bei Eingabe $w \in \Sigma^*$ ist (ϵ, q_0, w) .

Die Berechnung der TM M wird als Relation \rightarrow_M auf Konfigurationen formalisiert. Falls $\delta(q, first(\beta)) = (q', c, D)$:

$$(\alpha, q, \beta) \rightarrow_M \begin{cases} (\alpha, q', c \text{ rest}(\beta)) & \text{falls } D = N \\ (\alpha c, q', \text{rest}(\beta)) & \text{falls } D = R \\ (\text{butlast}(\alpha), q', \text{last}(\alpha) c \text{ rest}(\beta)) & \text{falls } D = L \end{cases}$$

wobei

$$\begin{aligned} first(aw) &= a & first(\epsilon) &= \square \\ rest(aw) &= w & rest(\epsilon) &= \epsilon \\ last(wa) &= a & last(\epsilon) &= \square \\ butlast(wa) &= w & butlast(\epsilon) &= \epsilon \end{aligned}$$

für $a \in \Gamma$ und $w \in \Gamma^*$.

Falls M nichtdeterministisch ist: $\delta(q, first(\beta)) \ni (q', c, D)$

Beispiel 5.12 (Unär +1)

$$M = (\{q, f\}, \{1\}, \{1, \square\}, \delta, q, \square, \{f\})$$

$$\delta(q, \square) = (f, 1, N)$$

$$\delta(q, 1) = (q, 1, R)$$

Beispiel 5.12 (Unär +1)

$$M = (\{q, f\}, \{1\}, \{1, \square\}, \delta, q, \square, \{f\})$$

$$\delta(q, \square) = (f, 1, N)$$

$$\delta(q, 1) = (q, 1, R)$$

- Beispiellauf:

$$(\epsilon, q, 11) \rightarrow_M (1, q, 1) \rightarrow_M (11, q, \epsilon) \rightarrow_M (11, f, 1)$$

Beispiel 5.13 (Binär +1)

ZB $1011 \mapsto 1100$

Beispiel 5.13 (Binär +1)

ZB $1011 \mapsto 1100$

$$M = (\{q_0, q_1, q_2, q_f\}, \{0, 1\}, \{0, 1, \square\}, \delta, q_0, \square, \{q_f\})$$

wobei

$$\begin{aligned} \delta(q_0, 0) &= (q_0, 0, R) & \delta(q_1, 1) &= (q_1, 0, L) & \delta(q_2, 0) &= (q_2, 0, L) \\ \delta(q_0, 1) &= (q_0, 1, R) & \delta(q_1, 0) &= (q_2, 1, L) & \delta(q_2, 1) &= (q_2, 1, L) \\ \delta(q_0, \square) &= (q_1, \square, L) & \delta(q_1, \square) &= (q_f, 1, N) & \delta(q_2, \square) &= (q_f, \square, R) \end{aligned}$$

Beispiel 5.13 (Binär +1)

ZB $1011 \mapsto 1100$

$$M = (\{q_0, q_1, q_2, q_f\}, \{0, 1\}, \{0, 1, \square\}, \delta, q_0, \square, \{q_f\})$$

wobei

$$\begin{aligned} \delta(q_0, 0) &= (q_0, 0, R) & \delta(q_1, 1) &= (q_1, 0, L) & \delta(q_2, 0) &= (q_2, 0, L) \\ \delta(q_0, 1) &= (q_0, 1, R) & \delta(q_1, 0) &= (q_2, 1, L) & \delta(q_2, 1) &= (q_2, 1, L) \\ \delta(q_0, \square) &= (q_1, \square, L) & \delta(q_1, \square) &= (q_f, 1, N) & \delta(q_2, \square) &= (q_f, \square, R) \end{aligned}$$

Beispiellauf:

$$\begin{aligned} (\epsilon, q_0, 101) &\rightarrow_M (1, q_0, 01) \rightarrow_M (10, q_0, 1) \rightarrow_M (101, q_0, \epsilon) \rightarrow_M \\ (10, q_1, 1\square) &\rightarrow_M (1, q_1, 00\square) \rightarrow_M \\ (\epsilon, q_2, 110\square) &\rightarrow_M (\epsilon, q_2, \square 110\square) \rightarrow_M \\ (\square, q_f, 110\square) & \end{aligned}$$

Definition 5.14

Eine Turingmaschine M akzeptiert die Sprache

$$L(M) = \{w \in \Sigma^* \mid \exists q \in F, \alpha, \beta \in \Gamma^*. (\epsilon, q_0, w) \rightarrow_M^* (\alpha, q, \beta)\}$$

Definition 5.14

Eine Turingmaschine M **akzeptiert** die Sprache

$$L(M) = \{w \in \Sigma^* \mid \exists q \in F, \alpha, \beta \in \Gamma^*. (\epsilon, q_0, w) \rightarrow_M^* (\alpha, q, \beta)\}$$

Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ heißt **Turing-berechenbar** gdw es eine Turingmaschine M gibt, so dass für alle $n_1, \dots, n_k, m \in \mathbb{N}$ gilt

$$\begin{aligned} f(n_1, \dots, n_k) = m &\iff \\ \exists r \in F. (\epsilon, q_0, \text{bin}(n_1)\#\text{bin}(n_2)\#\dots\#\text{bin}(n_k)) & \\ \rightarrow_M^* (\square\dots\square, r, \text{bin}(m)\square\dots\square) & \end{aligned}$$

wobei $\text{bin}(n)$ die Binärdarstellung der Zahl n ist.

Definition 5.14

Eine Turingmaschine M **akzeptiert** die Sprache

$$L(M) = \{w \in \Sigma^* \mid \exists q \in F, \alpha, \beta \in \Gamma^*. (\epsilon, q_0, w) \rightarrow_M^* (\alpha, q, \beta)\}$$

Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ heißt **Turing-berechenbar** gdw es eine Turingmaschine M gibt, so dass für alle $n_1, \dots, n_k, m \in \mathbb{N}$ gilt

$$\begin{aligned} f(n_1, \dots, n_k) = m &\Leftrightarrow \\ \exists r \in F. (\epsilon, q_0, \text{bin}(n_1)\#\text{bin}(n_2)\#\dots\#\text{bin}(n_k)) & \\ \rightarrow_M^* (\square\dots\square, r, \text{bin}(m)\square\dots\square) & \end{aligned}$$

wobei $\text{bin}(n)$ die Binärdarstellung der Zahl n ist.

Eine Funktion $f : \Sigma^* \rightarrow \Sigma^*$ heißt **Turing-berechenbar** gdw es eine Turingmaschine M gibt, so dass für alle $u, v \in \Sigma^*$ gilt

$$f(u) = v \Leftrightarrow \exists r \in F. (\epsilon, q_0, u) \rightarrow_M^* (\square\dots\square, r, v\square\dots\square)$$

Zum Halten/Terminieren von TM

Zum Halten/Terminieren von TM

Eine TM **hält** wenn sie eine Konfiguration $(\alpha, q, a\beta)$ erreicht und $\delta(q, a)$ nicht definiert oder (bei nichtdeterministische TM)
 $\delta(q, a) = \emptyset$.

Zum Halten/Terminieren von TM

Eine TM **hält** wenn sie eine Konfiguration $(\alpha, q, a\beta)$ erreicht und $\delta(q, a)$ nicht definiert oder (bei nichtdeterministische TM)
 $\delta(q, a) = \emptyset$.

Nach Annahme hält eine TM immer, wenn sie einen Endzustand erreicht. Damit ist die von einer TM berechnete Funktion wohldefiniert.

Zum Halten/Terminieren von TM

Eine TM **hält** wenn sie eine Konfiguration $(\alpha, q, a\beta)$ erreicht und $\delta(q, a)$ nicht definiert oder (bei nichtdeterministische TM)
 $\delta(q, a) = \emptyset$.

Nach Annahme hält eine TM immer, wenn sie einen Endzustand erreicht. Damit ist die von einer TM berechnete Funktion wohldefiniert.

Eine TM kann auch halten, bevor sie einen Endzustand erreicht.

Satz 5.15

Zu jeder nichtdeterministischen TM N gibt es eine deterministische TM M mit $L(N) = L(M)$.

Satz 5.15

Zu jeder nichtdeterministischen TM N gibt es eine deterministische TM M mit $L(N) = L(M)$.

Beweis:

M durchsucht den Baum der Berechnungen von N in *Breitensuche*, beginnend mit der Startkonfiguration (ϵ, q_0, w) , eine Ebene nach der anderen.

Satz 5.15

Zu jeder nichtdeterministischen TM N gibt es eine deterministische TM M mit $L(N) = L(M)$.

Beweis:

M durchsucht den Baum der Berechnungen von N in *Breitensuche*, beginnend mit der Startkonfiguration (ϵ, q_0, w) , eine Ebene nach der anderen.

Gibt es in dem Baum (auf Ebene n) eine Konfigurationen mit Endzustand, so wird diese (nach Zeit $O(c^n)$) gefunden. □

Satz 5.16

Die von Turingmaschinen akzeptierten Sprachen sind genau die Typ-0-Sprachen der Chomsky Hierarchie.

Satz 5.16

Die von Turingmaschinen akzeptierten Sprachen sind genau die Typ-0-Sprachen der Chomsky Hierarchie.

Beweis:

Wir beschreiben nur die Beweisidee. (Mehr Details: [Schöning]).

Satz 5.16

Die von Turingmaschinen akzeptierten Sprachen sind genau die Typ-0-Sprachen der Chomsky Hierarchie.

Beweis:

Wir beschreiben nur die Beweisidee. (Mehr Details: [Schöning]).

„ \implies “: Grammatikregeln können direkt die Rechenregeln einer TM simulieren.

Satz 5.16

Die von Turingmaschinen akzeptierten Sprachen sind genau die Typ-0-Sprachen der Chomsky Hierarchie.

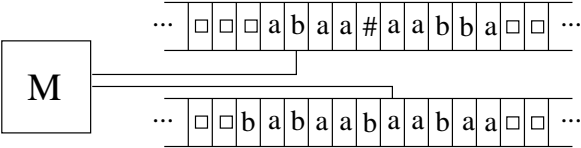
Beweis:

Wir beschreiben nur die Beweisidee. (Mehr Details: [Schöning]).

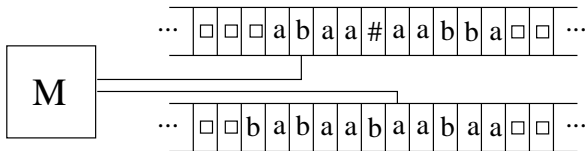
„ \implies “: Grammatikregeln können direkt die Rechenregeln einer TM simulieren.

„ \impliedby “: Die (nichtdeterministische) TM versucht von ihrer Eingabe aus das Startsymbol der Grammatik zu erreichen, indem sie die Produktionen der Grammatik von rechts nach links anwendet, (nichtdeterministisch) an jeder möglichen Stelle. □

Eine beliebige Modellvariante ist die k -Band-Turingmaschine:



Eine beliebige Modellvariante ist die k -Band-Turingmaschine:



Die k Köpfe sind völlig unabhängig voneinander:

$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, N\}^k$$

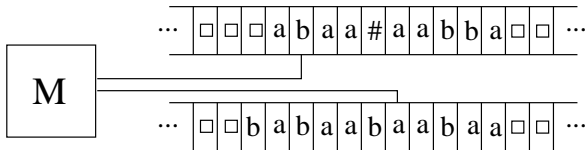
Satz 5.17

Jede k -Band-Turingmaschine kann effektiv durch eine 1-Band-TM simuliert werden.

Satz 5.17

Jede k -Band-Turingmaschine kann effektiv durch eine 1-Band-TM simuliert werden.

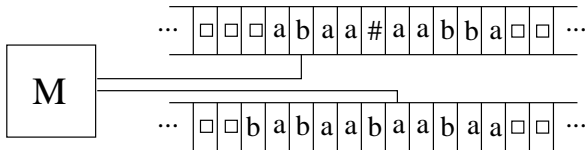
Beweisidee: Aus



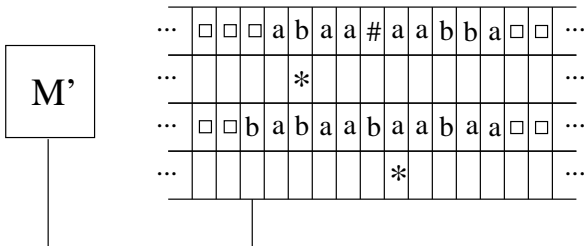
Satz 5.17

Jede k -Band-Turingmaschine kann effektiv durch eine 1-Band-TM simuliert werden.

Beweisidee: Aus



wird



Beweisskizze:

- $\Gamma' := (\Gamma \times \{\star, \square\})^k$
- M' simuliert einen M -Schritt durch mehrere Schritte: M'
 - startet mit Kopf links von allen \star .
 - geht nach rechts bis alle \star überschritten sind, und merkt sich dabei (in Q') die Zeichen über jedem \star .
 - hat jetzt alle Information, um δ_M anzuwenden.
 - geht nach links über alle \star hinweg und führt dabei δ_M aus.

Beweisskizze:

- $\Gamma' := (\Gamma \times \{\star, \square\})^k$
- M' simuliert einen M -Schritt durch mehrere Schritte: M'
 - startet mit Kopf links von allen \star .
 - geht nach rechts bis alle \star überschritten sind, und merkt sich dabei (in Q') die Zeichen über jedem \star .
 - hat jetzt alle Information, um δ_M anzuwenden.
 - geht nach links über alle \star hinweg und führt dabei δ_M aus.

Beobachtung:

n Schritte von M lassens sich durch
 $O(n^2)$ Schritte von M' simulieren.

Denn nach n Schritten von M trennen $\leq 2n$ Felder linken und rechten Kopf. Obige Simulation eines M -Schritts braucht daher $O(n)$ M' -Schritte. Simulation von n Schritten: $O(n^2)$ Schritte.

Die folgenden Basismaschinen sind leicht programmierbar:

- Band $i := \text{Band } i + 1$
- Band $i := \text{Band } i - 1$
- Band $i := 0$
- Band $i := \text{Band } j$

Seien $M_i = (Q_i, \Sigma, \Gamma_i, \delta_i, q_i, \square, F_i)$, $i = 1, 2$.

Die **sequentielle Komposition** (Hintereinanderschaltung) von M_1 und M_2 bezeichnen wir mit

$$\longrightarrow M_1 \longrightarrow M_2 \longrightarrow$$

Sie ist wie folgt definiert:

$$M := (Q_1 \cup Q_2, \Sigma, \Gamma_1 \cup \Gamma_2, \delta, q_1, \square, F_2)$$

wobei (oE) $Q_1 \cap Q_2 = \emptyset$ und

$$\delta := \delta_1 \cup \delta_2 \cup \{(f_1, a) \mapsto (q_2, a, N) \mid f_1 \in F_1, a \in \Gamma_1\}$$

Sind f_1 und f_2 Endzustände von M so bezeichnet

$$\begin{array}{c} \longrightarrow M \xrightarrow{f_1} M_1 \longrightarrow \\ \downarrow f_2 \\ M_2 \\ \downarrow \end{array}$$

eine **Fallunterscheidung**, dh eine TM, die vom Endzustand f_1 von M nach M_1 übergeht, und von f_2 aus nach M_2 .

Sind f_1 und f_2 Endzustände von M so bezeichnet

$$\begin{array}{c} \longrightarrow M \xrightarrow{f_1} M_1 \longrightarrow \\ \downarrow f_2 \\ M_2 \\ \downarrow \end{array}$$

eine **Fallunterscheidung**, dh eine TM, die vom Endzustand f_1 von M nach M_1 übergeht, und von f_2 aus nach M_2 .

Die folgende TM nennen wir „Band=0?“ (bzw „Band $i = 0$?“):

$$\begin{aligned} \delta(q_0, 0) &= (q_0, 0, R) \\ \delta(q_0, \square) &= (ja, \square, L) \\ \delta(q_0, a) &= (nein, a, N) \quad \text{für } a \neq 0, \square \end{aligned}$$

wobei ja und $nein$ Endzustände sind.

Analog zur Fallunterscheidung kann man auch eine TM für eine **Schleife** konstruieren

$$\begin{array}{l} \longrightarrow \text{Band } i = 0? \xrightarrow{\text{ja}} \\ \quad \uparrow \downarrow \text{nein} \\ \quad M \end{array}$$

die sich wie `while Band $i \neq 0$ do M` verhält.

Analog zur Fallunterscheidung kann man auch eine TM für eine **Schleife** konstruieren

$$\begin{array}{l} \longrightarrow \text{Band } i = 0? \xrightarrow{\text{ja}} \\ \uparrow \downarrow \text{nein} \\ M \end{array}$$

die sich wie `while Band $i \neq 0$ do M` verhält.

Moral: Mit TM kann man imperativ programmieren:

```
:=  
;  
if  
while
```

WHILE \equiv strukturierte Programme
mit `while`-Schleifen

GOTO \equiv Assembler

Wir definieren WHILE- und GOTO-Berechenbarkeit und zeigen ihre Äquivalenz mit Turing-Berechenbarkeit.

WHILE \equiv strukturierte Programme
mit while-Schleifen

GOTO \equiv Assembler

Wir definieren WHILE- und GOTO-Berechenbarkeit und zeigen ihre Äquivalenz mit Turing-Berechenbarkeit.

Syntax von WHILE-Programmen:

$$\begin{aligned} P &\rightarrow X := X + C \\ &| X := X - C \\ &| P ; P \\ &| \text{IF } X = 0 \text{ DO } P \text{ ELSE } Q \text{ END} \\ &| \text{WHILE } X \neq 0 \text{ DO } P \text{ END} \end{aligned}$$

wobei X eine der Variablen x_0, x_1, \dots und C eine der Konstanten $0, 1, \dots$ sein kann.

WHILE \equiv strukturierte Programme
mit while-Schleifen

GOTO \equiv Assembler

Wir definieren WHILE- und GOTO-Berechenbarkeit und zeigen ihre Äquivalenz mit Turing-Berechenbarkeit.

Syntax von WHILE-Programmen:

$$\begin{aligned} P &\rightarrow X := X + C \\ &| X := X - C \\ &| P ; P \\ &| \text{IF } X = 0 \text{ DO } P \text{ ELSE } Q \text{ END} \\ &| \text{WHILE } X \neq 0 \text{ DO } P \text{ END} \end{aligned}$$

wobei X eine der Variablen x_0, x_1, \dots und C eine der Konstanten $0, 1, \dots$ sein kann.

Beispiel 5.18

WHILE $x_2 \neq 0$ DO $x_1 := x_0 + 1$ END

Die **modifizierte Differenz** ist $m \dot{-} n := \begin{cases} m - n & \text{falls } m \geq n \\ 0 & \text{sonst} \end{cases}$

Semantik von WHILE-Programmen (informell):

$x_i := x_j + n$ Neuer Wert von x_i ist $x_j + n$.

$x_i := x_j - n$ Neuer Wert von x_i ist $x_j \dot{-} n$.

$P_1 ; P_2$ Führe zuerst P_1 und dann P_2 aus.

WHILE $x_i \neq 0$ **DO** P **END** Führe P bis die Variable x_i (wenn je) den Wert 0 annimmt.

Die modifizierte Differenz ist $m \dot{-} n := \begin{cases} m - n & \text{falls } m \geq n \\ 0 & \text{sonst} \end{cases}$

Semantik von WHILE-Programmen (informell):

$x_i := x_j + n$ Neuer Wert von x_i ist $x_j + n$.

$x_i := x_j - n$ Neuer Wert von x_i ist $x_j \dot{-} n$.

$P_1 ; P_2$ Führe zuerst P_1 und dann P_2 aus.

WHILE $x_i \neq 0$ **DO** P **END** Führe P bis die Variable x_i (wenn je) den Wert 0 annimmt.

Beispiel 5.19

WHILE $x_1 \neq 0$ **DO** $x_2 := x_2 + 1; x_1 := x_1 - 1$ **END**

simuliert

$x_2 := x_2 + x_1$

Die **modifizierte Differenz** ist $m \dot{-} n := \begin{cases} m - n & \text{falls } m \geq n \\ 0 & \text{sonst} \end{cases}$

Semantik von WHILE-Programmen (informell):

$x_i := x_j + n$ Neuer Wert von x_i ist $x_j + n$.

$x_i := x_j - n$ Neuer Wert von x_i ist $x_j \dot{-} n$.

$P_1 ; P_2$ Führe zuerst P_1 und dann P_2 aus.

WHILE $x_i \neq 0$ **DO** P **END** Führe P bis die Variable x_i (wenn je) den Wert 0 annimmt.

Beispiel 5.19

WHILE $x_1 \neq 0$ **DO** $x_2 := x_2 + 1; x_1 := x_1 - 1$ **END**

simuliert

$x_2 := x_2 + x_1$

Zu Beginn der Ausführung stehen die Eingaben in x_1, \dots, x_k .

Alle anderen Variablen sind 0. Die Ausgabe wird in x_0 berechnet.

Syntaktische Abkürzungen („Zucker“):

$$x_i := x_j \equiv x_i := x_j + 0$$

$$x_i := n \equiv x_i := x_j + n$$

(x_j wird sonst nirgends benutzt)

$$x_i := x_j + x_k \equiv x_i := x_j; x_\ell := x_k$$

($i \neq k$)

WHILE $x_k \neq 0$ DO

$$x_i := x_i + 1; x_k := x_k - 1$$

END;

$$x_k := x_\ell$$

(x_ℓ wird sonst nirgends benutzt)

$$x_i := x_j * x_k \equiv x_i := 0; x_\ell := x_k$$

($i \neq j, k$)

WHILE $x_k \neq 0$ DO

$$x_i := x_i + x_j; x_k := x_k - 1$$

END;

$$x_k := x_\ell$$

(x_ℓ wird sonst nirgends benutzt)

DIV, MOD, IF $x = n$ THEN P END,

IF $x = n$ AND $y = m$ THEN P END ...

Definition 5.20

Eine **totale** Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist **WHILE-berechenbar** gdw es ein WHILE-Programm P gibt, so dass für alle $n_1, \dots, n_k \in \mathbb{N}$:

P , gestartet mit n_1, \dots, n_k in x_1, \dots, x_k (0 in den anderen Var.) terminiert mit $f(n_1, \dots, n_k)$ in x_0 .

Definition 5.20

Eine **totale** Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist **WHILE-berechenbar** gdw es ein WHILE-Programm P gibt, so dass für alle $n_1, \dots, n_k \in \mathbb{N}$:

P , gestartet mit n_1, \dots, n_k in x_1, \dots, x_k (0 in den anderen Var.) terminiert mit $f(n_1, \dots, n_k)$ in x_0 .

Definition 5.21

Eine **partielle** Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist **WHILE-berechenbar** gdw es ein WHILE-Programm P gibt, so dass für alle $n_1, \dots, n_k \in \mathbb{N}$:

P , gestartet mit n_1, \dots, n_k in x_1, \dots, x_k (0 in den anderen Var.)

- terminiert mit $f(n_1, \dots, n_k)$ in x_0 , falls $f(n_1, \dots, n_k)$ definiert ist,
- terminiert nicht, falls $f(n_1, \dots, n_k)$ undefiniert ist.

Turingmaschinen können WHILE-Programme simulieren:

Satz 5.22 (WHILE \rightarrow TM)

Jede WHILE-berechenbare Funktion ist auch Turing-berechenbar.

Beweis:

Jede Programmvariable wird auf einem eigenen Band gespeichert. Wir haben bereits gezeigt: Alle Konstrukte der WHILE-Sprache können von einer Mehrband-TM simuliert werden, und eine Mehrband-TM kann von einer 1-Band TM simuliert werden. \square

GOTO

TM

WHILE



Übersetzungen

Ein **GOTO-Programm** ist eine Sequenzen von markierten Anweisungen

$$M_1 : A_1; M_2 : A_2; \dots; M_k : A_k$$

(wobei alle Marken verschieden und optional sind)

Mögliche Anweisungen A_i sind:

$$x_i := x_j + n$$

$$x_i := x_j - n$$

GOTO M_i

IF $x_i = n$ **GOTO** M_j

HALT

Die Semantik ist wie erwartet.

Ein **GOTO-Programm** ist eine Sequenzen von markierten Anweisungen

$$M_1 : A_1; M_2 : A_2; \dots; M_k : A_k$$

(wobei alle Marken verschieden und optional sind)

Mögliche Anweisungen A_i sind:

$$x_i := x_j + n$$

$$x_i := x_j - n$$

GOTO M_i

IF $x_i = n$ GOTO M_j

HALT

Die Semantik ist wie erwartet.

Fakt 5.23 (WHILE \rightarrow GOTO)

Jedes WHILE-Programm kann durch ein GOTO-Programm simuliert werden.

Satz 5.24 (GOTO \rightarrow WHILE)

Jedes GOTO-Programm kann durch ein WHILE-Programm simuliert werden.

Satz 5.24 (GOTO \rightarrow WHILE)

Jedes GOTO-Programm kann durch ein WHILE-Programm simuliert werden.

Beweis: Simuliere $M_1 : A_1; M_2 : A_2; \dots; M_k : A_k$ durch

```
pc := 1;
WHILE pc  $\neq$  0 DO
  IF pc = 1 THEN  $P_1$  ELSE
    :
  IF pc = k THEN  $P_k$  ELSE pc := 0
END
```

wobei $A_i \mapsto P_i$ wie folgt definiert ist:

$x_i := x_j +/- n$	\mapsto	$x_i := x_j +/- n; pc := pc + 1$
GOTO M_i	\mapsto	$pc := i$
IF $x_j = 0$ GOTO M_i	\mapsto	IF $x_j = 0$ THEN $pc := i$ ELSE $pc := pc + 1$ END
HALT	\mapsto	$pc := 0$



Korollar 5.25

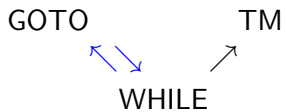
WHILE- und GOTO-Berechenbarkeit sind äquivalent.

Korollar 5.25

WHILE- und GOTO-Berechenbarkeit sind äquivalent.

Korollar 5.26 (Kleenesche Normalform)

Jedes WHILE-Programm ist zu einem WHILE-Programm mit genau einer WHILE-Schleife äquivalent.



Übersetzungen

Satz 5.27 (TM \rightarrow GOTO)

Jede TM kann durch ein GOTO-Programm simuliert werden.

Satz 5.27 (TM \rightarrow GOTO)

Jede TM kann durch ein GOTO-Programm simuliert werden.

Übersetzung: TM $(Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ \rightarrow GOTO-Programm.

Satz 5.27 (TM \rightarrow GOTO)

Jede TM kann durch ein GOTO-Programm simuliert werden.

Übersetzung: TM $(Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ \rightarrow GOTO-Programm.

$$Q = \{q_0, \dots, q_k\} \quad \Gamma = \{a_0(= \square), \dots, a_n\}$$

Satz 5.27 (TM \rightarrow GOTO)

Jede TM kann durch ein GOTO-Programm simuliert werden.

Übersetzung: TM $(Q, \Sigma, \Gamma, \delta, q_0, \square, F) \rightarrow$ GOTO-Programm.

$$Q = \{q_0, \dots, q_k\} \quad \Gamma = \{a_0(= \square), \dots, a_n\}$$

Eine Konfiguration

$$(a_{i_p} \dots a_{i_1}, q_l, a_{j_1} \dots a_{j_q})$$

wird durch die Programmvariablen x, y, z wie folgt repräsentiert:

$$x = (i_p \dots i_1)_b, \quad y = (j_q \dots j_1)_b, \quad z = l$$

wobei $(i_p \dots i_1)_b$ die Zahl $i_p \dots i_1$ zur Basis $b := n + 1$ ist:

$$x = \sum_{r=1}^p i_r b^{r-1}$$

Der Kern des GOTO-Programms ist die iterierte Simulation von δ :

```
M:   IF  $z \in F$  GOTO  $M_{end}$ ;  
       $a := y \text{ MOD } b$ ;  
      IF  $z = 0$  AND  $a = 0$  GOTO  $M_{00}$ ;  
      IF  $z = 0$  AND  $a = 1$  GOTO  $M_{01}$ ;  
      ...  
      IF  $z = k$  AND  $a = n$  GOTO  $M_{kn}$ ;  
M00:  $P_{00}$ ; GOTO M;  
M01:  $P_{01}$ ; GOTO M;  
      ...  
Mkn:  $P_{kn}$ ; GOTO M
```

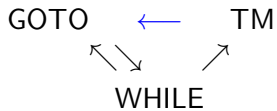
wobei P_{ij} die Simulation von $\delta(q_i, a_j) = (q_r, a_s, D)$ ist.

Der Kern des GOTO-Programms ist die iterierte Simulation von δ :

```
M:   IF  $z \in F$  GOTO  $M_{end}$ ;
       $a := y \text{ MOD } b$ ;
      IF  $z = 0$  AND  $a = 0$  GOTO  $M_{00}$ ;
      IF  $z = 0$  AND  $a = 1$  GOTO  $M_{01}$ ;
      ...
      IF  $z = k$  AND  $a = n$  GOTO  $M_{kn}$ ;
M00:  $P_{00}$ ; GOTO  $M$ ;
M01:  $P_{01}$ ; GOTO  $M$ ;
      ...
Mkn:  $P_{kn}$ ; GOTO  $M$ 
```

wobei P_{ij} die Simulation von $\delta(q_i, a_j) = (q_r, a_s, D)$ ist. Für $D = L$:

$z := r$;	Zustand aktualisieren
$y := y \text{ DIV } b$;	Löschen von a_j
$y := b*y + s$;	Schreiben von a_s
$y := b*y + (x \text{ MOD } b)$;	Bewegung L (I): Einfügen von a_{i_1} in y
$x := x \text{ DIV } b$	Bewegung L (II): Löschen von a_{i_1} aus x



Übersetzungen

5.2 Unentscheidbarkeit des Halteproblems

Definition 5.28

Eine Menge A ($\subseteq \mathbb{N}$ oder Σ^*) heißt **entscheidbar** gdw ihre **charakteristische Funktion**

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

berechenbar ist.

5.2 Unentscheidbarkeit des Halteproblems

Definition 5.28

Eine Menge A ($\subseteq \mathbb{N}$ oder Σ^*) heißt **entscheidbar** gdw ihre **charakteristische Funktion**

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

berechenbar ist.

Eine Eigenschaft/Problem $P(x)$ heißt **entscheidbar** gdw $\{x \mid P(x)\}$ entscheidbar ist.

5.2 Unentscheidbarkeit des Halteproblems

Definition 5.28

Eine Menge A ($\subseteq \mathbb{N}$ oder Σ^*) heißt **entscheidbar** gdw ihre **charakteristische Funktion**

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

berechenbar ist.

Eine Eigenschaft/Problem $P(x)$ heißt **entscheidbar** gdw $\{x \mid P(x)\}$ entscheidbar ist.

Fakt 5.29

- *Ist $A \subseteq \Sigma^*$ entscheidbar, dann gibt es eine TM M mit $L(M) = A$ (aber nicht umgekehrt).*
- *Die entscheidbaren Mengen sind abgeschlossen unter Komplement: Ist A entscheidbar, dann auch \overline{A} .*

Kodierung einer TM als Wort über $\Sigma = \{0, 1\}$, exemplarisch:

- Sei $\Gamma = \{a_0, \dots, a_k\}$ und $Q = \{q_0, \dots, q_n\}$.
- $\delta(q_i, a_j) = (q_{i'}, a_{j'}, d)$ wird kodiert als

$$\#bin(i)\#bin(j)\#bin(i')\#bin(j')\#bin(m)$$

wobei $bin : \mathbb{N} \rightarrow \{0, 1\}^*$ die Binärkodierung einer Zahl ist und $m = 0/1/2$ falls $d = L/R/N$.

- Kodierung von δ : Konkatenation der Kodierungen aller $\delta(.,.) = (.,.,.)$, in beliebiger Reihenfolge.
- Kodierung von $\{0, 1, \#\}^*$ in $\{0, 1\}^*$:

$$0 \mapsto 00$$

$$1 \mapsto 01$$

$$\# \mapsto 11$$

Die Kodierung $TM \rightarrow \{0, 1\}^*$ ist nicht surjektiv, dh nicht jedes Wort über $\{0, 1\}^*$ kodiert eine TM.

Sie kann jedoch ohne Probleme zu einer surjektiven Kodierung erweitert werden.

Die Kodierung $TM \rightarrow \{0, 1\}^*$ ist nicht surjektiv, dh nicht jedes Wort über $\{0, 1\}^*$ kodiert eine TM.

Sie kann jedoch ohne Probleme zu einer surjektiven Kodierung erweitert werden.

Sei \hat{M} eine beliebige feste TM.

Definition 5.30

Die zu einem Wort $w \in \{0, 1\}^*$ gehörige TM M_w ist

$$M_w := \begin{cases} M & \text{falls } w \text{ Kodierung von } M \text{ ist} \\ \hat{M} & \text{sonst} \end{cases}$$

Definition 5.31

$M[w]$ ist Abk. für „Maschine M mit Eingabe w “

$M[w]\downarrow$ bedeutet, dass $M[w]$ terminiert/hält.

Definition 5.31

$M[w]$ ist Abk. für „Maschine M mit Eingabe w “

$M[w]\downarrow$ bedeutet, dass $M[w]$ terminiert/hält.

Definition 5.32 (Spezielles Halteproblem)

Gegeben: Ein Wort $w \in \{0, 1\}^*$.

Problem: Hält M_w bei Eingabe w ?

Definition 5.31

$M[w]$ ist Abk. für „Maschine M mit Eingabe w “

$M[w]\downarrow$ bedeutet, dass $M[w]$ terminiert/hält.

Definition 5.32 (Spezielles Halteproblem)

Gegeben: Ein Wort $w \in \{0, 1\}^*$.

Problem: Hält M_w bei Eingabe w ?

Als Menge:

$$K := \{w \in \{0, 1\}^* \mid M_w[w]\downarrow\}$$

Satz 5.33

Das spezielle Halteproblem ist nicht entscheidbar.

Satz 5.33

Das spezielle Halteproblem ist nicht entscheidbar.

Beweis:

Angenommen, K sei entscheidbar, dh χ_K ist berechenbar.

Satz 5.33

Das spezielle Halteproblem ist nicht entscheidbar.

Beweis:

Angenommen, K sei entscheidbar, dh χ_K ist berechenbar.

Dann ist auch folgende Funktion f berechenbar:

$$f(w) := \begin{cases} 0 & \text{falls } \chi_K(w) = 0 \\ \perp & \text{falls } \chi_K(w) = 1 \end{cases}$$

Satz 5.33

Das spezielle Halteproblem ist nicht entscheidbar.

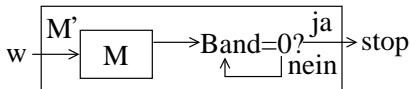
Beweis:

Angenommen, K sei entscheidbar, dh χ_K ist berechenbar.

Dann ist auch folgende Funktion f berechenbar:

$$f(w) := \begin{cases} 0 & \text{falls } \chi_K(w) = 0 \\ \perp & \text{falls } \chi_K(w) = 1 \end{cases}$$

In der Tat, berechnet eine TM M die Funktion χ_K so berechnet die folgende TM M' die Funktion f :



Satz 5.33

Das spezielle Halteproblem ist nicht entscheidbar.

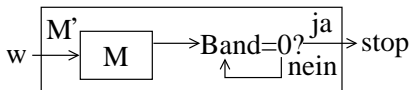
Beweis:

Angenommen, K sei entscheidbar, dh χ_K ist berechenbar.

Dann ist auch folgende Funktion f berechenbar:

$$f(w) := \begin{cases} 0 & \text{falls } \chi_K(w) = 0 \\ \perp & \text{falls } \chi_K(w) = 1 \end{cases}$$

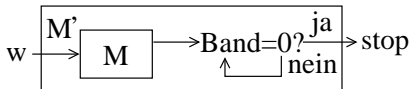
In der Tat, berechnet eine TM M die Funktion χ_K so berechnet die folgende TM M' die Funktion f :



Dann gibt es ein w' mit $M_{w'} = M'$.

Beweis (Forts.):

(Forts.)



$$\begin{aligned} f(w') = \perp &\Leftrightarrow \chi_K(w') = 1 && \text{(Def. von } f) \\ &\Leftrightarrow w' \in K && \text{(Def. von } \chi_K) \\ &\Leftrightarrow M_{w'}[w'] \downarrow && \text{(Def. von } K) \\ &\Leftrightarrow M'[w'] \downarrow && (M_{w'} = M') \\ &\Leftrightarrow f(w') \neq \perp && (M' \text{ berechnet } f) \end{aligned}$$

Wir erhalten einen Widerspruch: $f(w') = \perp \Leftrightarrow f(w') \neq \perp$.
Die Annahme, K sei entscheidbar, ist damit falsch.



Definition 5.34 ((Allgemeines) Halteproblem)

Gegeben: Wörter $w, x \in \{0, 1\}^*$.

Problem: Hält M_w bei Eingabe x ?

Als Menge:

$$H := \{w\#x \mid M_w[x]\downarrow\}$$

Satz 5.35

Das Halteproblem H ist nicht entscheidbar.

Beweis:

Wäre H entscheidbar, dann trivialerweise auch K :

$$\chi_K(w) = \chi_H(w, w)$$



Definition 5.36 (Reduktion)

Eine Menge $A \subseteq \Sigma^*$ ist **reduzierbar** auf eine Menge $B \subseteq \Gamma^*$ gdw es eine totale und berechenbare Funktion $f : \Sigma^* \rightarrow \Gamma^*$ gibt mit

$$\forall w \in \Sigma^*. w \in A \Leftrightarrow f(w) \in B$$

Wir schreiben dann $A \leq B$.

Definition 5.36 (Reduktion)

Eine Menge $A \subseteq \Sigma^*$ ist **reduzierbar** auf eine Menge $B \subseteq \Gamma^*$ gdw es eine totale und berechenbare Funktion $f : \Sigma^* \rightarrow \Gamma^*$ gibt mit

$$\forall w \in \Sigma^*. w \in A \Leftrightarrow f(w) \in B$$

Wir schreiben dann $A \leq B$.

Intuition:

- B ist mindestens so schwer zu lösen wie A .
- Ist A unlösbar, dann auch B .
- Ist B lösbar, dann erst recht A .

Lemma 5.37

Falls $A \leq B$ und B ist entscheidbar, so ist auch A entscheidbar.

Lemma 5.37

Falls $A \leq B$ und B ist entscheidbar, so ist auch A entscheidbar.

Beweis:

Es gelte $A \leq B$ mittels f und χ_B sei berechenbar.

Dann ist $\chi_B \circ f$ berechenbar und $\chi_A = \chi_B \circ f$:

$$\chi_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} = \begin{cases} 1, & f(x) \in B \\ 0, & f(x) \notin B \end{cases} = \chi_B(f(x)) \quad \square$$

Lemma 5.37

Falls $A \leq B$ und B ist entscheidbar, so ist auch A entscheidbar.

Beweis:

Es gelte $A \leq B$ mittels f und χ_B sei berechenbar.

Dann ist $\chi_B \circ f$ berechenbar und $\chi_A = \chi_B \circ f$:

$$\chi_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} = \begin{cases} 1, & f(x) \in B \\ 0, & f(x) \notin B \end{cases} = \chi_B(f(x)) \quad \square$$

Korollar 5.38

Falls $A \leq B$ und A ist unentscheidbar, dann ist auch B unentscheidbar.

Lemma 5.37

Falls $A \leq B$ und B ist entscheidbar, so ist auch A entscheidbar.

Beweis:

Es gelte $A \leq B$ mittels f und χ_B sei berechenbar.

Dann ist $\chi_B \circ f$ berechenbar und $\chi_A = \chi_B \circ f$:

$$\chi_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} = \begin{cases} 1, & f(x) \in B \\ 0, & f(x) \notin B \end{cases} = \chi_B(f(x)) \quad \square$$

Korollar 5.38

Falls $A \leq B$ und A ist unentscheidbar, dann ist auch B unentscheidbar.

Beispiel 5.39

Da $K \leq H$ (mit Reduktion $f(w) := w\#w$) und K unentscheidbar ist, ist auch H unentscheidbar.

Satz 5.40

Das *Halteproblem auf leerem Band*, H_0 , ist unentscheidbar.

$$H_0 := \{w \in \{0, 1\}^* \mid M_w[\epsilon] \downarrow\}$$

Satz 5.40

Das Halteproblem auf leerem Band, H_0 , ist unentscheidbar.

$$H_0 := \{w \in \{0, 1\}^* \mid M_w[\epsilon] \downarrow\}$$

Beweis:

Wir zeigen $K \leq H_0$ mit einer Funktion $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$
 $f(w)$ ist die Kodierung folgender TM:

Überschreibe die Eingabe mit w ; führe M_w aus.

Dh f berechnet aus w die Kodierung w_1 einer TM, die w schreibt,
und gibt die Kodierung von " $w_1; w$ " zurück.

Damit ist f total und berechenbar.

Satz 5.40

Das Halteproblem auf leerem Band, H_0 , ist unentscheidbar.

$$H_0 := \{w \in \{0, 1\}^* \mid M_w[\epsilon] \downarrow\}$$

Beweis:

Wir zeigen $K \leq H_0$ mit einer Funktion $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$
 $f(w)$ ist die Kodierung folgender TM:

Überschreibe die Eingabe mit w ; führe M_w aus.

Dh f berechnet aus w die Kodierung w_1 einer TM, die w schreibt,
und gibt die Kodierung von " $w_1; w$ " zurück.

Damit ist f total und berechenbar.

Es gilt:

$$w \in K \Leftrightarrow M_w[w] \downarrow \Leftrightarrow M_{f(w)}[\epsilon] \downarrow \Leftrightarrow f(w) \in H_0$$



Fazit:

Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Fazit:

Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Die Unentscheidbarkeit vieler Fragen über die Ausführung von Programmen folgt durch Reduktion des Halteproblems:

Fazit:

Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Die Unentscheidbarkeit vieler Fragen über die Ausführung von Programmen folgt durch Reduktion des Halteproblems:

- Kann ein WHILE-Programm mit einer bestimmten Eingabe einen bestimmten Programmpunkt erreichen?

Fazit:

Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Die Unentscheidbarkeit vieler Fragen über die Ausführung von Programmen folgt durch Reduktion des Halteproblems:

- Kann ein WHILE-Programm mit einer bestimmten Eingabe einen bestimmten Programmpunkt erreichen?
Der Spezialfall Programmpunkt=Programmende ist das Halteproblem.

Fazit:

Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Die Unentscheidbarkeit vieler Fragen über die Ausführung von Programmen folgt durch Reduktion des Halteproblems:

- Kann ein WHILE-Programm mit einer bestimmten Eingabe einen bestimmten Programmpunkt erreichen?
Der Spezialfall Programmpunkt=Programmende ist das Halteproblem.
- Kann Variable x_7 bei einer bestimmten Eingabe je den Wert 2^{32} erreichen?

Fazit:

Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Die Unentscheidbarkeit vieler Fragen über die Ausführung von Programmen folgt durch Reduktion des Halteproblems:

- Kann ein WHILE-Programm mit einer bestimmten Eingabe einen bestimmten Programmpunkt erreichen?
Der Spezialfall Programmpunkt=Programmende ist das Halteproblem.
- Kann Variable x_7 bei einer bestimmten Eingabe je den Wert 2^{32} erreichen?
Reduktion: Ein Programm P hält gdw

Fazit:

Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Die Unentscheidbarkeit vieler Fragen über die Ausführung von Programmen folgt durch Reduktion des Halteproblems:

- Kann ein WHILE-Programm mit einer bestimmten Eingabe einen bestimmten Programmpunkt erreichen?
Der Spezialfall Programmpunkt=Programmende ist das Halteproblem.
- Kann Variable x_7 bei einer bestimmten Eingabe je den Wert 2^{32} erreichen?

Reduktion: Ein Programm P hält gdw während der Ausführung von

$$P; x_7 := 2^{32}$$

Variable x_7 den Wert 2^{32} erreicht.

(OE: x_7 kommt in P nicht vor)

5.3 Semi-Entscheidbarkeit

5.3 Semi-Entscheidbarkeit

Definition 5.41

Eine Menge A ($\subseteq \mathbb{N}$ oder Σ^*) heißt **semi-entscheidbar (s-e)** gdw

$$\chi'_A(x) := \begin{cases} 1 & \text{falls } x \in A \\ \perp & \text{falls } x \notin A \end{cases}$$

berechenbar ist.

Satz 5.42

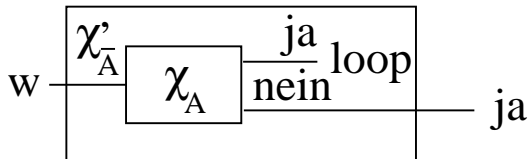
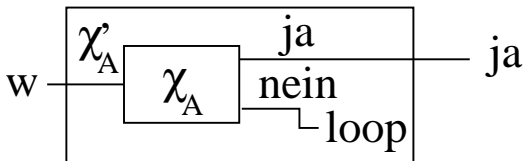
Eine Menge A ist entscheidbar gdw sowohl A als auch \overline{A} s-e sind.

Satz 5.42

Eine Menge A ist entscheidbar gdw sowohl A als auch \bar{A} s-e sind.

Beweis:

„ \Rightarrow “: Wandle TM für χ_A in TM für χ'_A und $\chi'_{\bar{A}}$ um:



Beweis (Forts.):

„ \Leftarrow “:

Wandle TM M_1 für χ'_A und TM M_2 für $\chi'_{\bar{A}}$ in TM für χ_A um:

input(x);

for $s := 0, 1, 2, \dots$ **do**

if $M_1[x]$ hält in s Schritten **then** output(1); **halt fi** ;

if $M_2[x]$ hält in s Schritten **then** output(0); **halt fi**

Beweis (Forts.):

„ \Leftarrow “:

Wandle TM M_1 für χ'_A und TM M_2 für $\chi'_{\bar{A}}$ in TM für χ_A um:

```
input( $x$ );
```

```
for  $s := 0, 1, 2, \dots$  do
```

```
    if  $M_1[x]$  hält in  $s$  Schritten then output(1); halt fi ;
```

```
    if  $M_2[x]$  hält in  $s$  Schritten then output(0); halt fi
```

Formulierung mit Parallelismus:

```
input( $x$ );
```

```
führe  $M_1[x]$  und  $M_2[x]$  parallel aus;
```

```
hält  $M_1$ , gib 1 aus, hält  $M_2$ , gib 0 aus.
```



Beweis (Forts.):

„ \Leftarrow “:

Wandle TM M_1 für χ'_A und TM M_2 für $\chi'_{\bar{A}}$ in TM für χ_A um:

input(x);

for $s := 0, 1, 2, \dots$ **do**

if $M_1[x]$ hält in s Schritten **then** output(1); **halt fi** ;

if $M_2[x]$ hält in s Schritten **then** output(0); **halt fi**

Formulierung mit Parallelismus:

input(x);

führe $M_1[x]$ und $M_2[x]$ parallel aus;

hält M_1 , gib 1 aus, hält M_2 , gib 0 aus. □

Lemma 5.43

Ist $A \leq B$ und ist B s-e, so ist auch A s-e.

Beweis: Übung

Definition 5.44

Eine Menge A heißt **rekursiv aufzählbar** (*recursively enumerable*)
gdw $A = \emptyset$ oder es eine berechenbare totale Funktion $f : \mathbb{N} \rightarrow A$
gibt, so dass

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

Definition 5.44

Eine Menge A heißt **rekursiv aufzählbar** (*recursively enumerable*) gdw $A = \emptyset$ oder es eine berechenbare totale Funktion $f : \mathbb{N} \rightarrow A$ gibt, so dass

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

Bemerkung:

- Es dürfen Elemente doppelt auftreten ($f(i) = f(j)$ für $i \neq j$)
- Die Reihenfolge ist beliebig.

Definition 5.44

Eine Menge A heißt **rekursiv aufzählbar** (*recursively enumerable*) gdw $A = \emptyset$ oder es eine berechenbare totale Funktion $f : \mathbb{N} \rightarrow A$ gibt, so dass

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

Bemerkung:

- Es dürfen Elemente doppelt auftreten ($f(i) = f(j)$ für $i \neq j$)
- Die Reihenfolge ist beliebig.

Warnung: Rekursiv aufzählbar \neq abzählbar!

- Jede Sprache ist abzählbar aber nicht jede Sprache ist rekursiv aufzählbar (s.u.)

Lemma 5.45

Eine Menge A ist rekursiv aufzählbar gdw sie semi-entscheidbar ist.

Lemma 5.45

Eine Menge A ist rekursiv aufzählbar gdw sie semi-entscheidbar ist.

Beweis:

Der Fall $A = \emptyset$ ist trivial. Sei $A \neq \emptyset$.

Lemma 5.45

Eine Menge A ist rekursiv aufzählbar gdw sie semi-entscheidbar ist.

Beweis:

Der Fall $A = \emptyset$ ist trivial. Sei $A \neq \emptyset$.

„ \Rightarrow “: Sei A rekursiv aufzählbar mit f . Dann ist A semi-entscheidbar:

```
input( $x$ );  
for  $i := 0, 1, 2, \dots$  do  
    if  $f(i) = x$  then output(1); halt fi
```

Lemma 5.45

Eine Menge A ist rekursiv aufzählbar gdw sie semi-entscheidbar ist.

Beweis:

Der Fall $A = \emptyset$ ist trivial. Sei $A \neq \emptyset$.

„ \Rightarrow “: Sei A rekursiv aufzählbar mit f . Dann ist A semi-entscheidbar:

```
input( $x$ );  
for  $i := 0, 1, 2, \dots$  do  
    if  $f(i) = x$  then output(1); halt fi
```

„ \Leftarrow “: O.B.d.A. nehmen wir $A \subseteq \mathbb{N}$ an.

Sei A semi-entscheidbar durch (zB) GOTO-Programm P .

Problem: $P[i]$ muss nicht halten und darf daher nur

„zeitbeschränkt“ ausgeführt werden. Gesucht: Paare (i, j) so dass $P[i]$ nach j Schritten hält.

Beweis (Forts.):

Idee: Wir benutzen eine geeignete Bijektion $c: \mathbb{N} \times \mathbb{N} \leftrightarrow \mathbb{N}$.

Seien $p_1: \mathbb{N} \rightarrow \mathbb{N}$ und $p_2: \mathbb{N} \rightarrow \mathbb{N}$ mit

$$p_1(c(n_1, n_2)) = n_1 \quad \text{und} \quad p_2(c(n_1, n_2)) = n_2$$

(Umkehrung von c).

Sei $d \in A$ beliebig.

Folgender Algorithmus berechnet eine Aufzählung von A :

input(n);

if $P[p_1(n)]$ hält nach $p_2(n)$ Schritten **then** output($p_1(n)$)

else output(d) **fi**

Beweis (Forts.):

Idee: Wir benutzen eine geeignete Bijektion $c: \mathbb{N} \times \mathbb{N} \leftrightarrow \mathbb{N}$.

Seien $p_1: \mathbb{N} \rightarrow \mathbb{N}$ und $p_2: \mathbb{N} \rightarrow \mathbb{N}$ mit

$$p_1(c(n_1, n_2)) = n_1 \quad \text{und} \quad p_2(c(n_1, n_2)) = n_2$$

(Umkehrung von c).

Sei $d \in A$ beliebig.

Folgender Algorithmus berechnet eine Aufzählung von A :

input(n);

if $P[p_1(n)]$ hält nach $p_2(n)$ Schritten **then** output($p_1(n)$)

else output(d) **fi**

Korrektheit: Der Algorithmus hält immer und liefert immer ein Element aus A .

Beweis (Forts.):

Idee: Wir benutzen eine geeignete Bijektion $c: \mathbb{N} \times \mathbb{N} \leftrightarrow \mathbb{N}$.

Seien $p_1: \mathbb{N} \rightarrow \mathbb{N}$ und $p_2: \mathbb{N} \rightarrow \mathbb{N}$ mit

$$p_1(c(n_1, n_2)) = n_1 \quad \text{und} \quad p_2(c(n_1, n_2)) = n_2$$

(Umkehrung von c).

Sei $d \in A$ beliebig.

Folgender Algorithmus berechnet eine Aufzählung von A :

input(n);

if $P[p_1(n)]$ hält nach $p_2(n)$ Schritten **then** output($p_1(n)$)

else output(d) **fi**

Korrektheit: Der Algorithmus hält immer und liefert immer ein Element aus A .

Vollständigkeit: Sei $a \in A \subseteq \mathbb{N}$.

Dann hält $P[a]$ nach einer endlichen Zahl k von Schritten. Dann liefert die Eingabe $n = c(a, k)$ die Ausgabe a . □

Folgende Aussagen sind äquivalent:

- A ist semi-entscheidbar

Folgende Aussagen sind äquivalent:

- A ist semi-entscheidbar
- A ist rekursiv aufzählbar

Folgende Aussagen sind äquivalent:

- A ist semi-entscheidbar
- A ist rekursiv aufzählbar
- χ'_A ist berechenbar

Folgende Aussagen sind äquivalent:

- A ist semi-entscheidbar
- A ist rekursiv aufzählbar
- χ'_A ist berechenbar
- $A = L(M)$ für eine TM M

Folgende Aussagen sind äquivalent:

- A ist semi-entscheidbar
- A ist rekursiv aufzählbar
- χ'_A ist berechenbar
- $A = L(M)$ für eine TM M
- A ist Definitionsbereich einer berechenbaren Funktion

Folgende Aussagen sind äquivalent:

- A ist semi-entscheidbar
- A ist rekursiv aufzählbar
- χ'_A ist berechenbar
- $A = L(M)$ für eine TM M
- A ist Definitionsbereich einer berechenbaren Funktion
- A ist Wertebereich einer berechenbaren Funktion

Satz 5.46

Die Menge $K = \{w \mid M_w[w] \downarrow\}$ ist semi-entscheidbar.

Satz 5.46

Die Menge $K = \{w \mid M_w[w] \downarrow\}$ ist semi-entscheidbar.

Beweis:

Die Funktion χ'_K ist wie folgt Turing-berechenbar:

Bei Eingabe w simuliere die Ausführung von $M_w[w]$;
gib 1 aus. □

- Hier haben wir benutzt, dass man einen Interpreter/Simulator für Turingmaschinen als Turingmaschine programmieren kann.
- Ein solcher Interpreter wird oft eine **Universelle Turingmaschine** (U) genannt.

Satz 5.46

Die Menge $K = \{w \mid M_w[w] \downarrow\}$ ist semi-entscheidbar.

Beweis:

Die Funktion χ'_K ist wie folgt Turing-berechenbar:

Bei Eingabe w simuliere die Ausführung von $M_w[w]$;
gib 1 aus. □

- Hier haben wir benutzt, dass man einen Interpreter/Simulator für Turingmaschinen als Turingmaschine programmieren kann.
- Ein solcher Interpreter wird oft eine **Universelle Turingmaschine** (U) genannt.

Korollar 5.47

\overline{K} ist nicht semi-entscheidbar.

Satz 5.46

Die Menge $K = \{w \mid M_w[w] \downarrow\}$ ist semi-entscheidbar.

Beweis:

Die Funktion χ'_K ist wie folgt Turing-berechenbar:

Bei Eingabe w simuliere die Ausführung von $M_w[w]$;
gib 1 aus. □

- Hier haben wir benutzt, dass man einen Interpreter/Simulator für Turingmaschinen als Turingmaschine programmieren kann.
- Ein solcher Interpreter wird oft eine **Universelle Turingmaschine** (U) genannt.

Korollar 5.47

\overline{K} ist nicht semi-entscheidbar.

Semi-Entscheidbarkeit ist nicht abgeschlossen unter Komplement.

5.4 Die Sätze von Rice und Shapiro

Die von der TM M_w berechnete Funktion bezeichnen wir mit φ_w .
Wir betrachten implizit nur einstellige Funktionen.

5.4 Die Sätze von Rice und Shapiro

Die von der TM M_w berechnete Funktion bezeichnen wir mit φ_w .
Wir betrachten implizit nur einstellige Funktionen.

Satz 5.48 (Rice)

Sei F eine Menge berechenbarer Funktionen.

Es gelte weder $F = \emptyset$ noch $F =$ alle ber. Funkt. („ F nicht trivial“)

Dann ist unentscheidbar, ob die von einer gegebenen TM M_w berechnete Funktion Element F ist, dh ob $\varphi_w \in F$.

5.4 Die Sätze von Rice und Shapiro

Die von der TM M_w berechnete Funktion bezeichnen wir mit φ_w .
Wir betrachten implizit nur einstellige Funktionen.

Satz 5.48 (Rice)

Sei F eine Menge berechenbarer Funktionen.

Es gelte weder $F = \emptyset$ noch $F =$ alle ber. Funkt. („ F nicht trivial“)

Dann ist unentscheidbar, ob die von einer gegebenen TM M_w berechnete Funktion Element F ist, dh ob $\varphi_w \in F$.

Alle nicht-triviale semantische Eigenschaften von Programmen sind unentscheidbar.

5.4 Die Sätze von Rice und Shapiro

Die von der TM M_w berechnete Funktion bezeichnen wir mit φ_w .
Wir betrachten implizit nur einstellige Funktionen.

Satz 5.48 (Rice)

Sei F eine Menge berechenbarer Funktionen.

Es gelte weder $F = \emptyset$ noch $F =$ alle ber. Funkt. („ F nicht trivial“)

Dann ist unentscheidbar, ob die von einer gegebenen TM M_w berechnete Funktion Element F ist, dh ob $\varphi_w \in F$.

Alle nicht-triviale semantische Eigenschaften von Programmen sind unentscheidbar.

Beispiel 5.49

Es ist unentscheidbar, ob ein Programm

- für mindestens eine Eingabe hält.
($F = \{\varphi_w \mid \exists x. M_w[x] \downarrow\}$)
- für alle Eingaben hält. ($F = \{\varphi_w \mid \forall x. M_w[x] \downarrow\}$)
- bei Eingabe 42 Ausgabe 42 produziert.

Warnung

Es ist entscheidbar, ob ein Programm

- länger als 5 Zeilen ist.
- eine Zuweisung an die Variable x_{17} enthält.

Im Satz von Rice geht es um die von einem Programm berechnete Funktion (Semantik), nicht um den Programmtext (Syntax).

Beweis:

Wir zeigen $C_F := \{w \in \{0, 1\}^* \mid \varphi_w \in F\}$ ist unentscheidbar.

Beweis:

Wir zeigen $C_F := \{w \in \{0, 1\}^* \mid \varphi_w \in F\}$ ist unentscheidbar.

Fall 1: $\Omega := (x \mapsto \perp) \notin F$.

Wähle $h \in F \neq \emptyset$ beliebig; sei u Kodierung einer TM mit $\varphi_u = h$.

Beweis:

Wir zeigen $C_F := \{w \in \{0, 1\}^* \mid \varphi_w \in F\}$ ist unentscheidbar.

Fall 1: $\Omega := (x \mapsto \perp) \notin F$.

Wähle $h \in F \neq \emptyset$ beliebig; sei u Kodierung einer TM mit $\varphi_u = h$.

Reduziere K auf C_F ($K \leq C_F$) mit $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ und $f(w)$ die Kodierung folgender TM:

Speichere die Eingabe x auf einem getrennten Band;
schreibe $w\#w$ auf die Eingabe; führe die universelle TM
 U auf $w\#w$ aus; führe M_u auf x aus.

Beweis:

Wir zeigen $C_F := \{w \in \{0, 1\}^* \mid \varphi_w \in F\}$ ist unentscheidbar.

Fall 1: $\Omega := (x \mapsto \perp) \notin F$.

Wähle $h \in F \neq \emptyset$ beliebig; sei u Kodierung einer TM mit $\varphi_u = h$.

Reduziere K auf C_F ($K \leq C_F$) mit $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ und $f(w)$ die Kodierung folgender TM:

Speichere die Eingabe x auf einem getrennten Band;
schreibe $w\#w$ auf die Eingabe; führe die universelle TM
 U auf $w\#w$ aus; führe M_u auf x aus.

Es gilt $\varphi_{f(w)} = \begin{cases} h & \text{falls } M_w[w] \downarrow \\ \Omega & \text{sonst} \end{cases}$ und damit

$$w \in K \Leftrightarrow M_w[w] \downarrow \stackrel{(*)}{\Leftrightarrow} \varphi_{f(w)} \in F \Leftrightarrow f(w) \in C_F$$

Beweis:

Wir zeigen $C_F := \{w \in \{0, 1\}^* \mid \varphi_w \in F\}$ ist unentscheidbar.

Fall 1: $\Omega := (x \mapsto \perp) \notin F$.

Wähle $h \in F \neq \emptyset$ beliebig; sei u Kodierung einer TM mit $\varphi_u = h$.

Reduziere K auf C_F ($K \leq C_F$) mit $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ und $f(w)$ die Kodierung folgender TM:

Speichere die Eingabe x auf einem getrennten Band;
schreibe $w\#w$ auf die Eingabe; führe die universelle TM
 U auf $w\#w$ aus; führe M_u auf x aus.

Es gilt $\varphi_{f(w)} = \begin{cases} h & \text{falls } M_w[w] \downarrow \\ \Omega & \text{sonst} \end{cases}$ und damit

$$w \in K \Leftrightarrow M_w[w] \downarrow \stackrel{(*)}{\Leftrightarrow} \varphi_{f(w)} \in F \Leftrightarrow f(w) \in C_F$$

$$(*) : \begin{cases} M_w[w] \downarrow \Rightarrow \varphi_{f(w)} = h \in F \\ \varphi_{f(w)} \in F \Rightarrow \varphi_{f(w)} = h \Rightarrow M_w[w] \downarrow \end{cases}$$

Beweis (Forts.):

Fall 2: $\Omega \in F$.

Wähle berechenbares $h \notin F$.

Zeige analog, dass $\overline{K} \leq C_F$.



Satz 5.50 (Rice-Shapiro)

Sei F eine Menge berechenbarer Funktionen.

Ist $C_F := \{w \mid \varphi_w \in F\}$ semi-entscheidbar,

so gilt für alle berechenbaren f :

$f \in F \Leftrightarrow$ es gibt eine endliche Teilfunktion $g \subseteq f$ mit $g \in F$.

Satz 5.50 (Rice-Shapiro)

Sei F eine Menge berechenbarer Funktionen.

Ist $C_F := \{w \mid \varphi_w \in F\}$ semi-entscheidbar,

so gilt für alle berechenbaren f :

$f \in F \Leftrightarrow$ es gibt eine endliche Teilfunktion $g \subseteq f$ mit $g \in F$.

Beweis:

„ \Rightarrow “ mit Widerspruch.

Sei $f \in F$, so dass für alle endlichen $g \subseteq f$ gilt $g \notin F$.

Wir zeigen $\overline{K} \leq C_F$ womit C_F nicht semi-entscheidbar ist. ⚡

Beweis (Forts.):

Reduktion $\overline{K} \leq C_F$ mit $h : \{0, 1\}^* \rightarrow \{0, 1\}^*$:

$h(w)$ ist die Kodierung folgender TM:

Beweis (Forts.):

Reduktion $\overline{K} \leq C_F$ mit $h : \{0, 1\}^* \rightarrow \{0, 1\}^*$:

$h(w)$ ist die Kodierung folgender TM:

Bei Eingabe t simuliere t Schritte von $M_w[w]$.

Hält diese Berechnung in $\leq t$ Schritten, gehe in eine endlos Schleife,
sonst berechne $f(t)$.

Beweis (Forts.):

Reduktion $\overline{K} \leq C_F$ mit $h : \{0, 1\}^* \rightarrow \{0, 1\}^*$:

$h(w)$ ist die Kodierung folgender TM:

Bei Eingabe t simuliere t Schritte von $M_w[w]$.

Hält diese Berechnung in $\leq t$ Schritten, gehe in eine endlos Schleife, sonst berechne $f(t)$.

Wir zeigen

$$w \in \overline{K} \Leftrightarrow h(w) \in C_F$$

Beweis (Forts.):

Reduktion $\overline{K} \leq C_F$ mit $h : \{0, 1\}^* \rightarrow \{0, 1\}^*$:

$h(w)$ ist die Kodierung folgender TM:

Bei Eingabe t simulierte t Schritte von $M_w[w]$.

Hält diese Berechnung in $\leq t$ Schritten, gehe in eine endlos Schleife, sonst berechne $f(t)$.

Wir zeigen

$$w \in \overline{K} \Leftrightarrow h(w) \in C_F$$

- $w \in \overline{K} \implies \neg M_w[w] \downarrow \implies \varphi_{h(w)} = f \in F \implies h(w) \in C_F$

Beweis (Forts.):

Reduktion $\overline{K} \leq C_F$ mit $h : \{0, 1\}^* \rightarrow \{0, 1\}^*$:

$h(w)$ ist die Kodierung folgender TM:

Bei Eingabe t simuliere t Schritte von $M_w[w]$.

Hält diese Berechnung in $\leq t$ Schritten, gehe in eine endlos Schleife, sonst berechne $f(t)$.

Wir zeigen

$$w \in \overline{K} \Leftrightarrow h(w) \in C_F$$

- $w \in \overline{K} \implies \neg M_w[w] \downarrow \implies \varphi_{h(w)} = f \in F \implies h(w) \in C_F$
- Falls $w \notin \overline{K}$ dann hält $M_w[w]$ nach eine Zahl t von Schritten. Damit gilt: $\varphi_{h(w)}$ ist f eingeschränkt auf $\{0, \dots, t-1\}$. Nach Annahme folgt $\varphi_{h(w)} \notin F$, dh $h(w) \notin C_F$.

Beweis (Forts.):

„ \Leftarrow “ mit Widerspruch.

Beweis (Forts.):

„ \Leftarrow “ mit Widerspruch.

Sei f berechenbar, sei $g \subseteq f$ endlich mit $g \in F$, aber sei $f \notin F$.

Wir zeigen $\overline{K} \leq C_F$ womit C_F nicht semi-entscheidbar ist. ⚡

Reduktion $\overline{K} \leq C_F$ mit $h : \{0, 1\}^* \rightarrow \{0, 1\}^*$:

$h(w)$ ist die Kodierung folgender TM:

Beweis (Forts.):

„ \Leftarrow “ mit Widerspruch.

Sei f berechenbar, sei $g \subseteq f$ endlich mit $g \in F$, aber sei $f \notin F$.

Wir zeigen $\overline{K} \leq C_F$ womit C_F nicht semi-entscheidbar ist. ⚡

Reduktion $\overline{K} \leq C_F$ mit $h : \{0, 1\}^* \rightarrow \{0, 1\}^*$:

$h(w)$ ist die Kodierung folgender TM:

Bei Eingabe t , teste ob t im *endlichen* Def. ber. von g ist.

Wenn ja, berechne $f(t)$,

sonst simuliere $M_w[w]$ und berechne dann $f(t)$.

Beweis (Forts.):

„ \Leftarrow “ mit Widerspruch.

Sei f berechenbar, sei $g \subseteq f$ endlich mit $g \in F$, aber sei $f \notin F$.

Wir zeigen $\overline{K} \leq C_F$ womit C_F nicht semi-entscheidbar ist. ⚡

Reduktion $\overline{K} \leq C_F$ mit $h : \{0, 1\}^* \rightarrow \{0, 1\}^*$:

$h(w)$ ist die Kodierung folgender TM:

Bei Eingabe t , teste ob t im *endlichen* Def. ber. von g ist.

Wenn ja, berechne $f(t)$,

sonst simuliere $M_w[w]$ und berechne dann $f(t)$.

Wir zeigen

$$w \in \overline{K} \Leftrightarrow h(w) \in C_F$$

Beweis (Forts.):

„ \Leftarrow “ mit Widerspruch.

Sei f berechenbar, sei $g \subseteq f$ endlich mit $g \in F$, aber sei $f \notin F$.

Wir zeigen $\overline{K} \leq C_F$ womit C_F nicht semi-entscheidbar ist. ⚡

Reduktion $\overline{K} \leq C_F$ mit $h : \{0, 1\}^* \rightarrow \{0, 1\}^*$:

$h(w)$ ist die Kodierung folgender TM:

Bei Eingabe t , teste ob t im *endlichen* Def. ber. von g ist.

Wenn ja, berechne $f(t)$,

sonst simuliere $M_w[w]$ und berechne dann $f(t)$.

Wir zeigen

$$w \in \overline{K} \Leftrightarrow h(w) \in C_F$$

- $w \in \overline{K} \implies \neg M_w[w] \downarrow \implies \varphi_{h(w)} = g \in F \implies h(w) \in C_F$
- $w \notin \overline{K} \implies M_w[w] \downarrow \implies \varphi_{h(w)} = f \notin F \implies h(w) \notin C_F$



Rice-Shapiro (in Kurzform): $C_F := \{w \mid \varphi_w \in F\}$ s-e \implies
 $f \in F \iff$ es gibt endliche Funkt. $g \subseteq f$ mit $g \in F$.

Rice-Shapiro (in Kurzform): $C_F := \{w \mid \varphi_w \in F\}$ s-e \implies
 $f \in F \Leftrightarrow$ es gibt endliche Funkt. $g \subseteq f$ mit $g \in F$.

Ein Programm heißt **terminierend** gdw es für alle Eingaben hält.

Korollar 5.51

- *Die Menge der terminierenden Programme ist nicht semi-entscheidbar.*
- *Die Menge der nicht-terminierenden Programme ist nicht semi-entscheidbar.*

Rice-Shapiro (in Kurzform): $C_F := \{w \mid \varphi_w \in F\}$ s-e \implies
 $f \in F \Leftrightarrow$ es gibt endliche Funkt. $g \subseteq f$ mit $g \in F$.

Ein Programm heißt **terminierend** gdw es für alle Eingaben hält.

Korollar 5.51

- Die Menge der terminierenden Programme ist nicht semi-entscheidbar.
- Die Menge der nicht-terminierenden Programme ist nicht semi-entscheidbar.

Beweis:

- $F :=$ Menge aller berechenbaren totalen Funktionen.
Sei $f \in F$. Jede endliche $g \subseteq f$ ist echt partiell, dh $g \notin F$.
Also kann C_F nicht semi-entscheidbar sein.

Rice-Shapiro (in Kurzform): $C_F := \{w \mid \varphi_w \in F\}$ s-e \implies
 $f \in F \Leftrightarrow$ es gibt endliche Funkt. $g \subseteq f$ mit $g \in F$.

Ein Programm heißt **terminierend** gdw es für alle Eingaben hält.

Korollar 5.51

- Die Menge der terminierenden Programme ist nicht semi-entscheidbar.
- Die Menge der nicht-terminierenden Programme ist nicht semi-entscheidbar.

Beweis:

- $F :=$ Menge aller berechenbaren totalen Funktionen.
Sei $f \in F$. Jede endliche $g \subseteq f$ ist echt partiell, dh $g \notin F$.
Also kann C_F nicht semi-entscheidbar sein.
- $F :=$ Menge aller berechenbaren nicht-totalen Funktionen.
Sei f total und berechenbar. Damit $f \notin F$.
Aber jede endliche $g \subseteq f$ ist in F .
Also kann C_F nicht semi-entscheidbar sein. \square

Grenzen automatischer Terminationsanalyse von Programmen

Grenzen automatischer Terminationsanalyse von Programmen

- Termination ist unentscheidbar (Rice):
Klare Ja/Nein Antwort unmöglich.

Grenzen automatischer Terminationsanalyse von Programmen

- Termination ist unentscheidbar (Rice):
Klare Ja/Nein Antwort unmöglich.
- Termination ist nicht semi-entscheidbar (Rice-Shapiro):
Es gibt kein Zertifizierungs-Programm,
das alle terminierenden Programme erkennt.

Grenzen automatischer Terminationsanalyse von Programmen

- Termination ist unentscheidbar (Rice):
Klare Ja/Nein Antwort unmöglich.
- Termination ist nicht semi-entscheidbar (Rice-Shapiro):
Es gibt kein Zertifizierungs-Programm,
das alle terminierenden Programme erkennt.
- Nicht-Termination ist nicht semi-entscheidbar (Rice-Shapiro):
Es gibt keinen perfekten *Bug Finder*,
der alle nicht-terminierenden Programme erkennt.

Grenzen automatischer Terminationsanalyse von Programmen

- Termination ist unentscheidbar (Rice):
Klare Ja/Nein Antwort unmöglich.
- Termination ist nicht semi-entscheidbar (Rice-Shapiro):
Es gibt kein Zertifizierungs-Programm,
das alle terminierenden Programme erkennt.
- Nicht-Termination ist nicht semi-entscheidbar (Rice-Shapiro):
Es gibt keinen perfekten *Bug Finder*,
der alle nicht-terminierenden Programme erkennt.

Aber es gibt mächtige heuristische Verfahren, die für relativ viele Programme aus der Praxis (Gerätetreiber)

- Termination beweisen können, oder
- Gegenbeispiele finden können.

5.5 Das Postsche Korrespondenzproblem

Gegeben beliebig viele Kopien der 3 „Spielkarten“

001	10	0
00	11	010

5.5 Das Postsche Korrespondenzproblem

Gegeben beliebig viele Kopien der 3 „Spielkarten“

001	10	0
00	11	010

gibt es dann eine Folge dieser Karten

...	...
...	...

so dass oben und unten das gleiche Wort steht?

5.5 Das Postsche Korrespondenzproblem

Gegeben beliebig viele Kopien der 3 „Spielkarten“

001	10	0
00	11	010

gibt es dann eine Folge dieser Karten

...	...
...	...

so dass oben und unten das gleiche Wort steht?

001	10	001	0
00	11	00	010

Kurz: 1,2,1,3.

Definition 5.52 (Postsche Korrespondenzproblem, *Post's Correspondence Problem*, PCP)

Gegeben: Eine endliche Folge $(x_1, y_1), \dots, (x_k, y_k)$, wobei $x_i, y_i \in \Sigma^+$.

Problem: Gibt es eine Folge von Indizes $i_1, \dots, i_n \in \{1, \dots, k\}$, $n > 0$, mit $x_{i_1} \dots x_{i_n} = y_{i_1} \dots y_{i_n}$?

Dann nennen wir i_1, \dots, i_n eine **Lösung** des Problems $(x_1, y_1), \dots, (x_k, y_k)$.

Definition 5.52 (Postsche Korrespondenzproblem, *Post's Correspondence Problem, PCP*)

Gegeben: Eine endliche Folge $(x_1, y_1), \dots, (x_k, y_k)$, wobei $x_i, y_i \in \Sigma^+$.

Problem: Gibt es eine Folge von Indizes $i_1, \dots, i_n \in \{1, \dots, k\}$, $n > 0$, mit $x_{i_1} \dots x_{i_n} = y_{i_1} \dots y_{i_n}$?

Dann nennen wir i_1, \dots, i_n eine **Lösung** des Problems $(x_1, y_1), \dots, (x_k, y_k)$.

Beispiel 5.53

- Hat $(1, 111), (10111, 10), (10, 0)$ eine Lösung? 2,1,1,3

Definition 5.52 (Postsche Korrespondenzproblem, *Post's Correspondence Problem*, PCP)

Gegeben: Eine endliche Folge $(x_1, y_1), \dots, (x_k, y_k)$, wobei $x_i, y_i \in \Sigma^+$.

Problem: Gibt es eine Folge von Indizes $i_1, \dots, i_n \in \{1, \dots, k\}$, $n > 0$, mit $x_{i_1} \dots x_{i_n} = y_{i_1} \dots y_{i_n}$?

Dann nennen wir i_1, \dots, i_n eine **Lösung** des Problems $(x_1, y_1), \dots, (x_k, y_k)$.

Beispiel 5.53

- Hat $(1, 111), (10111, 10), (10, 0)$ eine Lösung? 2,1,1,3
- Hat $(b, ca), (a, ab), (ca, a), (abc, c)$ eine Lösung? 2,1,3,2,4

Definition 5.52 (Postsche Korrespondenzproblem, *Post's Correspondence Problem, PCP*)

Gegeben: Eine endliche Folge $(x_1, y_1), \dots, (x_k, y_k)$, wobei $x_i, y_i \in \Sigma^+$.

Problem: Gibt es eine Folge von Indizes $i_1, \dots, i_n \in \{1, \dots, k\}$, $n > 0$, mit $x_{i_1} \dots x_{i_n} = y_{i_1} \dots y_{i_n}$?

Dann nennen wir i_1, \dots, i_n eine **Lösung** des Problems $(x_1, y_1), \dots, (x_k, y_k)$.

Beispiel 5.53

- Hat $(1, 111), (10111, 10), (10, 0)$ eine Lösung? 2,1,1,3
- Hat $(b, ca), (a, ab), (ca, a), (abc, c)$ eine Lösung? 2,1,3,2,4
- Hat $(101, 01), (101, 010), (010, 10)$ eine Lösung? Nein!

Definition 5.52 (Postsche Korrespondenzproblem, *Post's Correspondence Problem*, PCP)

Gegeben: Eine endliche Folge $(x_1, y_1), \dots, (x_k, y_k)$, wobei $x_i, y_i \in \Sigma^+$.

Problem: Gibt es eine Folge von Indizes $i_1, \dots, i_n \in \{1, \dots, k\}$, $n > 0$, mit $x_{i_1} \dots x_{i_n} = y_{i_1} \dots y_{i_n}$?

Dann nennen wir i_1, \dots, i_n eine **Lösung** des Problems $(x_1, y_1), \dots, (x_k, y_k)$.

Beispiel 5.53

- Hat $(1, 111), (10111, 10), (10, 0)$ eine Lösung? 2,1,1,3
- Hat $(b, ca), (a, ab), (ca, a), (abc, c)$ eine Lösung? 2,1,3,2,4
- Hat $(101, 01), (101, 010), (010, 10)$ eine Lösung? Nein!
- Hat $(10, 101), (011, 11), (101, 011)$ eine Lösung? [HMU]

Definition 5.52 (Postsche Korrespondenzproblem, *Post's Correspondence Problem*, PCP)

Gegeben: Eine endliche Folge $(x_1, y_1), \dots, (x_k, y_k)$, wobei $x_i, y_i \in \Sigma^+$.

Problem: Gibt es eine Folge von Indizes $i_1, \dots, i_n \in \{1, \dots, k\}$, $n > 0$, mit $x_{i_1} \dots x_{i_n} = y_{i_1} \dots y_{i_n}$?

Dann nennen wir i_1, \dots, i_n eine **Lösung** des Problems $(x_1, y_1), \dots, (x_k, y_k)$.

Beispiel 5.53

- Hat $(1, 111), (10111, 10), (10, 0)$ eine Lösung? 2,1,1,3
- Hat $(b, ca), (a, ab), (ca, a), (abc, c)$ eine Lösung? 2,1,3,2,4
- Hat $(101, 01), (101, 010), (010, 10)$ eine Lösung? Nein!
- Hat $(10, 101), (011, 11), (101, 011)$ eine Lösung? [HMU]
- Hat $(1000, 10), (1, 0011), (0, 111), (11, 0)$ eine Lösung?
Ja, mit Länge 495.



Emil Post.

A Variant of a Recursively Unsolvable Problem.

Bulletin American Mathematical Society, 1946.

Emil Leon Post, 1897 (Polen) – 1954 (NY).



Lemma 5.54

Das PCP ist semi-entscheidbar.

Lemma 5.54

Das PCP ist semi-entscheidbar.

Beweis:

Zähle die möglichen Lösungen der Länge nach auf, und probiere jeweils, ob es eine wirkliche Lösung ist. □

Lemma 5.54

Das PCP ist semi-entscheidbar.

Beweis:

Zähle die möglichen Lösungen der Länge nach auf, und probiere jeweils, ob es eine wirkliche Lösung ist. □

Wir zeigen nun:

$$H \leq MPCP \leq PCP$$

wobei

Definition 5.55 (Modifiziertes PCP, MPCP)

Gegeben: wie beim PCP

Problem: Gibt es eine Lösung i_1, \dots, i_n mit $i_1 = 1$?

Satz 5.56

$$MPCP \leq PCP$$

Beweis:

Für $w = a_1 \dots a_n$:

$$\overline{w} := \#a_1\#a_2\#\dots\#a_n\#$$

$$\overleftarrow{w} := a_1\#a_2\#\dots\#a_n\#$$

$$\overrightarrow{w} := \#a_1\#a_2\#\dots\#a_n$$

$$f((x_1, y_1), \dots, (x_k, y_k)) := ((\overline{x_1}, \overrightarrow{y_1}), (\overleftarrow{x_1}, \overleftarrow{y_1}), \dots, (\overleftarrow{x_k}, \overrightarrow{y_k}), (\$, \#\$))$$

Satz 5.57

$$H \leq MPCP$$

Beweis:

- $(\#, \# \square q_0 u \square \#)$
- (a, a) für alle $a \in \Gamma \cup \{\#\}$
- $(qa, q'a')$ falls $\delta(q, a) = (q', a', N)$
 $(qa, a'q')$ falls $\delta(q, a) = (q', a', R)$
 $(bqa, q'ba')$ falls $\delta(q, a) = (q', a', L)$, für alle $b \in \Gamma$
- $(\#, \square \#)$, $(\#, \# \square)$
- (aq, q) , (qa, q) für alle $q \in F, a \in \Gamma$
- $(q\#\#, \#)$ für alle $q \in F$

Aus $H \leq PCP$ folgt direkt

Korollar 5.58

Das PCP ist *unentscheidbar*.

Aus $H \leq PCP$ folgt direkt

Korollar 5.58

Das PCP ist *unentscheidbar*.

Korollar 5.59

Das PCP ist auch für $\Sigma = \{0, 1\}$ unentscheidbar

Beweis:

Wir nennen dies das 01-PCP und zeigen $PCP \leq 01\text{-PCP}$.

Sei $\Sigma = \{a_1, \dots, a_m\}$ das Alphabet des gegebenen PCPs.

Aus $H \leq PCP$ folgt direkt

Korollar 5.58

Das PCP ist *unentscheidbar*.

Korollar 5.59

Das PCP ist auch für $\Sigma = \{0, 1\}$ unentscheidbar

Beweis:

Wir nennen dies das 01-PCP und zeigen $PCP \leq 01\text{-PCP}$.

Sei $\Sigma = \{a_1, \dots, a_m\}$ das Alphabet des gegebenen PCPs.

Abbildung auf ein 01-PCP:

$$\begin{aligned} \widehat{a_j} &:= 01^j \\ \widehat{a_{j_1} \dots a_{j_n}} &:= \widehat{a_{j_1}} \dots \widehat{a_{j_n}} \end{aligned}$$

Aus $H \leq PCP$ folgt direkt

Korollar 5.58

Das PCP ist *unentscheidbar*.

Korollar 5.59

Das PCP ist auch für $\Sigma = \{0, 1\}$ unentscheidbar

Beweis:

Wir nennen dies das 01-PCP und zeigen $PCP \leq 01\text{-PCP}$.

Sei $\Sigma = \{a_1, \dots, a_m\}$ das Alphabet des gegebenen PCPs.

Abbildung auf ein 01-PCP:
$$\begin{aligned} \widehat{a}_j &:= 01^j \\ \widehat{a_{j_1} \dots a_{j_n}} &:= \widehat{a_{j_1}} \dots \widehat{a_{j_n}} \end{aligned}$$

Dann hat $(x_1, y_1), \dots, (x_k, y_k)$ eine Lösung gdw
 $(\widehat{x_1}, \widehat{y_1}), \dots, (\widehat{x_k}, \widehat{y_k})$ eine Lösung hat.

„ \Rightarrow “ klar, „ \Leftarrow “ folgt da $\hat{\cdot} : \Sigma^* \rightarrow \{0, 1\}^*$ injektive ist:

$$\begin{aligned} \widehat{x_{i_1}} \dots \widehat{x_{i_n}} = \widehat{y_{i_1}} \dots \widehat{y_{i_n}} &\implies \\ \widehat{x_{i_1} \dots x_{i_n}} = \widehat{y_{i_1} \dots y_{i_n}} &\implies x_{i_1} \dots x_{i_n} = y_{i_1} \dots y_{i_n} \end{aligned}$$



Bemerkungen

- Das PCP ist entscheidbar falls $|\Sigma| = 1$

Bemerkungen

- Das PCP ist entscheidbar falls $|\Sigma| = 1$
- Das PCP ist entscheidbar falls $k \leq 2$.

Bemerkungen

- Das PCP ist entscheidbar falls $|\Sigma| = 1$
- Das PCP ist entscheidbar falls $k \leq 2$.
- Das PCP ist unentscheidbar falls $k \geq 7$.

Bemerkungen

- Das PCP ist entscheidbar falls $|\Sigma| = 1$
- Das PCP ist entscheidbar falls $k \leq 2$.
- Das PCP ist unentscheidbar falls $k \geq 7$.
- Für $k = 3, \dots, 6$ ist noch offen, ob das PCP unentscheidbar ist.

5.6 Unentscheidbare CFG-Probleme

5.6 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:

5.6 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:

$$L(A) = \emptyset,$$

5.6 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:
 $L(A) = \emptyset$, $L(A) = L(B)$, ...

5.6 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:
 $L(A) = \emptyset$, $L(A) = L(B)$, ...
- Für TMs ist fast nichts entscheidbar:

5.6 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:
 $L(A) = \emptyset$, $L(A) = L(B)$, ...
- Für TMs ist fast nichts entscheidbar:
 $L(M) = \emptyset$,

5.6 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:
 $L(A) = \emptyset, L(A) = L(B), \dots$
- Für TMs ist fast nichts entscheidbar:
 $L(M) = \emptyset, L(M_1) = L(M_2), \dots$

5.6 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:
 $L(A) = \emptyset, L(A) = L(B), \dots$
- Für TMs ist fast nichts entscheidbar:
 $L(M) = \emptyset, L(M_1) = L(M_2), \dots$
- Für CFGs ist manches entscheidbar ($L(G) = \emptyset$),

5.6 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:
 $L(A) = \emptyset$, $L(A) = L(B)$, ...
- Für TMs ist fast nichts entscheidbar:
 $L(M) = \emptyset$, $L(M_1) = L(M_2)$, ...
- Für CFGs ist manches entscheidbar ($L(G) = \emptyset$),
und manches **unentscheidbar**: $L(G_1) = L(G_2)$, ...

5.6 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:
 $L(A) = \emptyset, L(A) = L(B), \dots$
- Für TMs ist fast nichts entscheidbar:
 $L(M) = \emptyset, L(M_1) = L(M_2), \dots$
- Für CFGs ist manches entscheidbar ($L(G) = \emptyset$),
und manches **unentscheidbar**: $L(G_1) = L(G_2), \dots$

Satz 5.60

Für CFGs G_1, G_2 sind folgende Probleme unentscheidbar:

5.6 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:
 $L(A) = \emptyset$, $L(A) = L(B)$, ...
- Für TMs ist fast nichts entscheidbar:
 $L(M) = \emptyset$, $L(M_1) = L(M_2)$, ...
- Für CFGs ist manches entscheidbar ($L(G) = \emptyset$),
und manches **unentscheidbar**: $L(G_1) = L(G_2)$, ...

Satz 5.60

Für CFGs G_1, G_2 sind folgende Probleme unentscheidbar:

- 1 Ist $L(G_1) \cap L(G_2) = \emptyset$?

5.6 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:
 $L(A) = \emptyset$, $L(A) = L(B)$, ...
- Für TMs ist fast nichts entscheidbar:
 $L(M) = \emptyset$, $L(M_1) = L(M_2)$, ...
- Für CFGs ist manches entscheidbar ($L(G) = \emptyset$),
und manches **unentscheidbar**: $L(G_1) = L(G_2)$, ...

Satz 5.60

Für CFGs G_1, G_2 sind folgende Probleme unentscheidbar:

- 1 Ist $L(G_1) \cap L(G_2) = \emptyset$?
- 2 Ist $|L(G_1) \cap L(G_2)| = \infty$?

5.6 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:
 $L(A) = \emptyset, L(A) = L(B), \dots$
- Für TMs ist fast nichts entscheidbar:
 $L(M) = \emptyset, L(M_1) = L(M_2), \dots$
- Für CFGs ist manches entscheidbar ($L(G) = \emptyset$),
und manches **unentscheidbar**: $L(G_1) = L(G_2), \dots$

Satz 5.60

Für CFGs G_1, G_2 sind folgende Probleme unentscheidbar:

- 1 Ist $L(G_1) \cap L(G_2) = \emptyset$?
- 2 Ist $|L(G_1) \cap L(G_2)| = \infty$?
- 3 Ist $L(G_1) \cap L(G_2)$ kontextfrei?

5.6 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:
 $L(A) = \emptyset$, $L(A) = L(B)$, ...
- Für TMs ist fast nichts entscheidbar:
 $L(M) = \emptyset$, $L(M_1) = L(M_2)$, ...
- Für CFGs ist manches entscheidbar ($L(G) = \emptyset$),
und manches **unentscheidbar**: $L(G_1) = L(G_2)$, ...

Satz 5.60

Für CFGs G_1, G_2 sind folgende Probleme unentscheidbar:

- 1 Ist $L(G_1) \cap L(G_2) = \emptyset$?
- 2 Ist $|L(G_1) \cap L(G_2)| = \infty$?
- 3 Ist $L(G_1) \cap L(G_2)$ kontextfrei?
- 4 Ist $L(G_1) \subseteq L(G_2)$?

5.6 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:
 $L(A) = \emptyset$, $L(A) = L(B)$, ...
- Für TMs ist fast nichts entscheidbar:
 $L(M) = \emptyset$, $L(M_1) = L(M_2)$, ...
- Für CFGs ist manches entscheidbar ($L(G) = \emptyset$),
und manches **unentscheidbar**: $L(G_1) = L(G_2)$, ...

Satz 5.60

Für CFGs G_1, G_2 sind folgende Probleme unentscheidbar:

- 1 Ist $L(G_1) \cap L(G_2) = \emptyset$?
- 2 Ist $|L(G_1) \cap L(G_2)| = \infty$?
- 3 Ist $L(G_1) \cap L(G_2)$ kontextfrei?
- 4 Ist $L(G_1) \subseteq L(G_2)$?
- 5 Ist $L(G_1) = L(G_2)$?

Beweis:

Sei $K = (x_1, y_1), \dots, (x_k, y_k)$. Definiere G_1

$$\begin{aligned} S &\rightarrow A\$B \\ A &\rightarrow a_1Ax_1 \mid \dots \mid a_kAx_k \\ A &\rightarrow a_1x_1 \mid \dots \mid a_kx_k \\ B &\rightarrow y_1^R Ba_1 \mid \dots \mid y_k^R Ba_k \\ B &\rightarrow y_1^R a_1 \mid \dots \mid y_k^R a_k \end{aligned}$$

und G_2 :

$$\begin{aligned} S &\rightarrow a_1Sa_1 \mid \dots \mid a_kSa_k \mid T \\ T &\rightarrow 0T0 \mid 1T1 \mid \$ \end{aligned}$$

① K hat keine Lösung gdw. $L(G_1) \cap L(G_2) = \emptyset$.

Beweis (Forts.):

- ② K hat mindestens eine Lösung gdw. $|L(G_1) \cap L(G_2)| = \infty$.
(Warum?)

Beweis (Forts.):

- ② K hat mindestens eine Lösung gdw. $|L(G_1) \cap L(G_2)| = \infty$.
(Warum?)
- ③ Sei $L_1 = \{a^i b^i c^j \mid i, j \geq 0\}$ und $L_2 = \{a^i b^j c^j \mid i, j \geq 0\}$.
Seien G'_1 und G'_2 mit $L(G'_1) = L(G_1) \# L_1$ und
 $L(G'_2) = L(G_2) \# L_2$.
Zeige: K hat keine Lösung gdw. $L(G'_1) \cap L(G'_2)$ kontextfrei.

Beweis (Forts.):

- ② K hat mindestens eine Lösung gdw. $|L(G_1) \cap L(G_2)| = \infty$.
(Warum?)
- ③ Sei $L_1 = \{a^i b^i c^j \mid i, j \geq 0\}$ und $L_2 = \{a^i b^j c^j \mid i, j \geq 0\}$.
Seien G'_1 und G'_2 mit $L(G'_1) = L(G_1) \# L_1$ und
 $L(G'_2) = L(G_2) \# L_2$.
Zeige: K hat keine Lösung gdw. $L(G'_1) \cap L(G'_2)$ kontextfrei.
- ④ Zeige: $\overline{L(G_2)}$ ist kontextfrei. Sei $\overline{G_2}$ CFG mit
 $L(\overline{G_2}) = \overline{L(G_2)}$. Es gilt: K hat keine Lösung gdw.
 $L(G_1) \cap L(\overline{G_2}) = \emptyset$ gdw. $L(G_1) \subseteq L(\overline{G_2})$.

Beweis (Forts.):

- ② K hat mindestens eine Lösung gdw. $|L(G_1) \cap L(G_2)| = \infty$.
(Warum?)
- ③ Sei $L_1 = \{a^i b^j c^k \mid i, j, k \geq 0\}$ und $L_2 = \{a^i b^j c^k \mid i, j, k \geq 0\}$.
Seien G'_1 und G'_2 mit $L(G'_1) = L(G_1) \# L_1$ und
 $L(G'_2) = L(G_2) \# L_2$.
Zeige: K hat keine Lösung gdw. $L(G'_1) \cap L(G'_2)$ kontextfrei.
- ④ Zeige: $\overline{L(G_2)}$ ist kontextfrei. Sei $\overline{G_2}$ CFG mit
 $L(\overline{G_2}) = \overline{L(G_2)}$. Es gilt: K hat keine Lösung gdw.
 $L(G_1) \cap L(\overline{G_2}) = \emptyset$ gdw. $L(G_1) \subseteq L(\overline{G_2})$.
- ⑤ Übung.



Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 *Ist G mehrdeutig?*

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig:

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig: Sei $G_3 := „G_1 \cup G_2“$.

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig: Sei $G_3 := „G_1 \cup G_2“$.

PCP lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset$

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig: Sei $G_3 := „G_1 \cup G_2“$.

PCP lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_3)$ ist mehrdeutig

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig: Sei $G_3 := „G_1 \cup G_2“$.

PCP lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_3)$ ist mehrdeutig

„ \Rightarrow “: Syntaxbäume von G_1 und G_2 disjunkt

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig: Sei $G_3 := „G_1 \cup G_2“$.

PCP lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_3)$ ist mehrdeutig

„ \Rightarrow “: Syntaxbäume von G_1 und G_2 disjunkt

„ \Leftarrow “: G_1 und G_2 sind DCFG und damit nicht mehrdeutig

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig: Sei $G_3 := „G_1 \cup G_2“$.

PCP lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_3)$ ist mehrdeutig

„ \Rightarrow “: Syntaxbäume von G_1 und G_2 disjunkt

„ \Leftarrow “: G_1 und G_2 sind DCFL und damit nicht mehrdeutig

Damit ist $PCP \mapsto G_3$ Reduktion für $PCP \leq$ Mehrdeutigkeit.

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig: Sei $G_3 := „G_1 \cup G_2“$.

PCP lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_3)$ ist mehrdeutig

„ \Rightarrow “: Syntaxbäume von G_1 und G_2 disjunkt

„ \Leftarrow “: G_1 und G_2 sind DCFL und damit nicht mehrdeutig

Damit ist $PCP \mapsto G_3$ Reduktion für $PCP \leq$ Mehrdeutigkeit.

2/3. $L(G)$ regulär/deterministisch:

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig: Sei $G_3 := „G_1 \cup G_2“$.

PCP lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_3)$ ist mehrdeutig

„ \Rightarrow “: Syntaxbäume von G_1 und G_2 disjunkt

„ \Leftarrow “: G_1 und G_2 sind DCFL und damit nicht mehrdeutig

Damit ist $PCP \mapsto G_3$ Reduktion für $PCP \leq$ Mehrdeutigkeit.

2/3. $L(G)$ regulär/deterministisch: Sei $G_4 := „\overline{G_1} \cup \overline{G_2}“$.

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig: Sei $G_3 := „G_1 \cup G_2“$.

PCP lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_3)$ ist mehrdeutig

„ \Rightarrow “: Syntaxbäume von G_1 und G_2 disjunkt

„ \Leftarrow “: G_1 und G_2 sind DCFL und damit nicht mehrdeutig

Damit ist $PCP \mapsto G_3$ Reduktion für $PCP \leq$ Mehrdeutigkeit.

2/3. $L(G)$ regulär/deterministisch: Sei $G_4 := „\overline{G_1} \cup \overline{G_2}“$.

PCP unlösbar

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig: Sei $G_3 := „G_1 \cup G_2“$.

PCP lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_3)$ ist mehrdeutig

„ \Rightarrow “: Syntaxbäume von G_1 und G_2 disjunkt

„ \Leftarrow “: G_1 und G_2 sind DCFG und damit nicht mehrdeutig

Damit ist $PCP \mapsto G_3$ Reduktion für $PCP \leq$ Mehrdeutigkeit.

2/3. $L(G)$ regulär/deterministisch: Sei $G_4 := „\overline{G_1} \cup \overline{G_2}“$.

PCP unlösbar $\Rightarrow L(G_1) \cap L(G_2) = \emptyset$

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig: Sei $G_3 := „G_1 \cup G_2“$.

PCP lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_3)$ ist mehrdeutig

„ \Rightarrow “: Syntaxbäume von G_1 und G_2 disjunkt

„ \Leftarrow “: G_1 und G_2 sind DCFG und damit nicht mehrdeutig

Damit ist $PCP \mapsto G_3$ Reduktion für $PCP \leq$ Mehrdeutigkeit.

2/3. $L(G)$ regulär/deterministisch: Sei $G_4 := „\overline{G_1} \cup \overline{G_2}“$.

PCP unlösbar $\Rightarrow L(G_1) \cap L(G_2) = \emptyset \Rightarrow L(G_4) = \Sigma^*$

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig: Sei $G_3 := „G_1 \cup G_2“$.

PCP lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_3)$ ist mehrdeutig

„ \Rightarrow “: Syntaxbäume von G_1 und G_2 disjunkt

„ \Leftarrow “: G_1 und G_2 sind DCFL und damit nicht mehrdeutig

Damit ist $PCP \mapsto G_3$ Reduktion für $PCP \leq$ Mehrdeutigkeit.

2/3. $L(G)$ regulär/deterministisch: Sei $G_4 := „\overline{G_1} \cup \overline{G_2}“$.

PCP unlösbar $\Rightarrow L(G_1) \cap L(G_2) = \emptyset \Rightarrow L(G_4) = \Sigma^* \Rightarrow L(G_4)$ reg/det.

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig: Sei $G_3 := „G_1 \cup G_2“$.

PCP lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_3)$ ist mehrdeutig

„ \Rightarrow “: Syntaxbäume von G_1 und G_2 disjunkt

„ \Leftarrow “: G_1 und G_2 sind DCFG und damit nicht mehrdeutig

Damit ist $PCP \mapsto G_3$ Reduktion für $PCP \leq$ Mehrdeutigkeit.

2/3. $L(G)$ regulär/deterministisch: Sei $G_4 := „\overline{G_1} \cup \overline{G_2}“$.

PCP unlösbar $\Rightarrow L(G_1) \cap L(G_2) = \emptyset \Rightarrow L(G_4) = \Sigma^* \Rightarrow L(G_4)$ reg/det.

PCP lösbar

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig: Sei $G_3 := „G_1 \cup G_2“$.

PCP lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_3)$ ist mehrdeutig

„ \Rightarrow “: Syntaxbäume von G_1 und G_2 disjunkt

„ \Leftarrow “: G_1 und G_2 sind DCFG und damit nicht mehrdeutig

Damit ist $PCP \mapsto G_3$ Reduktion für $PCP \leq$ Mehrdeutigkeit.

2/3. $L(G)$ regulär/deterministisch: Sei $G_4 := „\overline{G_1} \cup \overline{G_2}“$.

PCP unlösbar $\Rightarrow L(G_1) \cap L(G_2) = \emptyset \Rightarrow L(G_4) = \Sigma^* \Rightarrow L(G_4)$ reg/det.

PCP lösbar $\Rightarrow L(G_1) \cap L(G_2)$ nicht CFL

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig: Sei $G_3 := „G_1 \cup G_2“$.

PCP lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_3)$ ist mehrdeutig

„ \Rightarrow “: Syntaxbäume von G_1 und G_2 disjunkt

„ \Leftarrow “: G_1 und G_2 sind DCFL und damit nicht mehrdeutig

Damit ist $PCP \mapsto G_3$ Reduktion für $PCP \leq$ Mehrdeutigkeit.

2/3. $L(G)$ regulär/deterministisch: Sei $G_4 := „\overline{G_1} \cup \overline{G_2}“$.

PCP unlösbar $\Rightarrow L(G_1) \cap L(G_2) = \emptyset \Rightarrow L(G_4) = \Sigma^* \Rightarrow L(G_4)$ reg/det.

PCP lösbar $\Rightarrow L(G_1) \cap L(G_2)$ nicht CFL $\Rightarrow \overline{L(G_4)}$ nicht reg/det

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig: Sei $G_3 := „G_1 \cup G_2“$.

PCP lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_3)$ ist mehrdeutig

„ \Rightarrow “: Syntaxbäume von G_1 und G_2 disjunkt

„ \Leftarrow “: G_1 und G_2 sind DCFL und damit nicht mehrdeutig

Damit ist $PCP \mapsto G_3$ Reduktion für $PCP \leq$ Mehrdeutigkeit.

2/3. $L(G)$ regulär/deterministisch: Sei $G_4 := „\overline{G_1} \cup \overline{G_2}“$.

PCP unlösbar $\Rightarrow L(G_1) \cap L(G_2) = \emptyset \Rightarrow L(G_4) = \Sigma^* \Rightarrow L(G_4)$ reg/det.

PCP lösbar $\Rightarrow L(G_1) \cap L(G_2)$ nicht CFL $\Rightarrow \overline{L(G_4)}$ nicht reg/det

$\Rightarrow L(G_4)$ nicht reg/det.

Satz 5.61

Für eine CFG G sind folgende Probleme unentscheidbar:

- 1 Ist G mehrdeutig?
- 2 Ist $L(G)$ regulär?
- 3 Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig: Sei $G_3 := „G_1 \cup G_2“$.

PCP lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_3)$ ist mehrdeutig

„ \Rightarrow “: Syntaxbäume von G_1 und G_2 disjunkt

„ \Leftarrow “: G_1 und G_2 sind DCFL und damit nicht mehrdeutig

Damit ist $PCP \mapsto G_3$ Reduktion für $PCP \leq$ Mehrdeutigkeit.

2/3. $L(G)$ regulär/deterministisch: Sei $G_4 := „\overline{G_1} \cup \overline{G_2}“$.

PCP unlösbar $\Rightarrow L(G_1) \cap L(G_2) = \emptyset \Rightarrow L(\overline{G_4}) = \Sigma^* \Rightarrow L(G_4)$ reg/det.

PCP lösbar $\Rightarrow L(G_1) \cap L(G_2)$ nicht CFL $\Rightarrow \overline{L(G_4)}$ nicht reg/det

$\Rightarrow L(G_4)$ nicht reg/det.

Damit ist $PCP \mapsto G_4$ Reduktion für $PCP \leq L(G)$ reg/det. \square

Satz 5.62

Für eine CFG G und einen RE α ist $L(G) = L(\alpha)$ unentscheidbar.

Satz 5.62

Für eine CFG G und einen RE α ist $L(G) = L(\alpha)$ unentscheidbar.

Beweis:

Im Beweis des vorherigen Satzes hatten wir eine Reduktion

$PCP \mapsto G_4$ mit

$$PCP \text{ unlösbar} \Leftrightarrow L(G_4) = \Sigma^*.$$

Satz 5.62

Für eine CFG G und einen RE α ist $L(G) = L(\alpha)$ unentscheidbar.

Beweis:

Im Beweis des vorherigen Satzes hatten wir eine Reduktion

$PCP \mapsto G_4$ mit

$$PCP \text{ unlösbar} \Leftrightarrow L(G_4) = \Sigma^*.$$

Sei $\Sigma = \{a_1, \dots, a_n\}$.

Satz 5.62

Für eine CFG G und einen RE α ist $L(G) = L(\alpha)$ unentscheidbar.

Beweis:

Im Beweis des vorherigen Satzes hatten wir eine Reduktion
 $PCP \mapsto G_4$ mit

$$PCP \text{ unlösbar} \Leftrightarrow L(G_4) = \Sigma^*.$$

Sei $\Sigma = \{a_1, \dots, a_n\}$. Dann reduziert $PCP \mapsto (G_4, (a_1 | \dots | a_n)^*)$
das PCP auf das Problem $L(G) = L(\alpha)$, □

6. Komplexitätstheorie

6. Komplexitätstheorie

- Was ist mit beschränkten Mitteln (Zeit&Platz) berechenbar?
- Wieviel Zeit&Platz braucht man, um ein bestimmtes Problem/Sprache zu entscheiden?

6. Komplexitätstheorie

- Was ist mit beschränkten Mitteln (Zeit&Platz) berechenbar?
- Wieviel Zeit&Platz braucht man, um ein bestimmtes Problem/Sprache zu entscheiden?
- **Komplexität eines Problems**, nicht eines Algorithmus.

6. Komplexitätstheorie

- Was ist mit beschränkten Mitteln (Zeit&Platz) berechenbar?
- Wieviel Zeit&Platz braucht man, um ein bestimmtes Problem/Sprache zu entscheiden?
- **Komplexität eines Problems**, nicht eines Algorithmus.
- **Obere Schranken**: durch Angabe eines Algorithmus

6. Komplexitätstheorie

- Was ist mit beschränkten Mitteln (Zeit&Platz) berechenbar?
- Wieviel Zeit&Platz braucht man, um ein bestimmtes Problem/Sprache zu entscheiden?
- **Komplexität eines Problems**, nicht eines Algorithmus.
- **Obere Schranken**: durch Angabe eines Algorithmus
Bsp: $w \in L(G)$ für CFGs ist in Zeit $O(|w|^3)$ lösbar, mit CYK.

6. Komplexitätstheorie

- Was ist mit beschränkten Mitteln (Zeit&Platz) berechenbar?
- Wieviel Zeit&Platz braucht man, um ein bestimmtes Problem/Sprache zu entscheiden?
- **Komplexität eines Problems**, nicht eines Algorithmus.
- **Obere Schranken**: durch Angabe eines Algorithmus
Bsp: $w \in L(G)$ für CFGs ist in Zeit $O(|w|^3)$ lösbar, mit CYK.
- **Untere Schranken**: schwierig . . .

6. Komplexitätstheorie

- Was ist mit beschränkten Mitteln (Zeit&Platz) berechenbar?
- Wieviel Zeit&Platz braucht man, um ein bestimmtes Problem/Sprache zu entscheiden?
- **Komplexität eines Problems**, nicht eines Algorithmus.
- **Obere Schranken**: durch Angabe eines Algorithmus
Bsp: $w \in L(G)$ für CFGs ist in Zeit $O(|w|^3)$ lösbar, mit CYK.
- **Untere Schranken**: schwierig ...
Bsp: Palindrom-Test auf einer 1-Band TM braucht Zeit $\Theta(n^2)$
Nicht hier.

Polynomielle und exponentielle Komplexität

Taktfrequenz: 1GHz

Zeit	Problemgröße n					
	10	20	30	40	50	60
n	.01 μs	.02 μs	.03 μs	.04 μs	.05 μs	.06 μs
n^2	.1 μs	.4 μs	.9 μs	1.6 μs	2.5 μs	3.6 μs
n^5	100 μs	.003 s	.02 s	.1 s	.3 s	.7 s
2^n	1 μs	.001 s	1 s	18 m	13 T	36 J
3^n	59 μs	3 s	2 T	385 J	$2 \cdot 10^7$ J	10^{12} J

μs =Mikrosek., s=Sek., m=Minute, T=Tag, J=Jahr

Polynomielle und exponentielle Komplexität

Taktfrequenz: 1GHz

Zeit	Problemgröße n					
	10	20	30	40	50	60
n	.01 μs	.02 μs	.03 μs	.04 μs	.05 μs	.06 μs
n^2	.1 μs	.4 μs	.9 μs	1.6 μs	2.5 μs	3.6 μs
n^5	100 μs	.003 s	.02 s	.1 s	.3 s	.7 s
2^n	1 μs	.001 s	1 s	18 m	13 T	36 J
3^n	59 μs	3 s	2 T	385 J	$2 \cdot 10^7$ J	10^{12} J

μs =Mikrosek., s=Sek., m=Minute, T=Tag, J=Jahr

Problemgröße lösbar in fester Zeit: Speedup

Komplexität	n	n^2	n^5	2^n	3^n
1 MHz Prozessor	N_1	N_2	N_3	N_4	N_5
1 GHz Prozessor	$1000 N_1$	$32 N_2$	$4 N_3$	$N_4 + 10$	$N_5 + 6$

Zwei zentrale Komplexitätsklassen:

- P = die von einer DTM in polynomieller Zeit lösbaren Probleme
- = die „leichten“ Probleme (*feasible* problems)

Zwei zentrale Komplexitätsklassen:

P = die von einer DTM in polynomieller Zeit lösbaren Probleme
= die „leichten“ Probleme (*feasible problems*)

NP = die von einer NTM in polynomieller Zeit lösbaren Probleme
In exponentieller Zeit lösbar (s. Simulation NTM durch DTM)

Zwei zentrale Komplexitätsklassen:

P = die von einer DTM in polynomieller Zeit lösbaren Probleme
= die „leichten“ Probleme (*feasible* problems)

NP = die von einer NTM in polynomieller Zeit lösbaren Probleme
In exponentieller Zeit lösbar (s. Simulation NTM durch DTM)

Zentrale Frage:

¿ $P = NP$?

Zwei zentrale Komplexitätsklassen:

P = die von einer DTM in polynomieller Zeit lösbaren Probleme
= die „leichten“ Probleme (*feasible problems*)

NP = die von einer NTM in polynomieller Zeit lösbaren Probleme
In exponentieller Zeit lösbar (s. Simulation NTM durch DTM)

Zentrale Frage:

¿ $P = NP$?

Ist wichtig

Zwei zentrale Komplexitätsklassen:

P = die von einer DTM in polynomieller Zeit lösbaren Probleme
= die „leichten“ Probleme (*feasible problems*)

NP = die von einer NTM in polynomieller Zeit lösbaren Probleme
In exponentieller Zeit lösbar (s. Simulation NTM durch DTM)

Zentrale Frage:

¿ $P = NP$?

Ist wichtig

- weil viele *praktisch relevante* Such- und Optimierungsprobleme in NP liegen
- und alle schnell lösbar wären wenn $P = NP$.

Überblick:

- 1 Die Komplexitätsklassen P und NP

Überblick:

- 1 Die Komplexitätsklassen P und NP
- 2 NP-Vollständigkeit

Überblick:

- 1 Die Komplexitätsklassen P und NP
- 2 NP-Vollständigkeit
- 3 SAT: Ein NP-vollständiges Problem

Überblick:

- 1 Die Komplexitätsklassen P und NP
- 2 NP-Vollständigkeit
- 3 SAT: Ein NP-vollständiges Problem
- 4 Weitere NP-vollständige Probleme

6.1 Die Komplexitätsklasse P

Berechnungsmodelle:

DTM = deterministische Mehrband-TM

NTM = nichtdeterministische Mehrband-TM

6.1 Die Komplexitätsklasse P

Berechnungsmodelle:

DTM = deterministische Mehrband-TM

NTM = nichtdeterministische Mehrband-TM

Definition 6.1

$time_M(w)$:= Anzahl der Schritte bis die DTM $M[w]$ hält
 $\in \mathbb{N} \cup \{\infty\}$

6.1 Die Komplexitätsklasse P

Berechnungsmodelle:

DTM = deterministische Mehrband-TM

NTM = nichtdeterministische Mehrband-TM

Definition 6.1

$time_M(w)$:= Anzahl der Schritte bis die DTM $M[w]$ hält
 $\in \mathbb{N} \cup \{\infty\}$

Sei $f : \mathbb{N} \rightarrow \mathbb{N}$ eine totale Funktion.

Die Klasse der in Zeit $f(n)$ entscheidbaren Sprachen:

$TIME(f(n)) := \{A \subseteq \Sigma^* \mid \exists \text{DTM } M. A = L(M) \wedge$
 $\forall w \in \Sigma^*. time_M(w) \leq f(|w|)\}$

6.1 Die Komplexitätsklasse P

Berechnungsmodelle:

DTM = deterministische Mehrband-TM

NTM = nichtdeterministische Mehrband-TM

Definition 6.1

$time_M(w) :=$ Anzahl der Schritte bis die DTM $M[w]$ hält
 $\in \mathbb{N} \cup \{\infty\}$

Sei $f : \mathbb{N} \rightarrow \mathbb{N}$ eine totale Funktion.

Die Klasse der in Zeit $f(n)$ entscheidbaren Sprachen:

$TIME(f(n)) := \{A \subseteq \Sigma^* \mid \exists \text{DTM } M. A = L(M) \wedge$
 $\forall w \in \Sigma^*. time_M(w) \leq f(|w|)\}$

Merke:

- n ist implizit die Länge der Eingabe
- Die DTM entscheidet die Sprache A in $\leq f(n)$ Schritten

Wir betrachten Polynome mit Koeffizienten $a_k, \dots, a_0 \in \mathbb{N}$:

$$p(n) = a_k n^k + \dots + a_1 n + a_0$$

Wir betrachten Polynome mit Koeffizienten $a_k, \dots, a_0 \in \mathbb{N}$:

$$p(n) = a_k n^k + \dots + a_1 n + a_0$$

Definition 6.2

$$P := \bigcup_{p \text{ Polynom}} \text{TIME}(p(n))$$

Wir betrachten Polynome mit Koeffizienten $a_k, \dots, a_0 \in \mathbb{N}$:

$$p(n) = a_k n^k + \dots + a_1 n + a_0$$

Definition 6.2

$$P := \bigcup_{p \text{ Polynom}} \text{TIME}(p(n))$$

Damit gilt auch

$$P = \bigcup_{k \geq 0} \text{TIME}(O(n^k))$$

wobei

$$\text{TIME}(O(f)) := \bigcup_{g \in O(f)} \text{TIME}(g)$$

Beispiel 6.3

- $\{ww^R \mid w \in \Sigma^*\} \in \text{TIME}(O(n^2)) \subseteq \text{P}$

Beispiel 6.3

- $\{ww^R \mid w \in \Sigma^*\} \in \text{TIME}(O(n^2)) \subseteq \text{P}$
- $\{(G, w) \mid G \text{ ist CFG} \wedge w \in L(G)\} \in \text{P}$

Beispiel 6.3

- $\{ww^R \mid w \in \Sigma^*\} \in \text{TIME}(O(n^2)) \subseteq \text{P}$
- $\{(G, w) \mid G \text{ ist CFG} \wedge w \in L(G)\} \in \text{P}$
- $\{(G, w) \mid G \text{ ist Graph} \wedge w \text{ ist Pfad in } G\} \in \text{P}$

Beispiel 6.3

- $\{ww^R \mid w \in \Sigma^*\} \in \text{TIME}(O(n^2)) \subseteq \text{P}$
- $\{(G, w) \mid G \text{ ist CFG} \wedge w \in L(G)\} \in \text{P}$
- $\{(G, w) \mid G \text{ ist Graph} \wedge w \text{ ist Pfad in } G\} \in \text{P}$
- $\{\text{bin}(p) \mid p \text{ Primzahl}\} \in \text{P}$

Beispiel 6.3

- $\{ww^R \mid w \in \Sigma^*\} \in \text{TIME}(O(n^2)) \subseteq \text{P}$
- $\{(G, w) \mid G \text{ ist CFG} \wedge w \in L(G)\} \in \text{P}$
- $\{(G, w) \mid G \text{ ist Graph} \wedge w \text{ ist Pfad in } G\} \in \text{P}$
- $\{\text{bin}(p) \mid p \text{ Primzahl}\} \in \text{P}$

Beweis durch Angabe eines Algorithmus mit Komplexität $O(n^k)$.

Beispiel 6.3

- $\{ww^R \mid w \in \Sigma^*\} \in \text{TIME}(O(n^2)) \subseteq P$
- $\{(G, w) \mid G \text{ ist CFG} \wedge w \in L(G)\} \in P$
- $\{(G, w) \mid G \text{ ist Graph} \wedge w \text{ ist Pfad in } G\} \in P$
- $\{\text{bin}(p) \mid p \text{ Primzahl}\} \in P$

Beweis durch Angabe eines Algorithmus mit Komplexität $O(n^k)$.

Bemerkungen

- $O(n \log n) \subset O(n^2)$
- $n^{\log n}, 2^n \notin O(n^k)$ für alle k
- Beweis $A \notin P$ meist schwierig

- Warum P und nicht (zB) $O(n^{17})$?

Um robust bzgl Maschinenmodell zu sein:

1-Band DTM braucht $O(t^2)$ Schritte

um t Schritte einer k -Band DTM zu simulieren.

- Warum P und nicht (zB) $O(n^{17})$?

Um robust bzgl Maschinenmodell zu sein:

1-Band DTM braucht $O(t^2)$ Schritte

um t Schritte einer k -Band DTM zu simulieren.

Fast alle bekannten „vernünftigen“ Maschinenmodelle lassen sich von einer DTM in polynomieller Zeit simulieren.

Offen: Quantencomputer

- Warum P und nicht (zB) $O(n^{17})$?

Um robust bzgl Maschinenmodell zu sein:

1-Band DTM braucht $O(t^2)$ Schritte

um t Schritte einer k -Band DTM zu simulieren.

Fast alle bekannten „vernünftigen“ Maschinenmodelle lassen sich von einer DTM in polynomieller Zeit simulieren.

Offen: Quantencomputer

- Warum TM?

- Warum P und nicht (zB) $O(n^{17})$?

Um robust bzgl Maschinenmodell zu sein:

1-Band DTM braucht $O(t^2)$ Schritte

um t Schritte einer k -Band DTM zu simulieren.

Fast alle bekannten „vernünftigen“ Maschinenmodelle lassen sich von einer DTM in polynomieller Zeit simulieren.

Offen: Quantencomputer

- Warum TM? Historisch.

Ebenfalls möglich: (zB) WHILE.

Aber zwei mögliche Kostenmodelle:

- Warum P und nicht (zB) $O(n^{17})$?

Um robust bzgl Maschinenmodell zu sein:

1-Band DTM braucht $O(t^2)$ Schritte

um t Schritte einer k -Band DTM zu simulieren.

Fast alle bekannten „vernünftigen“ Maschinenmodelle lassen sich von einer DTM in polynomieller Zeit simulieren.

Offen: Quantencomputer

- Warum TM? Historisch.

Ebenfalls möglich: (zB) WHILE.

Aber zwei mögliche Kostenmodelle:

Uniform $x_i := x_j + n$ kostet 1 Zeiteinheit.

- Warum P und nicht (zB) $O(n^{17})$?

Um robust bzgl Maschinenmodell zu sein:

1-Band DTM braucht $O(t^2)$ Schritte

um t Schritte einer k -Band DTM zu simulieren.

Fast alle bekannten „vernünftigen“ Maschinenmodelle lassen sich von einer DTM in polynomieller Zeit simulieren.

Offen: Quantencomputer

- Warum TM? Historisch.

Ebenfalls möglich: (zB) WHILE.

Aber zwei mögliche Kostenmodelle:

Uniform $x_i := x_j + n$ kostet 1 Zeiteinheit.

Logarithmisch $x_i := x_j + n$ kostet $\log x_j$ Zeiteinheiten.

6.2 Die Komplexitätsklasse NP

In Worten:

- NP ist die Klasse der Sprachen, die von einer NTM in polynomieller Zeit akzeptiert werden.

6.2 Die Komplexitätsklasse NP

In Worten:

- NP ist die Klasse der Sprachen, die von einer NTM in polynomieller Zeit akzeptiert werden.
- Dh: Eine Sprache A liegt in NP gdw es ein Polynom $p(n)$ und eine NTM M gibt, so dass
 - ① $L(M) = A$ und
 - ② für alle $w \in A$ kann $M[w]$ in $\leq p(|w|)$ Schritten akzeptieren, dh einen Endzustand erreichen.

Definition 6.4

$$ntime_M(w) := \begin{cases} \text{minimale Anzahl der Schritte} & \text{falls } w \in L(M) \\ \text{bis NTM } M[w] \text{ akzeptiert} & \\ 0 & \text{falls } w \notin L(M) \end{cases}$$

Definition 6.4

$$ntime_M(w) := \begin{cases} \text{minimale Anzahl der Schritte} & \text{falls } w \in L(M) \\ \text{bis NTM } M[w] \text{ akzeptiert} & \\ 0 & \text{falls } w \notin L(M) \end{cases}$$

Sei $f : \mathbb{N} \rightarrow \mathbb{N}$ eine totale Funktion.

$$NTIME(f(n)) := \{A \subseteq \Sigma^* \mid \exists \text{NTM } M. A = L(M) \wedge \\ \forall w \in \Sigma^*. ntime_M(w) \leq f(|w|)\}$$

$$NP := \bigcup_{p \text{ Polynom}} NTIME(p(n))$$

Definition 6.4

$$ntime_M(w) := \begin{cases} \text{minimale Anzahl der Schritte} & \text{falls } w \in L(M) \\ \text{bis NTM } M[w] \text{ akzeptiert} & \\ 0 & \text{falls } w \notin L(M) \end{cases}$$

Sei $f : \mathbb{N} \rightarrow \mathbb{N}$ eine totale Funktion.

$$NTIME(f(n)) := \{A \subseteq \Sigma^* \mid \exists \text{NTM } M. A = L(M) \wedge \\ \forall w \in \Sigma^*. ntime_M(w) \leq f(|w|)\}$$

$$NP := \bigcup_{p \text{ Polynom}} NTIME(p(n))$$

Bemerkungen:

- $P \subseteq NP$
- Seit etwa 1970 ist offen ob $P = NP$.

Bemerkungen zur Definition von NP:

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnungsfolgen haben.

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnungsfolgen haben.

Äquivalente Definition NP' von NP: Die NTM $M[w]$ muss nach maximal $p(|w|)$ Schritten halten.

$NP' \subseteq NP$: Klar.

$NP \subseteq NP'$: Falls $A \in NP$ mit Polynom p und NTM M , so kann man NTM M' konstruieren mit $L(M') = A$, so dass $M'[w]$ immer innerhalb von polynomieller Zeit hält.

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnungsfolgen haben.

Äquivalente Definition NP' von NP: Die NTM $M[w]$ muss nach maximal $p(|w|)$ Schritten halten.

$NP' \subseteq NP$: Klar.

$NP \subseteq NP'$: Falls $A \in NP$ mit Polynom p und NTM M , so kann man NTM M' konstruieren mit $L(M') = A$, so dass $M'[w]$ immer innerhalb von polynomieller Zeit hält.

- 1 Eingabe w

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnungsfolgen haben.

Äquivalente Definition NP' von NP: Die NTM $M[w]$ muss nach maximal $p(|w|)$ Schritten halten.

NP' \subseteq NP: Klar.

NP \subseteq NP': Falls $A \in$ NP mit Polynom p und NTM M , so kann man NTM M' konstruieren mit $L(M') = A$, so dass $M'[w]$ immer innerhalb von polynomieller Zeit hält.

- 1 Eingabe w
- 2 Schreibe $p(|w|)$ auf ein getrenntes Band („timer“)

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnungsfolgen haben.

Äquivalente Definition NP' von NP: Die NTM $M[w]$ muss nach maximal $p(|w|)$ Schritten halten.

NP' \subseteq NP: Klar.

NP \subseteq NP': Falls $A \in$ NP mit Polynom p und NTM M , so kann man NTM M' konstruieren mit $L(M') = A$, so dass $M'[w]$ immer innerhalb von polynomieller Zeit hält.

- 1 Eingabe w
- 2 Schreibe $p(|w|)$ auf ein getrenntes Band („timer“)
- 3 Simuliere $M[w]$

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnungsfolgen haben.

Äquivalente Definition NP' von NP: Die NTM $M[w]$ *muss* nach maximal $p(|w|)$ Schritten halten.

NP' \subseteq NP: Klar.

NP \subseteq NP': Falls $A \in$ NP mit Polynom p und NTM M , so kann man NTM M' konstruieren mit $L(M') = A$, so dass $M'[w]$ immer innerhalb von polynomieller Zeit hält.

- 1 Eingabe w
- 2 Schreibe $p(|w|)$ auf ein getrenntes Band („timer“)
- 3 Simuliere $M[w]$, aber dekrementiere timer nach jedem Schritt.

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnungsfolgen haben.

Äquivalente Definition NP' von NP: Die NTM $M[w]$ muss nach maximal $p(|w|)$ Schritten halten.

NP' \subseteq NP: Klar.

NP \subseteq NP': Falls $A \in$ NP mit Polynom p und NTM M , so kann man NTM M' konstruieren mit $L(M') = A$, so dass $M'[w]$ immer innerhalb von polynomieller Zeit hält.

- 1 Eingabe w
- 2 Schreibe $p(|w|)$ auf ein getrenntes Band („timer“)
- 3 Simuliere $M[w]$, aber dekrementiere timer nach jedem Schritt.
- 4 Wird timer=0, ohne dass M gehalten hat, halte in einem nicht-Endzustand („timeout“)

Beispiel 6.5

- Ein Euler-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jede Kante genau einmal enthält.
- Ein Graph hat einen Euler-Kreis gdw er zusammenhängend ist, und jeder Knoten geraden Grad hat. Beide Eigenschaften sind in polynomieller Zeit von einer DTM überprüfbar.
- $\implies \{G \mid G \text{ hat Euler-Kreis}\} \in P$

Beispiel 6.6

- Ein Hamilton-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jeden Knoten genau einmal enthält.

Beispiel 6.6

- Ein Hamilton-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jeden Knoten genau einmal enthält.
- $\text{HAMILTON} := \{G \mid G \text{ hat Hamilton-Kreis}\} \in \text{NP}$ mit folgendem Algorithmus der Art „Rate und prüfe“:
 - ① Rate eine Permutation der Knoten des Graphen.
 - ② Prüfe, ob diese Permutation ein Hamilton-Kreis ist.

Beispiel 6.6

- Ein Hamilton-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jeden Knoten genau einmal enthält.
- $\text{HAMILTON} := \{G \mid G \text{ hat Hamilton-Kreis}\} \in \text{NP}$ mit folgendem Algorithmus der Art „Rate und prüfe“:
 - ① Rate eine Permutation der Knoten des Graphen.
 - ② Prüfe, ob diese Permutation ein Hamilton-Kreis ist.

Das Raten ist in polynomieller Zeit von einer NTM machbar.

Das Prüfen ist in polynomieller Zeit von einer DTM machbar.

Beispiel 6.6

- Ein Hamilton-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jeden Knoten genau einmal enthält.
- $\text{HAMILTON} := \{G \mid G \text{ hat Hamilton-Kreis}\} \in \text{NP}$ mit folgendem Algorithmus der Art „Rate und prüfe“:
 - ① Rate eine Permutation der Knoten des Graphen.
 - ② Prüfe, ob diese Permutation ein Hamilton-Kreis ist.

Das Raten ist in polynomieller Zeit von einer NTM machbar.

Das Prüfen ist in polynomieller Zeit von einer DTM machbar.

Vermutung: $\text{HAMILTON} \notin \text{P}$ da man keinen substanziiell besseren Algorithmus kennt, als alle Permutationen auszuprobieren.

Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob eine Lösungsvorschlag eine Lösung ist.

Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob eine Lösungsvorschlag eine Lösung ist.

Definition 6.7

Sei M eine DTM mit $L(M) \subseteq \{w\#c \mid w \in \Sigma^*, c \in \Delta^*\}$.

- Falls $w\#c \in L(M)$, so heißt c **Zertifikat** für w .
- M ist ein **polynomiell beschränkter Verifikator** für die Sprache $\{w \in \Sigma^* \mid \exists c \in \Delta^*. w\#c \in L(M)\}$ falls es ein Polynom p gibt, so dass $time_M(w\#c) \leq p(|w|)$.

Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob eine Lösungsvorschlag eine Lösung ist.

Definition 6.7

Sei M eine DTM mit $L(M) \subseteq \{w\#c \mid w \in \Sigma^*, c \in \Delta^*\}$.

- Falls $w\#c \in L(M)$, so heißt c **Zertifikat** für w .
- M ist ein **polynomiell beschränkter Verifikator** für die Sprache $\{w \in \Sigma^* \mid \exists c \in \Delta^*. w\#c \in L(M)\}$ falls es ein Polynom p gibt, so dass $time_M(w\#c) \leq p(|w|)$.

NB: In Zeit $p(n)$ kann M maximal die ersten $p(n)$ Zeichen von c lesen. Daher genügt für w ein Zertifikat der Länge $\leq p(|w|)$.

Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob eine Lösungsvorschlag eine Lösung ist.

Definition 6.7

Sei M eine DTM mit $L(M) \subseteq \{w\#c \mid w \in \Sigma^*, c \in \Delta^*\}$.

- Falls $w\#c \in L(M)$, so heißt c **Zertifikat** für w .
- M ist ein **polynomiell beschränkter Verifikator** für die Sprache $\{w \in \Sigma^* \mid \exists c \in \Delta^*. w\#c \in L(M)\}$ falls es ein Polynom p gibt, so dass $time_M(w\#c) \leq p(|w|)$.

NB: In Zeit $p(n)$ kann M maximal die ersten $p(n)$ Zeichen von c lesen. Daher genügt für w ein Zertifikat der Länge $\leq p(|w|)$.

Beispiel 6.8 (HAMILTON)

Zertifikat: Knotenpermutation. In polynomieller Zeit verifizierbar.

Beispiel 6.9 (RUCKSACK)

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$ und $c \in \mathbb{N}$.

Problem: Gibt es $R \subseteq \{1, \dots, n\}$ mit $\sum_{i \in R} a_i = c$?

$$\text{RUCKSACK} := \{ \text{bin}(a_1) \# \dots \# \text{bin}(a_n) \# \text{bin}(c) \mid \\ \exists R \subseteq \{1, \dots, n\}. \sum_{i \in R} a_i = c \}$$

Beispiel 6.9 (RUCKSACK)

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$ und $c \in \mathbb{N}$.

Problem: Gibt es $R \subseteq \{1, \dots, n\}$ mit $\sum_{i \in R} a_i = c$?

$$\text{RUCKSACK} := \{ \text{bin}(a_1) \# \dots \# \text{bin}(a_n) \# \text{bin}(c) \mid \\ \exists R \subseteq \{1, \dots, n\}. \sum_{i \in R} a_i = c \}$$

RUCKSACK \in NP:

Zertifikat: Indexmenge R . In polynomieller Zeit verifizierbar.

Beispiel 6.9 (RUCKSACK)

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$ und $c \in \mathbb{N}$.

Problem: Gibt es $R \subseteq \{1, \dots, n\}$ mit $\sum_{i \in R} a_i = c$?

$$\text{RUCKSACK} := \{ \text{bin}(a_1) \# \dots \# \text{bin}(a_n) \# \text{bin}(c) \mid \\ \exists R \subseteq \{1, \dots, n\}. \sum_{i \in R} a_i = c \}$$

RUCKSACK \in NP:

Zertifikat: Indexmenge R . In polynomieller Zeit verifizierbar.

Merke: Der Nachweis $A \in$ NP ist meistens einfach.

Satz 6.10

$A \in NP$ gdw es gibt polynomiell beschränkten Verifikator für A .

Satz 6.10

$A \in NP$ gdw es gibt polynomiell beschränkten Verifikator für A .

Beweis:

„ \Rightarrow “:

Satz 6.10

$A \in NP$ gdw es gibt polynomiell beschränkten Verifikator für A .

Beweis:

„ \Rightarrow “:

Sei $A \in NP$. Dh es gibt NTM N , die A in Zeit $p(n)$ akzeptiert.

Satz 6.10

$A \in NP$ gdw es gibt polynomiell beschränkten Verifikator für A .

Beweis:

„ \Rightarrow “:

Sei $A \in NP$. Dh es gibt NTM N , die A in Zeit $p(n)$ akzeptiert.
Ein Zertifikat für $w \in A$ ist die Folge der benutzten Transitionen $\delta(q, a) \ni (q', a', d)$ einer akzeptierenden Berechnungsfolge von $N[w]$ mit $\leq p(n)$ Schritten.

Satz 6.10

$A \in NP$ gdw es gibt polynomiell beschränkten Verifikator für A .

Beweis:

„ \Rightarrow “:

Sei $A \in NP$. Dh es gibt NTM N , die A in Zeit $p(n)$ akzeptiert. Ein Zertifikat für $w \in A$ ist die Folge der benutzten Transitionen $\delta(q, a) \ni (q', a', d)$ einer akzeptierenden Berechnungsfolge von $N[w]$ mit $\leq p(n)$ Schritten.

Ein polynomiell beschränkter Verifikator für $L(N)$:

- 1 Eingabe $w \# c$
- 2 Simuliere $N[w]$, gesteuert durch die Transitionen in c .
- 3 Überprüfe dabei, ob die Transition in c jeweils zu N und zur augenblicklichen Konfiguration von N passt.
- 4 Akzeptiere, falls c in einen Endzustand führt.

Beweis (Forts.):

„ \Leftarrow “:

Sei M ein polynomiell (durch p) beschränkter Verifikator für A .

Wir bauen eine polynomiell beschränkte NTM N mit $L(M) = A$:

Beweis (Forts.):

„ \Leftarrow “:

Sei M ein polynomiell (durch p) beschränkter Verifikator für A .

Wir bauen eine polynomiell beschränkte NTM N mit $L(M) = A$:

- 1 Eingabe w .
- 2 Schreibe hinter w zuerst $\#$ und dann ein nichtdeterministisch gewähltes Wort $c \in \Delta^*$

Beweis (Forts.):

„ \Leftarrow “:

Sei M ein polynomiell (durch p) beschränkter Verifikator für A .

Wir bauen eine polynomiell beschränkte NTM N mit $L(M) = A$:

- 1 Eingabe w .
- 2 Schreibe hinter w zuerst $\#$ und dann ein nichtdeterministisch gewähltes Wort $c \in \Delta^*$, $|c| = p(|w|)$:
for $i := 1, \dots, p(|w|)$ **do**
 schreibe ein Zeichen aus Δ und gehe nach rechts
- 3 Führe M aus (mit Eingabe $w\#c$).

Beweis (Forts.):

„ \Leftarrow “:

Sei M ein polynomiell (durch p) beschränkter Verifikator für A .

Wir bauen eine polynomiell beschränkte NTM N mit $L(M) = A$:

- 1 Eingabe w .
- 2 Schreibe hinter w zuerst $\#$ und dann ein nichtdeterministisch gewähltes Wort $c \in \Delta^*$, $|c| = p(|w|)$:
for $i := 1, \dots, p(|w|)$ **do**
 schreibe ein Zeichen aus Δ und gehe nach rechts
- 3 Führe M aus (mit Eingabe $w\#c$).

Nach Annahme gilt $\text{time}_M(w\#c) \leq p(|w|)$.

Damit braucht $N[w]$ $O(p(|w|))$ Schritte. □

Fazit:

- P** sind die Sprachen, bei denen $w \in L$ schnell **entschieden** werden kann.
- NP** sind die Sprachen, bei denen ein Zertifikat für $w \in L$ schnell **verifiziert/überprüft** werden kann.

Fazit:

P sind die Sprachen, bei denen $w \in L$ schnell **entschieden** werden kann.

NP sind die Sprachen, bei denen ein Zertifikat für $w \in L$ schnell **verifiziert/überprüft** werden kann.

Intuition:

Es ist leichter, eine Lösung zu verifizieren als zu finden.

Fazit:

P sind die Sprachen, bei denen $w \in L$ schnell **entschieden** werden kann.

NP sind die Sprachen, bei denen ein Zertifikat für $w \in L$ schnell **verifiziert/überprüft** werden kann.

Intuition:

Es ist leichter, eine Lösung zu verifizieren als zu finden.

Aber:

Noch wurde von keiner Sprache bewiesen, dass sie in $NP \setminus P$ liegt.

6.3 NP-Vollständigkeit

- 1 Polynomielle Reduzierbarkeit \leq_p
- 2 NP-vollständige Probleme = härteste Probleme in NP,

6.3 NP-Vollständigkeit

- 1 Polynomielle Reduzierbarkeit \leq_p
- 2 NP-vollständige Probleme = härteste Probleme in NP, alle anderen Probleme in NP darauf polynomiell reduzierbar
- 3 Satz: SAT ist NP-vollständig

Definition 6.11

Sei $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$.

Dann heißt A **polynomiell reduzierbar** auf B , $A \leq_p B$,
gdw es eine totale und von einer DTM in polynomieller Zeit
berechenbare Funktion $f : \Sigma^* \rightarrow \Gamma^*$ gibt, so dass für alle $w \in \Sigma^*$

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

Definition 6.11

Sei $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$.

Dann heißt A **polynomiell reduzierbar** auf B , $A \leq_p B$,
gdw es eine totale und von einer DTM in polynomieller Zeit
berechenbare Funktion $f : \Sigma^* \rightarrow \Gamma^*$ gibt, so dass für alle $w \in \Sigma^*$

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

Da $q(p(n))$ ein Polynom ist falls p und q Polynome sind:

Lemma 6.12

Die Relation \leq_p ist transitiv.

Lemma 6.13

Die Klassen P und NP sind unter polynomieller Reduzierbarkeit nach unten abgeschlossen:

$$A \leq_p B \in P/NP \implies A \in P/NP$$

Lemma 6.13

Die Klassen P und NP sind unter polynomieller Reduzierbarkeit nach unten abgeschlossen:

$$A \leq_p B \in P/NP \implies A \in P/NP$$

Beweis:

- Sei $A \leq_p B$ mittels f , die von DTM M_f berechnet wird. Polynom p begrenzt Rechenzeit von M_f .
- Sei $B \in P$ mittels DTM M . Polynom q begrenzt Rechenzeit von M .

Lemma 6.13

Die Klassen P und NP sind unter polynomieller Reduzierbarkeit nach unten abgeschlossen:

$$A \leq_p B \in P/NP \implies A \in P/NP$$

Beweis:

- Sei $A \leq_p B$ mittels f , die von DTM M_f berechnet wird. Polynom p begrenzt Rechenzeit von M_f .
- Sei $B \in P$ mittels DTM M . Polynom q begrenzt Rechenzeit von M .

Damit ist $M_f; M$ polynomiell zeitbeschränkt in $|w|$:

Lemma 6.13

Die Klassen P und NP sind unter polynomieller Reduzierbarkeit nach unten abgeschlossen:

$$A \leq_p B \in P/NP \implies A \in P/NP$$

Beweis:

- Sei $A \leq_p B$ mittels f , die von DTM M_f berechnet wird. Polynom p begrenzt Rechenzeit von M_f .
- Sei $B \in P$ mittels DTM M . Polynom q begrenzt Rechenzeit von M .

Damit ist $M_f; M$ polynomiell zeitbeschränkt in $|w|$:

- $M_f[w]$ macht $\leq p(|w|)$ Schritte.

Lemma 6.13

Die Klassen P und NP sind unter polynomieller Reduzierbarkeit nach unten abgeschlossen:

$$A \leq_p B \in P/NP \implies A \in P/NP$$

Beweis:

- Sei $A \leq_p B$ mittels f , die von DTM M_f berechnet wird. Polynom p begrenzt Rechenzeit von M_f .
- Sei $B \in P$ mittels DTM M . Polynom q begrenzt Rechenzeit von M .

Damit ist $M_f; M$ polynomiell zeitbeschränkt in $|w|$:

- $M_f[w]$ macht $\leq p(|w|)$ Schritte.
- Ausgabe $f(w)$ von M_f hat Länge $\leq |w| + p(|w|)$.

Lemma 6.13

Die Klassen P und NP sind unter polynomieller Reduzierbarkeit nach unten abgeschlossen:

$$A \leq_p B \in P/NP \implies A \in P/NP$$

Beweis:

- Sei $A \leq_p B$ mittels f , die von DTM M_f berechnet wird. Polynom p begrenzt Rechenzeit von M_f .
- Sei $B \in P$ mittels DTM M . Polynom q begrenzt Rechenzeit von M .

Damit ist $M_f; M$ polynomiell zeitbeschränkt in $|w|$:

- $M_f[w]$ macht $\leq p(|w|)$ Schritte.
- Ausgabe $f(w)$ von M_f hat Länge $\leq |w| + p(|w|)$.
- M macht $\leq q(|f(w)|) \leq q(|w| + p(|w|))$ Schritte

Lemma 6.13

Die Klassen P und NP sind unter polynomieller Reduzierbarkeit nach unten abgeschlossen:

$$A \leq_p B \in P/NP \implies A \in P/NP$$

Beweis:

- Sei $A \leq_p B$ mittels f , die von DTM M_f berechnet wird. Polynom p begrenzt Rechenzeit von M_f .
- Sei $B \in P$ mittels DTM M . Polynom q begrenzt Rechenzeit von M .

Damit ist $M_f; M$ polynomiell zeitbeschränkt in $|w|$:

- $M_f[w]$ macht $\leq p(|w|)$ Schritte.
- Ausgabe $f(w)$ von M_f hat Länge $\leq |w| + p(|w|)$.
- M macht $\leq q(|f(w)|) \leq q(|w| + p(|w|))$ Schritte (q monoton)

Lemma 6.13

Die Klassen P und NP sind unter polynomieller Reduzierbarkeit nach unten abgeschlossen:

$$A \leq_p B \in P/NP \implies A \in P/NP$$

Beweis:

- Sei $A \leq_p B$ mittels f , die von DTM M_f berechnet wird. Polynom p begrenzt Rechenzeit von M_f .
- Sei $B \in P$ mittels DTM M . Polynom q begrenzt Rechenzeit von M .

Damit ist $M_f; M$ polynomiell zeitbeschränkt in $|w|$:

- $M_f[w]$ macht $\leq p(|w|)$ Schritte.
- Ausgabe $f(w)$ von M_f hat Länge $\leq |w| + p(|w|)$.
- M macht $\leq q(|f(w)|) \leq q(|w| + p(|w|))$ Schritte (q monoton)

Daher macht $(M_f; M)[w]$ maximal $p(|w|) + q(|w| + p(|w|))$ Schritte, ein Polynom in $|w|$.

Lemma 6.13

Die Klassen P und NP sind unter polynomieller Reduzierbarkeit nach unten abgeschlossen:

$$A \leq_p B \in P/NP \implies A \in P/NP$$

Beweis:

- Sei $A \leq_p B$ mittels f , die von DTM M_f berechnet wird. Polynom p begrenzt Rechenzeit von M_f .
- Sei $B \in P$ mittels DTM M . Polynom q begrenzt Rechenzeit von M .

Damit ist $M_f; M$ polynomiell zeitbeschränkt in $|w|$:

- $M_f[w]$ macht $\leq p(|w|)$ Schritte.
- Ausgabe $f(w)$ von M_f hat Länge $\leq |w| + p(|w|)$.
- M macht $\leq q(|f(w)|) \leq q(|w| + p(|w|))$ Schritte (q monoton)

Daher macht $(M_f; M)[w]$ maximal $p(|w|) + q(|w| + p(|w|))$ Schritte, ein Polynom in $|w|$.

Analog: $A \leq_p B \in NP \implies A \in NP$



Ein Problem ist **NP-hart**,
wenn es mindestens so hart wie alles in NP ist:

Definition 6.14

Eine Sprache L heißt **NP-hart** gdw $A \leq_p L$ für alle $A \in \text{NP}$.

Ein Problem ist **NP-hart**,
wenn es mindestens so hart wie alles in NP ist:

Definition 6.14

Eine Sprache L heißt **NP-hart** gdw $A \leq_p L$ für alle $A \in \text{NP}$.

Definition 6.15

Eine Sprache L heißt **NP-vollständig** gdw
 L NP-hart ist und $L \in \text{NP}$.

Ein Problem ist **NP-hart**,
wenn es mindestens so hart wie alles in NP ist:

Definition 6.14

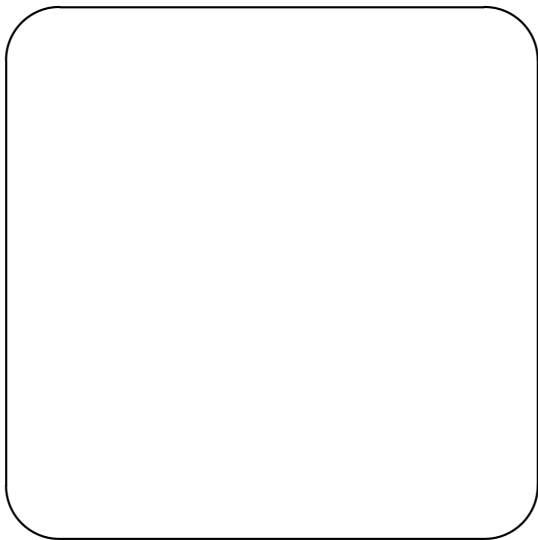
Eine Sprache L heißt **NP-hart** gdw $A \leq_p L$ für alle $A \in \text{NP}$.

Definition 6.15

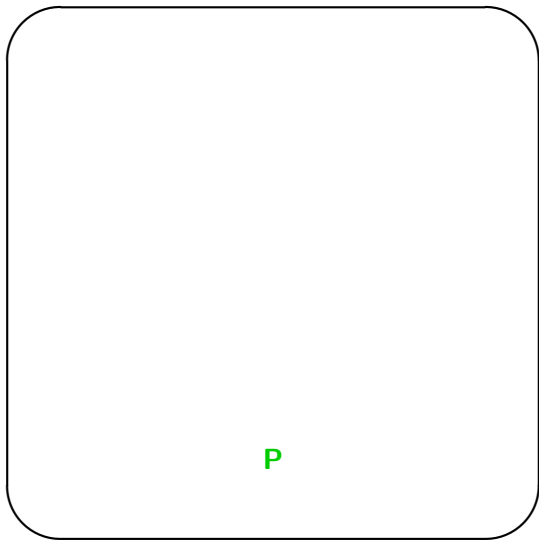
Eine Sprache L heißt **NP-vollständig** gdw
 L NP-hart ist und $L \in \text{NP}$.

NP-vollständige Probleme sind die “schwierigsten” Probleme in NP:

alle Probleme in NP sind polynomiell auf sie reduzierbar.

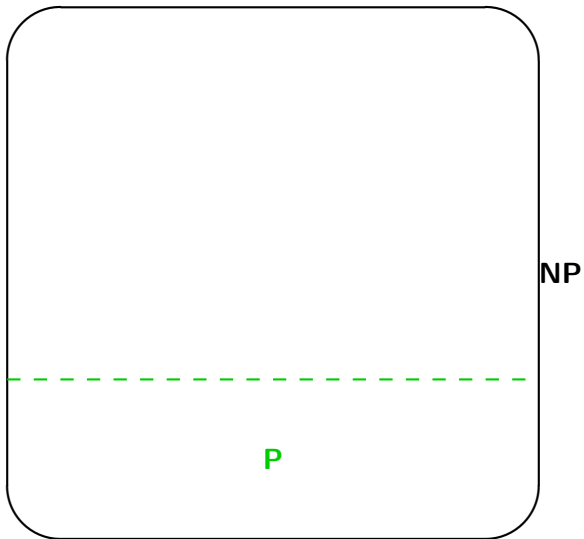


NP

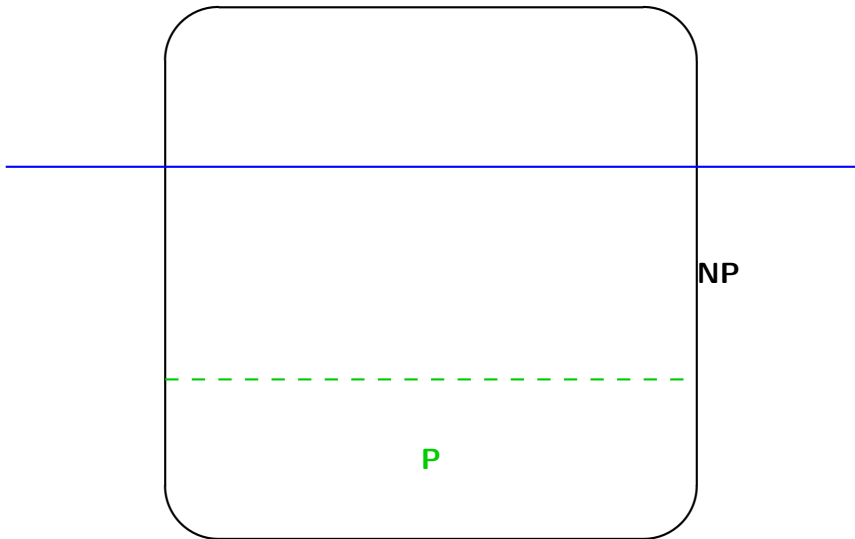


NP

P



NP-hart



NP-hart

NP-vollständig

NP

P

Wie man $P \stackrel{?}{=} NP$ lösen kann:

Lemma 6.16

Es gilt $P=NP$ gdw ein NP-vollständiges Problem in P liegt.

Wie man $P \stackrel{?}{=} NP$ lösen kann:

Lemma 6.16

Es gilt $P=NP$ gdw ein NP-vollständiges Problem in P liegt.

Beweis:

„ \Rightarrow “: Falls $P=NP$, so liegt jedes NP-vollständige Problem in P .

Wie man $P \stackrel{?}{=} NP$ lösen kann:

Lemma 6.16

Es gilt $P=NP$ gdw ein NP-vollständiges Problem in P liegt.

Beweis:

„ \Rightarrow “: Falls $P=NP$, so liegt jedes NP-vollständige Problem in P .

„ \Leftarrow “: Sei L ein NP-vollständiges Problem in P .

Dann gilt $P \supseteq NP$:

Ist $A \in NP$, so gilt $A \leq_p L$ (da L NP-hart)

und nach Lemma 6.13 $A \in P$ (da $L \in P$). □

Wie man $P \stackrel{?}{=} NP$ lösen kann:

Lemma 6.16

Es gilt $P=NP$ gdw ein NP-vollständiges Problem in P liegt.

Beweis:

„ \Rightarrow “: Falls $P=NP$, so liegt jedes NP-vollständige Problem in P .

„ \Leftarrow “: Sei L ein NP-vollständiges Problem in P .

Dann gilt $P \supseteq NP$:

Ist $A \in NP$, so gilt $A \leq_p L$ (da L NP-hart)

und nach Lemma 6.13 $A \in P$ (da $L \in P$). □

Starke Vermutung:

- $P \neq NP$
- dh kein NP-vollständiges Problem ist in P .

Wie man $P \stackrel{?}{=} NP$ lösen kann:

Lemma 6.16

Es gilt $P=NP$ gdw ein NP-vollständiges Problem in P liegt.

Beweis:

„ \Rightarrow “: Falls $P=NP$, so liegt jedes NP-vollständige Problem in P .

„ \Leftarrow “: Sei L ein NP-vollständiges Problem in P .

Dann gilt $P \supseteq NP$:

Ist $A \in NP$, so gilt $A \leq_p L$ (da L NP-hart)

und nach Lemma 6.13 $A \in P$ (da $L \in P$). □

Starke Vermutung:

- $P \neq NP$
- dh kein NP-vollständiges Problem ist in P .

Aber gibt es überhaupt NP-vollständige Probleme?

Aussagenlogik

Aussagenlogik

Syntax der Aussagenlogik:

Formeln: $F \rightarrow \neg F \mid (F \wedge F) \mid (F \vee F) \mid X$
Variablen: $X \rightarrow x \mid y \mid z \mid \dots$

Aussagenlogik

Syntax der Aussagenlogik:

Formeln: $F \rightarrow \neg F \mid (F \wedge F) \mid (F \vee F) \mid X$

Variablen: $X \rightarrow x \mid y \mid z \mid \dots$

Bsp: $((\neg x \wedge y) \vee (x \wedge \neg z))$

Aussagenlogik

Syntax der Aussagenlogik:

Formeln: $F \rightarrow \neg F \mid (F \wedge F) \mid (F \vee F) \mid X$

Variablen: $X \rightarrow x \mid y \mid z \mid \dots$

Bsp: $((\neg x \wedge y) \vee (x \wedge \neg z))$

Konvention: man darf äußerste Klammern weglassen

und \wedge bindet stärker als \vee : $x \wedge y \vee z$ ist Abk. für $(x \wedge y) \vee z$

Aussagenlogik

Syntax der Aussagenlogik:

$$\begin{aligned} \text{Formeln: } F &\rightarrow \neg F \mid (F \wedge F) \mid (F \vee F) \mid X \\ \text{Variablen: } X &\rightarrow x \mid y \mid z \mid \dots \end{aligned}$$

Bsp: $((\neg x \wedge y) \vee (x \wedge \neg z))$

Konvention: man darf äußerste Klammern weglassen
und \wedge bindet stärker als \vee : $x \wedge y \vee z$ ist Abk. für $(x \wedge y) \vee z$

Semantik der Aussagenlogik:

- Eine **Belegung** ist eine Funktion von Variablen auf $\{0, 1\}$.
Bsp: $\sigma = \{x \mapsto 0, y \mapsto 1, z \mapsto 0, \dots\}$
- Belegungen werden mittels Wahrheitstabellen auf Formeln erweitert. Bsp: $\sigma((\neg x \wedge y) \vee (x \wedge \neg z)) = 1$

Aussagenlogik

Syntax der Aussagenlogik:

$$\begin{aligned} \text{Formeln: } F &\rightarrow \neg F \mid (F \wedge F) \mid (F \vee F) \mid X \\ \text{Variablen: } X &\rightarrow x \mid y \mid z \mid \dots \end{aligned}$$

Bsp: $((\neg x \wedge y) \vee (x \wedge \neg z))$

Konvention: man darf äußerste Klammern weglassen
und \wedge bindet stärker als \vee : $x \wedge y \vee z$ ist Abk. für $(x \wedge y) \vee z$

Semantik der Aussagenlogik:

- Eine **Belegung** ist eine Funktion von Variablen auf $\{0, 1\}$.
Bsp: $\sigma = \{x \mapsto 0, y \mapsto 1, z \mapsto 0, \dots\}$
- Belegungen werden mittels Wahrheitstabellen auf Formeln erweitert. Bsp: $\sigma((\neg x \wedge y) \vee (x \wedge \neg z)) = 1$
- Eine Formel F ist **erfüllbar** gdw es eine Belegung σ gibt mit $\sigma(F) = 1$

SAT

Gegeben: Eine aussagenlogische Formel F .

Problem: Ist F erfüllbar?

SAT

Gegeben: Eine aussagenlogische Formel F .

Problem: Ist F erfüllbar?

Praktische Bedeutung von SAT:

Äquivalenztest von Schaltungen

SAT

Gegeben: Eine aussagenlogische Formel F .

Problem: Ist F erfüllbar?

Praktische Bedeutung von SAT:

Äquivalenztest von Schaltungen (und vieles mehr)

Fakt 6.17

Zwei Formeln F_1 und F_2 sind genau dann äquivalent ($F_1 \leftrightarrow F_2$), wenn $(F_1 \wedge \neg F_2) \vee (\neg F_1 \wedge F_2)$ nicht erfüllbar ist.

SAT

Gegeben: Eine aussagenlogische Formel F .

Problem: Ist F erfüllbar?

Praktische Bedeutung von SAT:

Äquivalenztest von Schaltungen (und vieles mehr)

Fakt 6.17

Zwei Formeln F_1 und F_2 sind genau dann äquivalent ($F_1 \leftrightarrow F_2$), wenn $(F_1 \wedge \neg F_2) \vee (\neg F_1 \wedge F_2)$ nicht erfüllbar ist.

Lemma 6.18

$SAT \in NP$

SAT

Gegeben: Eine aussagenlogische Formel F .

Problem: Ist F erfüllbar?

Praktische Bedeutung von SAT:

Äquivalenztest von Schaltungen (und vieles mehr)

Fakt 6.17

Zwei Formeln F_1 und F_2 sind genau dann äquivalent ($F_1 \leftrightarrow F_2$), wenn $(F_1 \wedge \neg F_2) \vee (\neg F_1 \wedge F_2)$ nicht erfüllbar ist.

Lemma 6.18

$SAT \in NP$

Beweis:

Belegungen sind Zertifikate, die in polynomieller Zeit geprüft werden können: Es gibt eine DTM, die bei Eingabe einer Formel F und einer Belegung σ für die Variablen in F , in polynomieller Zeit $\sigma(F)$ berechnet. □

Satz 6.19 (Cook 1971)

SAT ist NP-vollständig.

Satz 6.19 (Cook 1971)

SAT ist NP-vollständig.

Beweis:

Da $SAT \in NP$, bleibt noch zu zeigen, SAT ist NP-hart.

Sei $A \in NP$. Wir zeigen $A \leq_p SAT$.

Satz 6.19 (Cook 1971)

SAT ist NP-vollständig.

Beweis:

Da $SAT \in NP$, bleibt noch zu zeigen, SAT ist NP-hart.

Sei $A \in NP$. Wir zeigen $A \leq_p SAT$.

Da $A \in NP$ gibt es NTM M mit $A = L(M)$ und
Polynom p mit $ntime_M(w) \leq p(|w|)$.

Satz 6.19 (Cook 1971)

SAT ist NP-vollständig.

Beweis:

Da $SAT \in NP$, bleibt noch zu zeigen, SAT ist NP-hart.

Sei $A \in NP$. Wir zeigen $A \leq_p SAT$.

Da $A \in NP$ gibt es NTM M mit $A = L(M)$ und

Polynom p mit $ntime_M(w) \leq p(|w|)$.

Reduktion bildet $w = x_1 \dots x_n \in \Sigma^*$ auf eine Formel F ab.

In polynomieller Zeit. So dass $w \in L(M) \Leftrightarrow F \in SAT$.

Satz 6.19 (Cook 1971)

SAT ist NP-vollständig.

Beweis:

Da $SAT \in NP$, bleibt noch zu zeigen, SAT ist NP-hart.

Sei $A \in NP$. Wir zeigen $A \leq_p SAT$.

Da $A \in NP$ gibt es NTM M mit $A = L(M)$ und

Polynom p mit $ntime_M(w) \leq p(|w|)$.

Reduktion bildet $w = x_1 \dots x_n \in \Sigma^*$ auf eine Formel F ab.

In polynomieller Zeit. So dass $w \in L(M) \Leftrightarrow F \in SAT$.

Die Variablen beschreiben das mögliche Verhalten von $M[w]$:

$zust_{t,q}$	$t = 0, \dots, p(n)$ $q \in Q$	$zust_{t,q} = 1 \Leftrightarrow$ Zustand nach t Schritten ist q
--------------	-----------------------------------	--

Satz 6.19 (Cook 1971)

SAT ist NP-vollständig.

Beweis:

Da $SAT \in NP$, bleibt noch zu zeigen, SAT ist NP-hart.

Sei $A \in NP$. Wir zeigen $A \leq_p SAT$.

Da $A \in NP$ gibt es NTM M mit $A = L(M)$ und

Polynom p mit $ntime_M(w) \leq p(|w|)$.

Reduktion bildet $w = x_1 \dots x_n \in \Sigma^*$ auf eine Formel F ab.

In polynomieller Zeit. So dass $w \in L(M) \Leftrightarrow F \in SAT$.

Die Variablen beschreiben das mögliche Verhalten von $M[w]$:

$zust_{t,q}$	$t = 0, \dots, p(n)$ $q \in Q$	$zust_{t,q} = 1 \Leftrightarrow$ Zustand nach t Schritten ist q
$pos_{t,i}$	$t = 0, \dots, p(n)$ $i = -p(n), \dots, p(n)$	$pos_{t,i} = 1 \Leftrightarrow$ Kopfposition nach t Schritten ist i

Satz 6.19 (Cook 1971)

SAT ist NP-vollständig.

Beweis:

Da $SAT \in NP$, bleibt noch zu zeigen, SAT ist NP-hart.

Sei $A \in NP$. Wir zeigen $A \leq_p SAT$.

Da $A \in NP$ gibt es NTM M mit $A = L(M)$ und

Polynom p mit $ntime_M(w) \leq p(|w|)$.

Reduktion bildet $w = x_1 \dots x_n \in \Sigma^*$ auf eine Formel F ab.

In polynomieller Zeit. So dass $w \in L(M) \Leftrightarrow F \in SAT$.

Die Variablen beschreiben das mögliche Verhalten von $M[w]$:

$zust_{t,q}$	$t = 0, \dots, p(n)$ $q \in Q$	$zust_{t,q} = 1 \Leftrightarrow$ Zustand nach t Schritten ist q
$pos_{t,i}$	$t = 0, \dots, p(n)$ $i = -p(n), \dots, p(n)$	$pos_{t,i} = 1 \Leftrightarrow$ Kopfposition nach t Schritten ist i
$band_{t,i,a}$	$t = 0, \dots, p(n)$ $i = -p(n), \dots, p(n)$ $a \in \Gamma$	$band_{t,i,a} = 1 \Leftrightarrow$ Bandinhalt nach t Schritten auf Bandposition i ist Zeichen a

Beweis (Forts.):

Sei $Q = \{q_1, \dots, q_k\}$ und $\Gamma = \{a_1, \dots, a_l\}$.

$$F := R \wedge A \wedge T_1 \wedge T_2 \wedge E$$

Beweis (Forts.):

Sei $Q = \{q_1, \dots, q_k\}$ und $\Gamma = \{a_1, \dots, a_l\}$.

$$F := R \wedge A \wedge T_1 \wedge T_2 \wedge E$$

$$R := \bigwedge_t [G(\text{zust}_{t,q_1}, \dots, \text{zust}_{t,q_k}) \wedge G(\text{pos}_{t,-p(n)}, \dots, \text{pos}_{t,p(n)}) \wedge \bigwedge_i G(\text{band}_{t,i,a_1}, \dots, \text{band}_{t,i,a_l})] \quad (G = \text{Genau1})$$

Beweis (Forts.):

Sei $Q = \{q_1, \dots, q_k\}$ und $\Gamma = \{a_1, \dots, a_l\}$.

$$F := R \wedge A \wedge T_1 \wedge T_2 \wedge E$$

$$R := \bigwedge_t [G(\mathit{zust}_{t,q_1}, \dots, \mathit{zust}_{t,q_k}) \wedge G(\mathit{pos}_{t,-p(n)}, \dots, \mathit{pos}_{t,p(n)}) \wedge \bigwedge_i G(\mathit{band}_{t,i,a_1}, \dots, \mathit{band}_{t,i,a_l})] \quad (G = \textit{Genau1})$$

$$A := \mathit{zust}_{0,q_1} \wedge \mathit{pos}_{0,1} \wedge \bigwedge_{j=1}^n \mathit{band}_{0,j,x_j} \wedge \bigwedge_{j=-p(n)}^0 \mathit{band}_{0,j,\square} \wedge \bigwedge_{j=n+1}^{p(n)} \mathit{band}_{0,j,\square}$$

Beweis (Forts.):

$$\begin{aligned} T_1 &:= \bigwedge_{t,q,i,a} [zust_{t,q} \wedge pos_{t,i} \wedge band_{t,i,a} \\ &\rightarrow \bigvee_{(q',a',y) \in \delta(q,a)} (zust_{t+1,q'} \wedge pos_{t+1,i+y} \wedge band_{t+1,i,a'})] \end{aligned}$$

Beweis (Forts.):

$$T_1 := \bigwedge_{t,q,i,a} [zust_{t,q} \wedge pos_{t,i} \wedge band_{t,i,a} \\ \rightarrow \bigvee_{(q',a',y) \in \delta(q,a)} (zust_{t+1,q'} \wedge pos_{t+1,i+y} \wedge band_{t+1,i,a'})]$$

$$T_2 := \bigwedge_{t,i,a} ((\neg pos_{t,i} \wedge band_{t,i,a}) \rightarrow band_{t+1,i,a})$$

Beweis (Forts.):

$$T_1 := \bigwedge_{t,q,i,a} [zust_{t,q} \wedge pos_{t,i} \wedge band_{t,i,a} \\ \rightarrow \bigvee_{(q',a',y) \in \delta(q,a)} (zust_{t+1,q'} \wedge pos_{t+1,i+y} \wedge band_{t+1,i,a'})]$$

$$T_2 := \bigwedge_{t,i,a} ((\neg pos_{t,i} \wedge band_{t,i,a}) \rightarrow band_{t+1,i,a})$$

$$E := \bigvee_t \bigvee_{q \in F} zust_{t,q}$$

$$G(v_1, \dots, v_r) := \left(\bigvee_{i=1}^r v_i \right) \wedge \left(\bigwedge_{i=1}^{r-1} \bigwedge_{j=i+1}^r \neg(v_i \wedge v_j) \right)$$



Von der Lösbarkeit zur Lösung

Von der Lösbarkeit zur Lösung

Wenn man SAT lösen kann, dann kostet es nicht viel mehr, dazu eine erfüllenden Belegung zu berechnen:

Von der Lösbarkeit zur Lösung

Wenn man SAT lösen kann, dann kostet es nicht viel mehr, dazu eine erfüllenden Belegung zu berechnen:

Sei F eine Formel mit den Variablen x_1, \dots, x_k :

if $F \notin \text{SAT}$ **then** output(“nicht lösbar”)

else

for $i := 1$ **to** k **do**

if $F[x_i := 0] \in \text{SAT}$ **then** $b := 0$ **else** $b := 1$

 output(x_i “=” b)

$F := F[x_i := b]$

wobei $F[x := b] = F$ mit x ersetzt durch b .

Von der Lösbarkeit zur Lösung

Wenn man SAT lösen kann, dann kostet es nicht viel mehr, dazu eine erfüllende Belegung zu berechnen:

Sei F eine Formel mit den Variablen x_1, \dots, x_k :

if $F \notin \text{SAT}$ **then** output(“nicht lösbar”)

else

for $i := 1$ **to** k **do**

if $F[x_i := 0] \in \text{SAT}$ **then** $b := 0$ **else** $b := 1$

 output(x_i “=” b)

$F := F[x_i := b]$

wobei $F[x := b] = F$ mit x ersetzt durch b .

Entscheidung von SAT in Zeit $O(f(n))$

Von der Lösbarkeit zur Lösung

Wenn man SAT lösen kann, dann kostet es nicht viel mehr, dazu eine erfüllenden Belegung zu berechnen:

Sei F eine Formel mit den Variablen x_1, \dots, x_k :

if $F \notin \text{SAT}$ **then** output(“nicht lösbar”)

else

for $i := 1$ **to** k **do**

if $F[x_i := 0] \in \text{SAT}$ **then** $b := 0$ **else** $b := 1$

 output(x_i “=” b)

$F := F[x_i := b]$

wobei $F[x := b] = F$ mit x ersetzt durch b .

Entscheidung von SAT in Zeit $O(f(n))$

\implies Berechnung einer erf. Bel. in Zeit $O(n \cdot (f(n) + n))$

(falls es eine gibt.)

Von der Lösbarkeit zur Lösung

Wenn man SAT lösen kann, dann kostet es nicht viel mehr, dazu eine erfüllenden Belegung zu berechnen:

Sei F eine Formel mit den Variablen x_1, \dots, x_k :

if $F \notin \text{SAT}$ **then** output(“nicht lösbar”)

else

for $i := 1$ **to** k **do**

if $F[x_i := 0] \in \text{SAT}$ **then** $b := 0$ **else** $b := 1$

 output(x_i “=” b)

$F := F[x_i := b]$

wobei $F[x := b] = F$ mit x ersetzt durch b .

Entscheidung von SAT in Zeit $O(f(n))$

\implies Berechnung einer erf. Bel. in Zeit $O(n \cdot (f(n) + n))$

(falls es eine gibt.)

$f(n)$ polynomiell $\implies n \cdot (f(n) + n)$ polynomiell

$f(n)$ exponentiell $\implies n \cdot (f(n) + n)$ exponentiell

Bemerkungen:

- Die Reduzierung der Lösungsberechnung auf SAT ist eine rein theoretische Konstruktion.
- Sie zeigt, dass man sich auf SAT beschränken kann, wenn man nur an polynomiell/exponentiell interessiert ist.
- Alle bekannten Entscheidungsverfahren für SAT berechnen auch eine Lösung.

Von NP-hart zu „NP-leicht“

Von NP-hart zu „NP-leicht“

- Vor dem Jahr 2000:

NP-vollständig = Todesurteil

Von NP-hart zu „NP-leicht“

- Vor dem Jahr 2000:

NP-vollständig = Todesurteil

- In den letzten Jahren:

Spektakuläre Fortschritte bei *Implementierung* von
SAT-Lösern (*SAT-solver*): <http://satcompetition.org>
Stand der Kunst:

Erfüllbar: bis 10^5 Variablen Unerfüllbar: bis 10^3 Variablen

Von NP-hart zu „NP-leicht“

- Vor dem Jahr 2000:

NP-vollständig = Todesurteil

- In den letzten Jahren:

Spektakuläre Fortschritte bei *Implementierung* von SAT-Lösern (*SAT-solver*): <http://satcompetition.org>
Stand der Kunst:

Erfüllbar: bis 10^5 Variablen Unerfüllbar: bis 10^3 Variablen

- Jetzt:

NP-vollständig = Hoffnung durch SAT

- Paradigma:

SAT (Logik!) als universelle Sprache
zur Kodierung kombinatorischer Probleme

Reduktion auf SAT manchmal schneller als problemspezifische Löser! Und fast immer einfacher.

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

3COL

Gegeben: Ein ungerichteter Graph

Problem: Gibt es eine Färbung der Knoten mit 3 Farben, so dass keine benachbarten Knoten gleich gefärbt sind?

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

3COL

Gegeben: Ein ungerichteter Graph

Problem: Gibt es eine Färbung der Knoten mit 3 Farben, so dass keine benachbarten Knoten gleich gefärbt sind?

Lineare Reduktion von Graph $G = (V, E)$ auf SAT:

- Variablen = $V \times \{1, 2, 3\}$.

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

3COL

Gegeben: Ein ungerichteter Graph

Problem: Gibt es eine Färbung der Knoten mit 3 Farben, so dass keine benachbarten Knoten gleich gefärbt sind?

Lineare Reduktion von Graph $G = (V, E)$ auf SAT:

- Variablen = $V \times \{1, 2, 3\}$. Notation: x_{vi}
- Interpretation: $x_{vi} = 1$ gdw Knoten v hat Farbe i

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

3COL

Gegeben: Ein ungerichteter Graph

Problem: Gibt es eine Färbung der Knoten mit 3 Farben, so dass keine benachbarten Knoten gleich gefärbt sind?

Lineare Reduktion von Graph $G = (V, E)$ auf SAT:

- Variablen = $V \times \{1, 2, 3\}$. Notation: x_{vi}
- Interpretation: $x_{vi} = 1$ gdw Knoten v hat Farbe i

Instanz von SAT:

$$\bigwedge_{v \in V} \text{Genau1}(x_{v1}, x_{v2}, x_{v3})$$

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

3COL

Gegeben: Ein ungerichteter Graph

Problem: Gibt es eine Färbung der Knoten mit 3 Farben, so dass keine benachbarten Knoten gleich gefärbt sind?

Lineare Reduktion von Graph $G = (V, E)$ auf SAT:

- Variablen = $V \times \{1, 2, 3\}$. Notation: x_{vi}
- Interpretation: $x_{vi} = 1$ gdw Knoten v hat Farbe i

Instanz von SAT:

$$\bigwedge_{v \in V} \text{Genau1}(x_{v1}, x_{v2}, x_{v3}) \wedge \bigwedge_{(u,v) \in E} \neg(x_{u1} \wedge x_{v1} \vee x_{u2} \wedge x_{v2} \vee x_{u3} \wedge x_{v3})$$

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

3COL

Gegeben: Ein ungerichteter Graph

Problem: Gibt es eine Färbung der Knoten mit 3 Farben, so dass keine benachbarten Knoten gleich gefärbt sind?

Lineare Reduktion von Graph $G = (V, E)$ auf SAT:

- Variablen = $V \times \{1, 2, 3\}$. Notation: x_{vi}
- Interpretation: $x_{vi} = 1$ gdw Knoten v hat Farbe i

Instanz von SAT:

$$\bigwedge_{v \in V} \text{Genau1}(x_{v1}, x_{v2}, x_{v3}) \wedge \bigwedge_{(u,v) \in E} \neg(x_{u1} \wedge x_{v1} \vee x_{u2} \wedge x_{v2} \vee x_{u3} \wedge x_{v3})$$

Bemerkungen

- Zeigt $3\text{COL} \leq_p \text{SAT}$ und damit $3\text{COL} \in \text{NP}$.
- **Zeigt nicht, dass 3COL NP-vollständig ist.**

Ähnlich direkt polynomiell auf SAT reduzierbar:

- HAMILTON
- SUDOKU
- ...

Eine Anwendung von k -Färbbarkeit: **Registerverteilung**

*Kann man in einem Programmstück n Variablen so auf k Register verteilen, dass jede Variable so lange in einem Register bleibt, wie sie **lebendig** ist?*

Eine Anwendung von k -Färbbarkeit: **Registerverteilung**

*Kann man in einem Programmstück n Variablen so auf k Register verteilen, dass jede Variable so lange in einem Register bleibt, wie sie **lebendig** ist?*

Variable ist an einem Punkt **lebendig**

gdw sie später noch gelesen wird, ohne vorher überschrieben worden zu sein.

Eine Anwendung von k -Färbbarkeit: **Registerverteilung**

*Kann man in einem Programmstück n Variablen so auf k Register verteilen, dass jede Variable so lange in einem Register bleibt, wie sie **lebendig** ist?*

Variable ist an einem Punkt **lebendig**

gdw sie später noch gelesen wird, ohne vorher überschrieben worden zu sein.

Reduktion auf k -Färbbarkeit:

Variable = Knoten

u und v verbunden = $u \neq v$ und es gibt einen Programmpunkt, an dem u und v lebendig sind

Farbe = Register

k -Färbung = Zuordnung eines Registers zu jeder Variablen

Eine Anwendung von k -Färbbarkeit: **Registerverteilung**

*Kann man in einem Programmstück n Variablen so auf k Register verteilen, dass jede Variable so lange in einem Register bleibt, wie sie **lebendig** ist?*

Variable ist an einem Punkt **lebendig**

gdw sie später noch gelesen wird, ohne vorher überschrieben worden zu sein.

Reduktion auf k -Färbbarkeit:

Variable = Knoten

u und v verbunden = $u \neq v$ und es gibt einen Programmpunkt, an dem u und v lebendig sind

Farbe = Register

k -Färbung = Zuordnung eines Registers zu jeder Variablen

Sowohl k -Färbbarkeit als auch Registerverteilung ist für $k \geq 3$ NP-vollständig.

Mehr Information: Vorlesung *Programmoptimierung*

Weitere Anwendungen von SAT:

Bounded Model Checking

Weitere Anwendungen von SAT:

Bounded Model Checking

Hardware Entscheide, ob eine Schaltung mit Zustand für **alle** Eingaben innerhalb von 10 Zyklen ein bestimmtes Verhalten (nicht) hat.

Weitere Anwendungen von SAT:

Bounded Model Checking

Hardware Entscheide, ob eine Schaltung mit Zustand für **alle** Eingaben innerhalb von 10 Zyklen ein bestimmtes Verhalten (nicht) hat.

Software Entscheide, ob ein Programm für **alle** Eingaben innerhalb von 10 Schritten ein bestimmtes Verhalten (nicht) hat.

Weitere Anwendungen von SAT:

Bounded Model Checking

- Hardware** Entscheide, ob eine Schaltung mit Zustand für **alle** Eingaben innerhalb von 10 Zyklen ein bestimmtes Verhalten (nicht) hat.
- Software** Entscheide, ob ein Programm für **alle** Eingaben innerhalb von 10 Schritten ein bestimmtes Verhalten (nicht) hat.
Variablen müssen auf kleine Wertebereiche eingeschränkt werden!

6.4 Weitere NP-vollständige Probleme

6.4 Weitere NP-vollständige Probleme

Wie zeigt man, dass ein (weiteres) Problem B NP-vollständig ist?

6.4 Weitere NP-vollständige Probleme

Wie zeigt man, dass ein (weiteres) Problem B NP-vollständig ist?

- Zeige $B \in \text{NP}$ (meist trivial)
- Zeige $A \leq_p B$ für ein NP-vollständiges Problem A .

6.4 Weitere NP-vollständige Probleme

Wie zeigt man, dass ein (weiteres) Problem B NP-vollständig ist?

- Zeige $B \in \text{NP}$ (meist trivial)
- Zeige $A \leq_p B$ für ein NP-vollständiges Problem A .

Lemma 6.20

Ist A NP-vollständig, so ist $B \in \text{NP}$ ebenfalls NP-vollständig, falls $A \leq_p B$.

6.4 Weitere NP-vollständige Probleme

Wie zeigt man, dass ein (weiteres) Problem B NP-vollständig ist?

- Zeige $B \in \text{NP}$ (meist trivial)
- Zeige $A \leq_p B$ für ein NP-vollständiges Problem A .

Lemma 6.20

Ist A NP-vollständig, so ist $B \in \text{NP}$ ebenfalls NP-vollständig, falls $A \leq_p B$.

Beweis:

Folgt direkt aus der Transitivität von \leq_p :

B ist NP-hart, denn für $C \in \text{NP}$ gilt $C \leq_p A \leq_p B$. □

Warum will man wissen, dass ein Problem B NP-vollständig ist?

Warum will man wissen, dass ein Problem B NP-vollständig ist?
Um sicher zu sein, dass

- ein polynomieller Algorithmus für B ein Durchbruch wäre
- und daher wahrscheinlich nicht existiert.

Praktische Instanzen von B könnten trotzdem (zB mit SAT)
„effizient“ lösbar sein.

Definition 6.21

- Eine Formel ist in **Konjunktiver Normalform (KNF)** gdw sie eine Konjunktion von **Klauseln** ist: $K_1 \wedge \cdots \wedge K_n$
- Eine **Klausel** ist eine Disjunktion von **Literalen**: $L_1 \vee \cdots \vee L_m$
- Ein **Literal** ist eine (evtl. negierte) Variable.
- Eine Formel ist in **3KNF** gdw jede Klausel ≤ 3 Literale enthält.

Definition 6.21

- Eine Formel ist in **Konjunktiver Normalform (KNF)** gdw sie eine Konjunktion von **Klauseln** ist: $K_1 \wedge \cdots \wedge K_n$
- Eine **Klausel** ist eine Disjunktion von **Literalen**: $L_1 \vee \cdots \vee L_m$
- Ein **Literal** ist eine (evtl. negierte) Variable.
- Eine Formel ist in **3KNF** gdw jede Klausel ≤ 3 Literale enthält.

Dh eine KNF ist ein Konjunktion von Disjunktionen von (evtl negierten) Variablen.

Bsp: $(x_9 \vee \neg x_2) \wedge (\neg x_7 \vee x_1 \vee x_6)$

Definition 6.21

- Eine Formel ist in **Konjunktiver Normalform (KNF)** gdw sie eine Konjunktion von **Klauseln** ist: $K_1 \wedge \cdots \wedge K_n$
- Eine **Klausel** ist eine Disjunktion von **Literalen**: $L_1 \vee \cdots \vee L_m$
- Ein **Literal** ist eine (evtl. negierte) Variable.
- Eine Formel ist in **3KNF** gdw jede Klausel ≤ 3 Literale enthält.

Dh eine KNF ist ein Konjunktion von Disjunktionen von (evtl negierten) Variablen.

Bsp: $(x_9 \vee \neg x_2) \wedge (\neg x_7 \vee x_1 \vee x_6)$

3KNF-SAT

Gegeben: Ein Formel in 3KNF

Problem: Ist die Formel erfüllbar?

Definition 6.21

- Eine Formel ist in **Konjunktiver Normalform (KNF)** gdw sie eine Konjunktion von **Klauseln** ist: $K_1 \wedge \dots \wedge K_n$
- Eine **Klausel** ist eine Disjunktion von **Literalen**: $L_1 \vee \dots \vee L_m$
- Ein **Literal** ist eine (evtl. negierte) Variable.
- Eine Formel ist in **3KNF** gdw jede Klausel ≤ 3 Literale enthält.

Dh eine KNF ist ein Konjunktion von Disjunktionen von (evtl negierten) Variablen.

Bsp: $(x_9 \vee \neg x_2) \wedge (\neg x_7 \vee x_1 \vee x_6)$

3KNF-SAT

Gegeben: Ein Formel in 3KNF

Problem: Ist die Formel erfüllbar?

Offensichtlich gilt $3KNF-SAT \in NP$.

Aber vielleicht ist 3KNF-SAT einfacher als SAT?

Satz 6.22

3KNF-SAT ist NP-vollständig.

Satz 6.22

3KNF-SAT ist NP-vollständig.

Beweis:

Wir zeigen $\text{SAT} \leq_p \text{3KNF-SAT}$. Wir geben eine polynomielle Reduktion $F \mapsto F'$ an, so dass F' in 3KNF ist und

$$F \text{ ist erfüllbar} \Leftrightarrow F' \text{ ist erfüllbar}$$

Satz 6.22

3KNF-SAT ist NP-vollständig.

Beweis:

Wir zeigen $\text{SAT} \leq_p \text{3KNF-SAT}$. Wir geben eine polynomielle Reduktion $F \mapsto F'$ an, so dass F' in 3KNF ist und

$$F \text{ ist erfüllbar} \Leftrightarrow F' \text{ ist erfüllbar}$$

NB F und F' sind **erfüllbarkeitsäquivalent**, aber nicht notwendigerweise auch äquivalent.

Satz 6.22

3KNF-SAT ist NP-vollständig.

Beweis:

Wir zeigen $\text{SAT} \leq_p \text{3KNF-SAT}$. Wir geben eine polynomielle Reduktion $F \mapsto F'$ an, so dass F' in 3KNF ist und

$$F \text{ ist erfüllbar} \Leftrightarrow F' \text{ ist erfüllbar}$$

NB F und F' sind **erfüllbarkeitsäquivalent**, aber nicht notwendigerweise auch äquivalent.

1. Transformiere F in **Negations-Normalform (NNF)** durch fortgesetzte Anwendung der de Morganschen Gesetze

Satz 6.22

3KNF-SAT ist NP-vollständig.

Beweis:

Wir zeigen $\text{SAT} \leq_p \text{3KNF-SAT}$. Wir geben eine polynomielle Reduktion $F \mapsto F'$ an, so dass F' in 3KNF ist und

$$F \text{ ist erfüllbar} \Leftrightarrow F' \text{ ist erfüllbar}$$

NB F und F' sind **erfüllbarkeitsäquivalent**, aber nicht notwendigerweise auch äquivalent.

1. Transformiere F in **Negations-Normalform (NNF)** durch fortgesetzte Anwendung der de Morganschen Gesetze

$$\neg(A \wedge B) = \neg A \vee \neg B$$

$$\neg(A \vee B) = \neg A \wedge \neg B$$

Satz 6.22

3KNF-SAT ist NP-vollständig.

Beweis:

Wir zeigen $\text{SAT} \leq_p \text{3KNF-SAT}$. Wir geben eine polynomielle Reduktion $F \mapsto F'$ an, so dass F' in 3KNF ist und

$$F \text{ ist erfüllbar} \Leftrightarrow F' \text{ ist erfüllbar}$$

NB F und F' sind **erfüllbarkeitsäquivalent**, aber nicht notwendigerweise auch äquivalent.

1. Transformiere F in **Negations-Normalform (NNF)** durch fortgesetzte Anwendung der de Morganschen Gesetze

$$\neg(A \wedge B) = \neg A \vee \neg B$$

$$\neg(A \vee B) = \neg A \wedge \neg B$$

$$\neg\neg A = A$$

Satz 6.22

3KNF-SAT ist NP-vollständig.

Beweis:

Wir zeigen $\text{SAT} \leq_p \text{3KNF-SAT}$. Wir geben eine polynomielle Reduktion $F \mapsto F'$ an, so dass F' in 3KNF ist und

$$F \text{ ist erfüllbar} \Leftrightarrow F' \text{ ist erfüllbar}$$

NB F und F' sind **erfüllbarkeitsäquivalent**, aber nicht notwendigerweise auch äquivalent.

1. Transformiere F in **Negations-Normalform (NNF)** durch fortgesetzte Anwendung der de Morganschen Gesetze

$$\neg(A \wedge B) = \neg A \vee \neg B$$

$$\neg(A \vee B) = \neg A \wedge \neg B$$

$$\neg\neg A = A$$

von links nach rechts.

Satz 6.22

3KNF-SAT ist NP-vollständig.

Beweis:

Wir zeigen $\text{SAT} \leq_p \text{3KNF-SAT}$. Wir geben eine polynomielle Reduktion $F \mapsto F'$ an, so dass F' in 3KNF ist und

$$F \text{ ist erfüllbar} \Leftrightarrow F' \text{ ist erfüllbar}$$

NB F und F' sind **erfüllbarkeitsäquivalent**, aber nicht notwendigerweise auch äquivalent.

1. Transformiere F in **Negations-Normalform (NNF)** durch fortgesetzte Anwendung der de Morganschen Gesetze

$$\neg(A \wedge B) = \neg A \vee \neg B$$

$$\neg(A \vee B) = \neg A \wedge \neg B$$

$$\neg\neg A = A$$

von links nach rechts. F_1 ist Resultat.

Beweis (Forts.):

Für F_1 gilt: \neg nur noch direkt vor Variablen.

Beweis (Forts.):

Für F_1 gilt: \neg nur noch direkt vor Variablen.

2. Betrachte F_1 als Baum, wobei die Literale Blätter sind.
Ordne jedem inneren Knoten eine neue Variable $\in \{y_0, y_1, \dots\}$ zu.
Ordne dabei der Wurzel von F_1 die Variable y_0 zu.

3. Betrachte die y_i als Abkürzung für die Teilbäume, an deren Wurzeln sie stehen

$$y_i = \begin{array}{c} \circ_i \\ / \quad \backslash \\ l_i \quad r_i \end{array}$$

wobei $\circ_i \in \{\wedge, \vee\}$ und l_i, r_i ein Literal oder eine Variable y_j ist.

Beweis (Forts.):

Für F_1 gilt: \neg nur noch direkt vor Variablen.

2. Betrachte F_1 als Baum, wobei die Literale Blätter sind.
Ordne jedem inneren Knoten eine neue Variable $\in \{y_0, y_1, \dots\}$ zu.
Ordne dabei der Wurzel von F_1 die Variable y_0 zu.

3. Betrachte die y_i als Abkürzung für die Teilbäume, an deren Wurzeln sie stehen

$$y_i = \begin{array}{c} \circ_i \\ / \quad \backslash \\ l_i \quad r_i \end{array}$$

wobei $\circ_i \in \{\wedge, \vee\}$ und l_i, r_i ein Literal oder eine Variable y_j ist.
Beschreibe F_1 Knoten für Knoten:

$$y_0 \wedge (y_0 \leftrightarrow (l_0 \circ_0 r_0)) \wedge (y_1 \leftrightarrow (l_1 \circ_1 r_1)) \dots$$

Beweis (Forts.):

Für F_1 gilt: \neg nur noch direkt vor Variablen.

2. Betrachte F_1 als Baum, wobei die Literale Blätter sind.
Ordne jedem inneren Knoten eine neue Variable $\in \{y_0, y_1, \dots\}$ zu.
Ordne dabei der Wurzel von F_1 die Variable y_0 zu.

3. Betrachte die y_i als Abkürzung für die Teilbäume, an deren Wurzeln sie stehen

$$y_i = \begin{array}{c} \circ_i \\ / \quad \backslash \\ l_i \quad r_i \end{array}$$

wobei $\circ_i \in \{\wedge, \vee\}$ und l_i, r_i ein Literal oder eine Variable y_j ist.
Beschreibe F_1 Knoten für Knoten:

$$y_0 \wedge (y_0 \leftrightarrow (l_0 \circ_0 r_0)) \wedge (y_1 \leftrightarrow (l_1 \circ_1 r_1)) \dots =: F_2$$

Beweis (Forts.):

F_1 erf. $\implies F_2$ erf.: y_i bekommt Wert seines Teilbaums.

F_2 erf. $\implies F_1$ erf.: klar

Beweis (Forts.):

F_1 erf. $\implies F_2$ erf.: y_i bekommt Wert seines Teilbaums.

F_2 erf. $\implies F_1$ erf.: klar

4. Transformiere jede Äquivalenz in 3KNF:

$$(a \leftrightarrow (b \vee c)) \quad \mapsto \quad (a \vee \neg b) \wedge (a \vee \neg c) \wedge (\neg a \vee b \vee c)$$

$$(a \leftrightarrow (b \wedge c)) \quad \mapsto \quad (\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg b \vee \neg c)$$

Beweis (Forts.):

F_1 erf. $\implies F_2$ erf.: y_i bekommt Wert seines Teilbaums.

F_2 erf. $\implies F_1$ erf.: klar

4. Transformiere jede Äquivalenz in 3KNF:

$$(a \leftrightarrow (b \vee c)) \mapsto (a \vee \neg b) \wedge (a \vee \neg c) \wedge (\neg a \vee b \vee c)$$

$$(a \leftrightarrow (b \wedge c)) \mapsto (\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg b \vee \neg c)$$

Ergebnis ist F' .

Beweis (Forts.):

F_1 erf. $\implies F_2$ erf.: y_i bekommt Wert seines Teilbaums.

F_2 erf. $\implies F_1$ erf.: klar

4. Transformiere jede Äquivalenz in 3KNF:

$$(a \leftrightarrow (b \vee c)) \mapsto (a \vee \neg b) \wedge (a \vee \neg c) \wedge (\neg a \vee b \vee c)$$

$$(a \leftrightarrow (b \wedge c)) \mapsto (\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg b \vee \neg c)$$

Ergebnis ist F' .

Jeder Schritt ist in polynomieller Zeit in $|F|$ berechenbar.

Beweis (Forts.):

F_1 erf. $\implies F_2$ erf.: y_i bekommt Wert seines Teilbaums.

F_2 erf. $\implies F_1$ erf.: klar

4. Transformiere jede Äquivalenz in 3KNF:

$$(a \leftrightarrow (b \vee c)) \mapsto (a \vee \neg b) \wedge (a \vee \neg c) \wedge (\neg a \vee b \vee c)$$

$$(a \leftrightarrow (b \wedge c)) \mapsto (\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg b \vee \neg c)$$

Ergebnis ist F' .

Jeder Schritt ist in polynomieller Zeit in $|F|$ berechenbar.

Bei Transformation in NNF nimmt mit jedem Schritt

Summe der $|G|$ für alle Teilformeln $\neg G$ von F

ab.

Daher erreicht man die NNF in $\leq |F|^2$ Schritten.



Da jede Formel in 3KNF auch in KNF ist:

Korollar 6.23

KNF-SAT ist NP-vollständig.

Da jede Formel in 3KNF auch in KNF ist:

Korollar 6.23

KNF-SAT ist NP-vollständig.

Kann man wie folgt die NP-Vollständigkeit von KNF-SAT zeigen?

Man zeigt $\text{SAT} \leq_p \text{KNF-SAT}$

indem man jede Formel in KNF transformiert.

Da jede Formel in 3KNF auch in KNF ist:

Korollar 6.23

KNF-SAT ist NP-vollständig.

Kann man wie folgt die NP-Vollständigkeit von KNF-SAT zeigen?

Man zeigt $SAT \leq_p KNF-SAT$

indem man jede Formel in KNF transformiert.

Satz 6.24

$2KNF-SAT \in P$

Ohne Beweis

MENGENÜBERDECKUNG (MÜ)

Gegeben: Teilmengen $T_1, \dots, T_n \subseteq M$ einer endlichen Menge M
und eine Zahl $k \leq n$.

Problem: Gibt es $i_1, \dots, i_k \in \{1, \dots, n\}$ mit $M = T_{i_1} \cup \dots \cup T_{i_k}$?

MENGENÜBERDECKUNG (MÜ)

Gegeben: Teilmengen $T_1, \dots, T_n \subseteq M$ einer endlichen Menge M und eine Zahl $k \leq n$.

Problem: Gibt es $i_1, \dots, i_k \in \{1, \dots, n\}$ mit $M = T_{i_1} \cup \dots \cup T_{i_k}$?

Beispiel 6.25

$$\begin{array}{ll} T_1 = \{1, 2\} & T_2 = \{1, 3\} \\ T_3 = \{3, 4\} & T_4 = \{3, 5\} \\ M = \{1, 2, 3, 4, 5\} & k = 3 \end{array}$$

Lösung: Ja

Anwendung: Zulieferer

M Menge der Teile, die eine Firma einkaufen muss

T_i Menge der Teile, die Zulieferer i anbietet

Kann die Firma ihre Bedürfnisse mit k Zulieferern abdecken?

MENGENÜBERDECKUNG (MÜ)

Gegeben: Teilmengen $T_1, \dots, T_n \subseteq M$ einer endlichen Menge M und eine Zahl $k \leq n$.

Problem: Gibt es $i_1, \dots, i_k \in \{1, \dots, n\}$ mit $M = T_{i_1} \cup \dots \cup T_{i_k}$?

Beispiel 6.25

$$\begin{array}{ll} T_1 = \{1, 2\} & T_2 = \{1, 3\} \\ T_3 = \{3, 4\} & T_4 = \{3, 5\} \\ M = \{1, 2, 3, 4, 5\} & k = 3 \end{array}$$

Lösung: Ja

Anwendung: Zulieferer

M Menge der Teile, die eine Firma einkaufen muss

T_i Menge der Teile, die Zulieferer i anbietet

Kann die Firma ihre Bedürfnisse mit k Zulieferern abdecken?

Fakt 6.26

$MÜ \in NP$.

Satz 6.27

MÜ ist NP-vollständig.

Satz 6.27

MÜ ist NP-vollständig.

Beweis:

Wir zeigen $\text{KNF-SAT} \leq_p \text{MÜ}$.

Satz 6.27

MÜ ist NP-vollständig.

Beweis:

Wir zeigen $\text{KNF-SAT} \leq_p \text{MÜ}$.

Sei $F = K_1 \wedge \dots \wedge K_m$ in KNF, mit Variablen x_1, \dots, x_k .

Satz 6.27

MÜ ist NP-vollständig.

Beweis:

Wir zeigen $\text{KNF-SAT} \leq_p \text{MÜ}$.

Sei $F = K_1 \wedge \dots \wedge K_m$ in KNF, mit Variablen x_1, \dots, x_k .

$$M := \{1, \dots, m, m+1, \dots, m+k\}$$

$$T_i := \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\}$$

$$T'_i := \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\}$$

Satz 6.27

$M\ddot{U}$ ist NP-vollstandig.

Beweis:

Wir zeigen $\text{KNF-SAT} \leq_p M\ddot{U}$.

Sei $F = K_1 \wedge \dots \wedge K_m$ in KNF, mit Variablen x_1, \dots, x_k .

$$M := \{1, \dots, m, m+1, \dots, m+k\}$$

$$T_i := \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\}$$

$$T'_i := \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\}$$

F ist erfulbar

gdw

M wird durch k der Teilmengen $T_1, \dots, T_k, T'_1, \dots, T'_k$ uberdeckt



Beweis (Forts.):

Sei $F = K_1 \wedge \dots \wedge K_m$ in KNF, mit Variablen x_1, \dots, x_l .

$$M := \{1, \dots, m\} \cup \{m+1, \dots, m+l\}$$

$$T_i := \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\}$$

$$T'_i := \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\}$$

Beweis (Forts.):

Sei $F = K_1 \wedge \dots \wedge K_m$ in KNF, mit Variablen x_1, \dots, x_l .

$$M := \{1, \dots, m\} \cup \{m+1, \dots, m+l\}$$

$$T_i := \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\}$$

$$T'_i := \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\}$$

Beispiel: $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

Beweis (Forts.):

Sei $F = K_1 \wedge \dots \wedge K_m$ in KNF, mit Variablen x_1, \dots, x_l .

$$M := \{1, \dots, m\} \cup \{m+1, \dots, m+l\}$$

$$T_i := \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\}$$

$$T'_i := \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\}$$

Beispiel: $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

\implies

$$T_1 = \{1, 2+1\} \quad T'_1 = \{2, 2+1\}$$

Beweis (Forts.):

Sei $F = K_1 \wedge \dots \wedge K_m$ in KNF, mit Variablen x_1, \dots, x_l .

$$M := \{1, \dots, m\} \cup \{m+1, \dots, m+l\}$$

$$T_i := \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\}$$

$$T'_i := \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\}$$

Beispiel: $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

\implies

$$\begin{array}{ll} T_1 = \{1, 2+1\} & T'_1 = \{2, 2+1\} \\ T_2 = \{1, 2, 2+2\} & T'_2 = \{2+2\} \end{array}$$

Beweis (Forts.):

Sei $F = K_1 \wedge \dots \wedge K_m$ in KNF, mit Variablen x_1, \dots, x_l .

$$M := \{1, \dots, m\} \cup \{m+1, \dots, m+l\}$$

$$T_i := \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\}$$

$$T'_i := \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\}$$

Beispiel: $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

\implies

$$T_1 = \{1, 2+1\} \quad T'_1 = \{2, 2+1\}$$

$$T_2 = \{1, 2, 2+2\} \quad T'_2 = \{2+2\}$$

$$T_3 = \{1, 2+3\} \quad T'_3 = \{2+3\}$$

Beweis (Forts.):

Sei $F = K_1 \wedge \dots \wedge K_m$ in KNF, mit Variablen x_1, \dots, x_l .

$$M := \{1, \dots, m\} \cup \{m+1, \dots, m+l\}$$

$$T_i := \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\}$$

$$T'_i := \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\}$$

Beispiel: $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

\implies

$$T_1 = \{1, 2+1\} \quad T'_1 = \{2, 2+1\}$$

$$T_2 = \{1, 2, 2+2\} \quad T'_2 = \{2+2\}$$

$$T_3 = \{1, 2+3\} \quad T'_3 = \{2+3\}$$

$$T_4 = \{2, 2+4\} \quad T'_4 = \{2+4\}$$

Beweis (Forts.):

Sei $F = K_1 \wedge \dots \wedge K_m$ in KNF, mit Variablen x_1, \dots, x_l .

$$M := \{1, \dots, m\} \cup \{m+1, \dots, m+l\}$$

$$T_i := \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\}$$

$$T'_i := \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\}$$

Beispiel: $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

\implies

$$\begin{array}{ll} T_1 = \{1, 2+1\} & T'_1 = \{2, 2+1\} \\ T_2 = \{1, 2, 2+2\} & T'_2 = \{2+2\} \\ T_3 = \{1, 2+3\} & T'_3 = \{2+3\} \\ T_4 = \{2, 2+4\} & T'_4 = \{2+4\} \end{array}$$

Überdeckung: T'_1, T_2, T'_3, T'_4

Beweis (Forts.):

Sei $F = K_1 \wedge \dots \wedge K_m$ in KNF, mit Variablen x_1, \dots, x_l .

$$M := \{1, \dots, m\} \cup \{m+1, \dots, m+l\}$$

$$T_i := \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\}$$

$$T'_i := \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\}$$

Beispiel: $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

\implies

$$T_1 = \{1, 2+1\} \quad T'_1 = \{2, 2+1\}$$

$$T_2 = \{1, 2, 2+2\} \quad T'_2 = \{2+2\}$$

$$T_3 = \{1, 2+3\} \quad T'_3 = \{2+3\}$$

$$T_4 = \{2, 2+4\} \quad T'_4 = \{2+4\}$$

Überdeckung: $T'_1, T_2, T'_3, T'_4 \approx x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 0$

Beweis (Forts.):

Sei $F = K_1 \wedge \dots \wedge K_m$ in KNF, mit Variablen x_1, \dots, x_l .

$$M := \{1, \dots, m\} \cup \{m+1, \dots, m+l\}$$

$$T_i := \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\}$$

$$T'_i := \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\}$$

Beweis (Forts.):

Sei $F = K_1 \wedge \dots \wedge K_m$ in KNF, mit Variablen x_1, \dots, x_l .

$$M := \{1, \dots, m\} \cup \{m+1, \dots, m+l\}$$

$$T_i := \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\}$$

$$T'_i := \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\}$$

„ \Leftarrow “ Sei $U_1, \dots, U_n \in \{T_1, \dots, T'_n\}$ eine Überdeckung von M .

Beweis (Forts.):

Sei $F = K_1 \wedge \dots \wedge K_m$ in KNF, mit Variablen x_1, \dots, x_l .

$$M := \{1, \dots, m\} \cup \{m+1, \dots, m+l\}$$

$$T_i := \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\}$$

$$T'_i := \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\}$$

„ \Leftarrow “ Sei $U_1, \dots, U_n \in \{T_1, \dots, T'_n\}$ eine Überdeckung vom M .

\Rightarrow Es gibt für alle $1 \leq i \leq n$ genau eine Menge $U_i \in \{T_i, T'_i\}$ mit $m+i \in U_i$.

Beweis (Forts.):

Sei $F = K_1 \wedge \dots \wedge K_m$ in KNF, mit Variablen x_1, \dots, x_l .

$$M := \{1, \dots, m\} \cup \{m+1, \dots, m+l\}$$

$$T_i := \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\}$$

$$T'_i := \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\}$$

„ \Leftarrow “ Sei $U_1, \dots, U_n \in \{T_1, \dots, T'_n\}$ eine Überdeckung von M .

\Rightarrow Es gibt für alle $1 \leq i \leq n$ genau eine Menge $U_i \in \{T_i, T'_i\}$ mit $m+i \in U_i$.

Setze $\sigma(x_i) := \text{if } U_i = T_i \text{ then } 1 \text{ else } 0$

Beweis (Forts.):

Sei $F = K_1 \wedge \dots \wedge K_m$ in KNF, mit Variablen x_1, \dots, x_l .

$$M := \{1, \dots, m\} \cup \{m+1, \dots, m+l\}$$

$$T_i := \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\}$$

$$T'_i := \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\}$$

„ \Leftarrow “ Sei $U_1, \dots, U_n \in \{T_1, \dots, T'_n\}$ eine Überdeckung von M .

\Rightarrow Es gibt für alle $1 \leq i \leq n$ genau eine Menge $U_i \in \{T_i, T'_i\}$ mit $m+i \in U_i$.

Setze $\sigma(x_i) := \text{if } U_i = T_i \text{ then } 1 \text{ else } 0$

$\Rightarrow \sigma(K_j) = 1$ für alle $1 \leq j \leq m$

Beweis (Forts.):

Sei $F = K_1 \wedge \dots \wedge K_m$ in KNF, mit Variablen x_1, \dots, x_l .

$$M := \{1, \dots, m\} \cup \{m+1, \dots, m+l\}$$

$$T_i := \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\}$$

$$T'_i := \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\}$$

„ \Leftarrow “ Sei $U_1, \dots, U_n \in \{T_1, \dots, T'_n\}$ eine Überdeckung von M .

\Rightarrow Es gibt für alle $1 \leq i \leq n$ genau eine Menge $U_i \in \{T_i, T'_i\}$ mit $m+i \in U_i$.

Setze $\sigma(x_i) := \text{if } U_i = T_i \text{ then } 1 \text{ else } 0$

$\Rightarrow \sigma(K_j) = 1$ für alle $1 \leq j \leq m$

$\Rightarrow \sigma(F) = 1$



Das **Minimierungsproblem**

Gegeben: Teilmengen $T_1, \dots, T_n \subseteq M$ einer endlichen Menge M

Problem: Finde das kleinste k , so dass M von k der Teilmengen überdeckt wird.

kann auf das Entscheidungsproblem reduziert werden:

Das **Minimierungsproblem**

Gegeben: Teilmengen $T_1, \dots, T_n \subseteq M$ einer endlichen Menge M

Problem: Finde das kleinste k , so dass M von k der Teilmengen überdeckt wird.

kann auf das Entscheidungsproblem reduziert werden:

Finde kleinstes k durch binäre Suche im Intervall $[1, n]$ mit $O(\log n)$ Aufrufen von MÜ.

Das **Minimierungsproblem**

Gegeben: Teilmengen $T_1, \dots, T_n \subseteq M$ einer endlichen Menge M

Problem: Finde das kleinste k , so dass M von k der Teilmengen überdeckt wird.

kann auf das Entscheidungsproblem reduziert werden:

Finde kleinstes k durch binäre Suche im Intervall $[1, n]$
mit $O(\log n)$ Aufrufen von MÜ.

Kann man MÜ in Zeit $O(f(n))$ entscheiden,
dann kann man das kleinste k in Zeit $O(f(n) \cdot \log n)$ berechnen.

Das **Minimierungsproblem**

Gegeben: Teilmengen $T_1, \dots, T_n \subseteq M$ einer endlichen Menge M

Problem: Finde das kleinste k , so dass M von k der Teilmengen überdeckt wird.

kann auf das Entscheidungsproblem reduziert werden:

Finde kleinstes k durch binäre Suche im Intervall $[1, n]$
mit $O(\log n)$ Aufrufen von MÜ.

Kann man MÜ in Zeit $O(f(n))$ entscheiden,
dann kann man das kleinste k in Zeit $O(f(n) \cdot \log n)$ berechnen.

$$\begin{aligned} f(n) \text{ polynomiell} &\implies f(n) \cdot \log n \text{ polynomiell} \\ f(n) \text{ exponentiell} &\implies f(n) \cdot \log n \text{ exponentiell} \end{aligned}$$

Die Berechnung einer **Lösung**

Gegeben: Teilmengen $\vec{T} := T_1, \dots, T_n \subseteq M$ einer endlichen Menge M , und eine Zahl $k \leq n$.

Problem: Finde eine Überdeckung von M durch k der Teilmengen.

kann auf das Entscheidungsproblem reduziert werden:

Die Berechnung einer **Lösung**

Gegeben: Teilmengen $\vec{T} := T_1, \dots, T_n \subseteq M$ einer endlichen Menge M , und eine Zahl $k \leq n$.

Problem: Finde eine Überdeckung von M durch k der Teilmengen.

kann auf das Entscheidungsproblem reduziert werden:

if $(\vec{T}, M, k) \notin \text{MÜ}$ **then** output(“nicht lösbar”)

else

$\ddot{U} := \emptyset$

for $i := 1$ **to** n **do**

if $(\vec{T} - T_i, M, k) \in \text{MÜ}$

then $\vec{T} := \vec{T} - T_i$

else $\ddot{U} := \ddot{U} \cup \{T_i\}$

CLIQUE

Gegeben: Ungerichteter Graph $G = (V, E)$ und Zahl $k \in \mathbb{N}$.

Problem: Besitzt G eine „Clique“ der Größe mindestens k ?

CLIQUE

Gegeben: Ungerichteter Graph $G = (V, E)$ und Zahl $k \in \mathbb{N}$.

Problem: Besitzt G eine „Clique“ der Größe mindestens k ?
Dh eine Teilmenge $V' \subseteq V$ mit $|V'| \geq k$
und alle $u, v \in V'$ mit $u \neq v$ sind benachbart.

CLIQUE

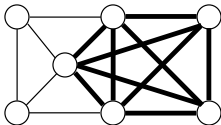
Gegeben: Ungerichteter Graph $G = (V, E)$ und Zahl $k \in \mathbb{N}$.

Problem: Besitzt G eine „Clique“ der Größe mindestens k ?

Dh eine Teilmenge $V' \subseteq V$ mit $|V'| \geq k$

und alle $u, v \in V'$ mit $u \neq v$ sind benachbart.

Beispiel mit 5-Clique:

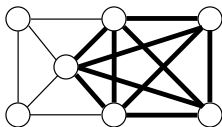


CLIQUE

Gegeben: Ungerichteter Graph $G = (V, E)$ und Zahl $k \in \mathbb{N}$.

Problem: Besitzt G eine „Clique“ der Größe mindestens k ?
Dh eine Teilmenge $V' \subseteq V$ mit $|V'| \geq k$
und alle $u, v \in V'$ mit $u \neq v$ sind benachbart.

Beispiel mit 5-Clique:



Anwendung von Clique-Berechnung:

Knoten = Prozess

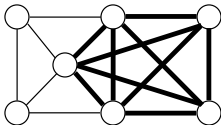
Kante = Zwei Prozesse sind parallel ausführbar

CLIQUE

Gegeben: Ungerichteter Graph $G = (V, E)$ und Zahl $k \in \mathbb{N}$.

Problem: Besitzt G eine „Clique“ der Größe mindestens k ?
Dh eine Teilmenge $V' \subseteq V$ mit $|V'| \geq k$
und alle $u, v \in V'$ mit $u \neq v$ sind benachbart.

Beispiel mit 5-Clique:



Anwendung von Clique-Berechnung:

Knoten = Prozess

Kante = Zwei Prozesse sind parallel ausführbar

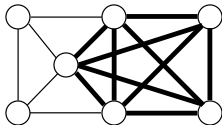
\implies Clique ist Gruppe von parallelisierbaren Prozessen

CLIQUE

Gegeben: Ungerichteter Graph $G = (V, E)$ und Zahl $k \in \mathbb{N}$.

Problem: Besitzt G eine „Clique“ der Größe mindestens k ?
Dh eine Teilmenge $V' \subseteq V$ mit $|V'| \geq k$
und alle $u, v \in V'$ mit $u \neq v$ sind benachbart.

Beispiel mit 5-Clique:



Anwendung von Clique-Berechnung:

Knoten = Prozess

Kante = Zwei Prozesse sind parallel ausführbar

\implies Clique ist Gruppe von parallelisierbaren Prozessen

Fakt 6.28

$CLIQUE \in NP$

Satz 6.29

CLIQUE ist NP-vollständig.

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT \leq_p CLIQUE:

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT \leq_p CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \dots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT \leq_p CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \dots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

wird auf einen Graphen abgebildet:

$$V = \{(1, 1), (1, 2), (1, 3), \dots, (k, 1), (k, 2), (k, 3)\}$$

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT \leq_p CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \dots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

wird auf einen Graphen abgebildet:

$$V = \{(1, 1), (1, 2), (1, 3), \dots, (k, 1), (k, 2), (k, 3)\}$$

$$E = \{(i, j), (p, q) \mid i \neq p \text{ und } z_{ij} \neq \neg z_{pq}\}$$

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT \leq_p CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \dots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

wird auf einen Graphen abgebildet:

$$V = \{(1, 1), (1, 2), (1, 3), \dots, (k, 1), (k, 2), (k, 3)\}$$

$$E = \{\{(i, j), (p, q)\} \mid i \neq p \text{ und } z_{ij} \neq \neg z_{pq}\}$$

F ist erfüllbar durch eine Belegung $\sigma \iff$

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT \leq_p CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \dots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

wird auf einen Graphen abgebildet:

$$V = \{(1, 1), (1, 2), (1, 3), \dots, (k, 1), (k, 2), (k, 3)\}$$

$$E = \{\{(i, j), (p, q)\} \mid i \neq p \text{ und } z_{ij} \neq \neg z_{pq}\}$$

F ist erfüllbar durch eine Belegung $\sigma \iff$

Es gibt in jeder Klausel i ein Literal z_{ij_i} mit $\sigma(z_{ij_i}) = 1$

\iff

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT \leq_p CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \dots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

wird auf einen Graphen abgebildet:

$$V = \{(1, 1), (1, 2), (1, 3), \dots, (k, 1), (k, 2), (k, 3)\}$$

$$E = \{\{(i, j), (p, q)\} \mid i \neq p \text{ und } z_{ij} \neq \neg z_{pq}\}$$

F ist erfüllbar durch eine Belegung $\sigma \iff$

Es gibt in jeder Klausel i ein Literal z_{ij_i} mit $\sigma(z_{ij_i}) = 1$

\iff

Die Literale $z_{1j_1}, \dots, z_{kj_k}$ sind paarweise nicht komplementär

\iff

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT \leq_p CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \dots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

wird auf einen Graphen abgebildet:

$$V = \{(1, 1), (1, 2), (1, 3), \dots, (k, 1), (k, 2), (k, 3)\}$$

$$E = \{\{(i, j), (p, q)\} \mid i \neq p \text{ und } z_{ij} \neq \neg z_{pq}\}$$

F ist erfüllbar durch eine Belegung $\sigma \iff$

Es gibt in jeder Klausel i ein Literal z_{ij_i} mit $\sigma(z_{ij_i}) = 1$

\iff

Die Literale $z_{1j_1}, \dots, z_{kj_k}$ sind paarweise nicht komplementär

\iff

Die Knoten $(1, j_1), \dots, (k, j_k)$ sind paarweise benachbart

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT \leq_p CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \dots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

wird auf einen Graphen abgebildet:

$$V = \{(1, 1), (1, 2), (1, 3), \dots, (k, 1), (k, 2), (k, 3)\}$$

$$E = \{\{(i, j), (p, q)\} \mid i \neq p \text{ und } z_{ij} \neq \neg z_{pq}\}$$

F ist erfüllbar durch eine Belegung $\sigma \iff$

Es gibt in jeder Klausel i ein Literal z_{ij_i} mit $\sigma(z_{ij_i}) = 1$

\iff

Die Literale $z_{1j_1}, \dots, z_{kj_k}$ sind paarweise nicht komplementär

\iff

Die Knoten $(1, j_1), \dots, (k, j_k)$ sind paarweise benachbart

$\iff G$ hat eine Clique der Größe k . □

RUCKSACK

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$ und $b \in \mathbb{N}$.

Problem: Gibt es $R \subseteq \{1, \dots, n\}$ mit $\sum_{i \in R} a_i = b$?

RUCKSACK

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$ und $b \in \mathbb{N}$.

Problem: Gibt es $R \subseteq \{1, \dots, n\}$ mit $\sum_{i \in R} a_i = b$?

Satz 6.30

RUCKSACK ist NP-vollständig.

RUCKSACK

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$ und $b \in \mathbb{N}$.

Problem: Gibt es $R \subseteq \{1, \dots, n\}$ mit $\sum_{i \in R} a_i = b$?

Satz 6.30

RUCKSACK ist NP-vollständig.

Beweis: 3KNF-SAT \leq_p RUCKSACK:

RUCKSACK

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$ und $b \in \mathbb{N}$.

Problem: Gibt es $R \subseteq \{1, \dots, n\}$ mit $\sum_{i \in R} a_i = b$?

Satz 6.30

RUCKSACK ist NP-vollständig.

Beweis: 3KNF-SAT \leq_p RUCKSACK:

Sei $F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \dots \wedge (z_{m1} \vee z_{m2} \vee z_{m3})$, wobei

$$z_{ij} \in \{x_1, \dots, x_n\} \cup \{\neg x_1, \neg x_2, \dots, \neg x_n\}$$

D.h. m = Anzahl der Klauseln und n Anzahl der Variablen.

Beweis (Forts.):

Wir geben Zahlen v_{i0}, v_{i1} für jede variable x_i , Zahlen k_{j1}, k_{j2} für jede Klausel K_j , und eine Zahl b an.

Beweis (Forts.):

Wir geben Zahlen v_{i0}, v_{i1} für jede variable x_i , Zahlen k_{j1}, k_{j2} für jede Klausel K_j , und eine Zahl b an.

- Alle Zahlen haben genau $m + n$ Stellen im Dezimalsystem.
Die j -te Stelle, $1 \leq j \leq m$, nennen wir
"Stelle (der Klausel) K_j ".
Die $m + i$ Stelle, $1 \leq i \leq n$, nennen wir
"Stelle (der Variable) x_i ".

Beweis (Forts.):

Wir geben Zahlen v_{i0}, v_{i1} für jede variable x_i , Zahlen k_{j1}, k_{j2} für jede Klausel K_j , und eine Zahl b an.

- Alle Zahlen haben genau $m + n$ Stellen im Dezimalsystem.

Die j -te Stelle, $1 \leq j \leq m$, nennen wir
"Stelle (der Klausel) K_j ".

Die $m + i$ Stelle, $1 \leq i \leq n$, nennen wir
"Stelle (der Variable) x_i ".

- $b = \underbrace{44 \dots 444}_m \underbrace{11 \dots 11}_n$

Beweis (Forts.):

Wir geben Zahlen v_{i0}, v_{i1} für jede variable x_i , Zahlen k_{j1}, k_{j2} für jede Klausel K_j , und eine Zahl b an.

- Alle Zahlen haben genau $m + n$ Stellen im Dezimalsystem. Die j -te Stelle, $1 \leq j \leq m$, nennen wir "Stelle (der Klausel) K_j ". Die $m + i$ Stelle, $1 \leq i \leq n$, nennen wir "Stelle (der Variable) x_i ".
- $b = \underbrace{44 \dots 44}_m \underbrace{11 \dots 11}_n$
- An der Stelle x_i haben v_{i0}, v_{i1} eine 1, und die Zahlen und k_{j1}, k_{j2} eine 0.
 - genau eine von v_{i0}, v_{i1} muss in den Rucksack
 - modelliert die Wahl einer Belegung.

Beweis (Forts.):

Beschreibung der Zahlen $v_{i0}, v_{i1}, k_{j1}, k_{j2}$:

- v_{i0} hat eine 1 an der Stelle x_i sowie an den Stellen der Klauseln, die $\neg x_i$ enthalten, sonst 0en.
- v_{i1} hat eine 1 an der Stelle x_i sowie an den Stellen der Klauseln, die x_i enthalten, sonst 0en.

Beweis (Forts.):

Beschreibung der Zahlen $v_{i0}, v_{i1}, k_{j1}, k_{j2}$:

- v_{i0} hat eine 1 an der Stelle x_i sowie an den Stellen der Klauseln, die $\neg x_i$ enthalten, sonst 0en.
- v_{i1} hat eine 1 an der Stelle x_i sowie an den Stellen der Klauseln, die x_i enthalten, sonst 0en.
- k_{j1} hat eine 1 an der Stelle K_j , sonst 0en.
- k_{j2} hat eine 2 an der Stelle K_j , sonst 0en.

Beweis (Forts.):

Beschreibung der Zahlen $v_{i0}, v_{i1}, k_{j1}, k_{j2}$:

- v_{i0} hat eine 1 an der Stelle x_i sowie an den Stellen der Klauseln, die $\neg x_i$ enthalten, sonst 0en.
- v_{i1} hat eine 1 an der Stelle x_i sowie an den Stellen der Klauseln, die x_i enthalten, sonst 0en.
- k_{j1} hat eine 1 an der Stelle K_j , sonst 0en.
- k_{j2} hat eine 2 an der Stelle K_j , sonst 0en.

Wenn die Summe einer Untermenge R die Zahl b ergibt, dann erfüllt die Belegung σ mit $\sigma(x_i) = 1$ gdw $v_{i1} \in R$ die Formel F .

Beweis (Forts.):

Beschreibung der Zahlen $v_{i0}, v_{i1}, k_{j1}, k_{j2}$:

- v_{i0} hat eine 1 an der Stelle x_i sowie an den Stellen der Klauseln, die $\neg x_i$ enthalten, sonst 0en.
- v_{i1} hat eine 1 an der Stelle x_i sowie an den Stellen der Klauseln, die x_i enthalten, sonst 0en.
- k_{j1} hat eine 1 an der Stelle K_j , sonst 0en.
- k_{j2} hat eine 2 an der Stelle K_j , sonst 0en.

Wenn die Summe einer Untermenge R die Zahl b ergibt, dann erfüllt die Belegung σ mit $\sigma(x_i) = 1$ gdw $v_{i1} \in R$ die Formel F .

Wenn F erfüllbar ist, dann wähle R so:

- Nehme eine erfüllende Belegung σ von F .
- Für jede Variable x_i : Füge v_{i1} zu R wenn $\sigma(x_i) = 1$, sonst füge v_{i0} .
- Für jede Klausel K_j : Füge k_{j1} oder k_{j2} hinzu, um eine 4 an der Stelle K_j zu gewinnen.

PARTITION

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \dots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

PARTITION

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \dots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

PARTITION

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \dots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK \leq_p PARTITION

PARTITION

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \dots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK \leq_p PARTITION

Beispiel: $\vec{a} = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

PARTITION

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \dots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK \leq_p PARTITION

Beispiel: $\vec{a} = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

Lösung: Die Zahlen 7, 9, 7, 15, dh $R = \{2, 4, 5, 7\}$

PARTITION

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \dots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK \leq_p PARTITION

Beispiel: $\vec{a} = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

Lösung: Die Zahlen 7, 9, 7, 15, dh $R = \{2, 4, 5, 7\}$

RUCKSACK \rightarrow PARTITION:

PARTITION

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \dots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK \leq_p PARTITION

Beispiel: $\vec{a} = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

Lösung: Die Zahlen 7, 9, 7, 15, dh $R = \{2, 4, 5, 7\}$

RUCKSACK \rightarrow PARTITION:

$$M := \sum_{i=1}^n a_i = 57,$$

PARTITION

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \dots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK \leq_p PARTITION

Beispiel: $\vec{a} = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

Lösung: Die Zahlen 7, 9, 7, 15, dh $R = \{2, 4, 5, 7\}$

RUCKSACK \rightarrow PARTITION:

$$M := \sum_{i=1}^n a_i = 57, \quad M - b + 1 = 20, \quad b + 1 = 39$$

PARTITION

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \dots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK \leq_p PARTITION

Beispiel: $\vec{a} = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

Lösung: Die Zahlen 7, 9, 7, 15, dh $R = \{2, 4, 5, 7\}$

RUCKSACK \rightarrow PARTITION:

$$M := \sum_{i=1}^n a_i = 57, \quad M - b + 1 = 20, \quad b + 1 = 39$$

Das resultierende PARTITION-Problem: 12, 7, 4, 9, 7, 3, 15, 20, 39

PARTITION

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \dots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK \leq_p PARTITION

Beispiel: $\vec{a} = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

Lösung: Die Zahlen 7, 9, 7, 15, dh $R = \{2, 4, 5, 7\}$

RUCKSACK \rightarrow PARTITION:

$$M := \sum_{i=1}^n a_i = 57, \quad M - b + 1 = 20, \quad b + 1 = 39$$

Das resultierende PARTITION-Problem: 12, 7, 4, 9, 7, 3, 15, 20, 39

Lösung: $\{7, 9, 7, 15, 20\}$ und $\{12, 4, 3, 39\}$,

PARTITION

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \dots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK \leq_p PARTITION

Beispiel: $\vec{a} = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

Lösung: Die Zahlen 7, 9, 7, 15, dh $R = \{2, 4, 5, 7\}$

RUCKSACK \rightarrow PARTITION:

$$M := \sum_{i=1}^n a_i = 57, \quad M - b + 1 = 20, \quad b + 1 = 39$$

Das resultierende PARTITION-Problem: 12, 7, 4, 9, 7, 3, 15, 20, 39

Lösung: $\{7, 9, 7, 15, 20\}$ und $\{12, 4, 3, 39\}$, dh $I = \{2, 4, 5, 7, 8\}$.

PARTITION

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \dots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK \leq_p PARTITION

Beispiel: $\vec{a} = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

Lösung: Die Zahlen 7, 9, 7, 15, dh $R = \{2, 4, 5, 7\}$

RUCKSACK \rightarrow PARTITION:

$$M := \sum_{i=1}^n a_i = 57, \quad M - b + 1 = 20, \quad b + 1 = 39$$

Das resultierende PARTITION-Problem: 12, 7, 4, 9, 7, 3, 15, 20, 39

Lösung: $\{7, 9, 7, 15, 20\}$ und $\{12, 4, 3, 39\}$, dh $I = \{2, 4, 5, 7, 8\}$.

Reduktion im Allgemeinen:

$$a_1, a_2, \dots, a_k, b \mapsto a_1, a_2, \dots, a_k, M - b + 1, b + 1 \quad \square$$

BIN PACKING

Gegeben: Eine „Behältergröße“ $b \in \mathbb{N}$, die Anzahl der Behälter $k \in \mathbb{N}$ und „Objekte“ a_1, a_2, \dots, a_n .

BIN PACKING

Gegeben: Eine „Behältergröße“ $b \in \mathbb{N}$, die Anzahl der Behälter $k \in \mathbb{N}$ und „Objekte“ a_1, a_2, \dots, a_n .

Problem: Können die Objekte so auf die k Behälter verteilt werden, dass kein Behälter überläuft?

BIN PACKING

Gegeben: Eine „Behältergröße“ $b \in \mathbb{N}$, die Anzahl der Behälter $k \in \mathbb{N}$ und „Objekte“ a_1, a_2, \dots, a_n .

Problem: Können die Objekte so auf die k Behälter verteilt werden, dass kein Behälter überläuft?

Satz 6.32

BIN PACKING ist NP-vollständig.

BIN PACKING

Gegeben: Eine „Behältergröße“ $b \in \mathbb{N}$, die Anzahl der Behälter $k \in \mathbb{N}$ und „Objekte“ a_1, a_2, \dots, a_n .

Problem: Können die Objekte so auf die k Behälter verteilt werden, dass kein Behälter überläuft?

Satz 6.32

BIN PACKING ist NP-vollständig.

Beweis: PARTITION \leq_p BIN PACKING

BIN PACKING

Gegeben: Eine „Behältergröße“ $b \in \mathbb{N}$, die Anzahl der Behälter $k \in \mathbb{N}$ und „Objekte“ a_1, a_2, \dots, a_n .

Problem: Können die Objekte so auf die k Behälter verteilt werden, dass kein Behälter überläuft?

Satz 6.32

BIN PACKING ist NP-vollständig.

Beweis: PARTITION \leq_p BIN PACKING

$$(a_1, \dots, a_n) \mapsto (b, k, 2a_1, \dots, 2a_n)$$

wobei $b := \sum_{i=1}^n a_i$ und $k := 2$. □

HAMILTON

Gegeben: Ungerichteter Graph G

Problem: Enthält G einen Hamilton-Kreis, dh einen geschlossenen Pfad, der jeden Knoten genau einmal enthält?

HAMILTON

Gegeben: Ungerichteter Graph G

Problem: Enthält G einen Hamilton-Kreis, dh einen geschlossenen Pfad, der jeden Knoten genau einmal enthält?

Satz 6.33

HAMILTON ist NP-vollständig.

TRAVELLING SALESMAN (TSP)

Gegeben: Eine $n \times n$ Matrix $M_{ij} \in \mathbb{N}$ von „Entfernungen“
und eine Zahl $k \in \mathbb{N}$

TRAVELLING SALESMAN (TSP)

Gegeben: Eine $n \times n$ Matrix $M_{ij} \in \mathbb{N}$ von „Entfernungen“
und eine Zahl $k \in \mathbb{N}$

Problem: Gibt es eine „Rundreise“ (Hamilton-Kreis) der Länge
 $\leq k$?

TRAVELLING SALESMAN (TSP)

Gegeben: Eine $n \times n$ Matrix $M_{ij} \in \mathbb{N}$ von „Entfernungen“
und eine Zahl $k \in \mathbb{N}$

Problem: Gibt es eine „Rundreise“ (Hamilton-Kreis) der Länge
 $\leq k$?

Satz 6.34

TSP ist NP-vollständig.

TRAVELLING SALESMAN (TSP)

Gegeben: Eine $n \times n$ Matrix $M_{ij} \in \mathbb{N}$ von „Entfernungen“
und eine Zahl $k \in \mathbb{N}$

Problem: Gibt es eine „Rundreise“ (Hamilton-Kreis) der Länge
 $\leq k$?

Satz 6.34

TSP ist NP-vollständig.

Beweis: HAMILTON \leq_p TSP

TRAVELLING SALESMAN (TSP)

Gegeben: Eine $n \times n$ Matrix $M_{ij} \in \mathbb{N}$ von „Entfernungen“
und eine Zahl $k \in \mathbb{N}$

Problem: Gibt es eine „Rundreise“ (Hamilton-Kreis) der Länge
 $\leq k$?

Satz 6.34

TSP ist NP-vollständig.

Beweis: HAMILTON \leq_p TSP

$$(\{1, \dots, n\}, E) \mapsto (M, n)$$

wobei

$$M_{ij} := \begin{cases} 1 & \text{falls } \{i, j\} \in E \\ 2 & \text{sonst} \end{cases}$$



FÄRBBARKEIT (COL)

Gegeben: Ein ungerichteter Graph (V, E) und eine Zahl k .

FÄRBBARKEIT (COL)

Gegeben: Ein ungerichteter Graph (V, E) und eine Zahl k .

Problem: Gibt es eine Färbung der Knoten V mit k Farben, so dass keine zwei benachbarten Knoten die gleiche Farbe haben?

FÄRBBARKEIT (COL)

Gegeben: Ein ungerichteter Graph (V, E) und eine Zahl k .

Problem: Gibt es eine Färbung der Knoten V mit k Farben, so dass keine zwei benachbarten Knoten die gleiche Farbe haben?

Satz 6.35

FÄRBBARKEIT ist NP-vollständig für $k \geq 3$.

FÄRBBARKEIT (COL)

Gegeben: Ein ungerichteter Graph (V, E) und eine Zahl k .

Problem: Gibt es eine Färbung der Knoten V mit k Farben, so dass keine zwei benachbarten Knoten die gleiche Farbe haben?

Satz 6.35

FÄRBBARKEIT ist NP-vollständig für $k \geq 3$.

Beweis:

3KNF-SAT \leq_p 3FÄRBBARKEIT



FÄRBBARKEIT (COL)

Gegeben: Ein ungerichteter Graph (V, E) und eine Zahl k .

Problem: Gibt es eine Färbung der Knoten V mit k Farben, so dass keine zwei benachbarten Knoten die gleiche Farbe haben?

Satz 6.35

FÄRBBARKEIT ist NP-vollständig für $k \geq 3$.

Beweis:

3KNF-SAT \leq_p 3FÄRBBARKEIT



Satz 6.36

2FÄRBBARKEIT $\in P$

Die NP-Bibel, der NP-Klassiker:



[Michael Garey and David Johnson.](#)

Computers and Intractability: A Guide to the Theory of NP-Completeness. 1979.

Die NP-Bibel, der NP-Klassiker:



[Michael Garey and David Johnson.](#)

Computers and Intractability: A Guide to the Theory of NP-Completeness. 1979.

Despite the 23 years that have passed since its publication, I consider Garey and Johnson the single most important book on my office bookshelf.

Lance Fortnow, 2002.

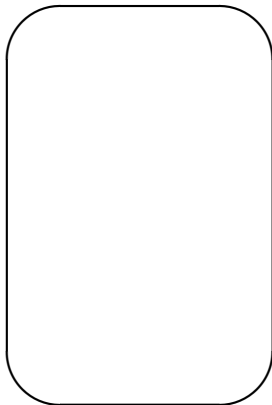
Man weiß nicht ob $P = NP$.

Man weiß nicht ob $P = NP$. Aber man weiß (Ladner 1975)

$$P \neq NP \implies$$

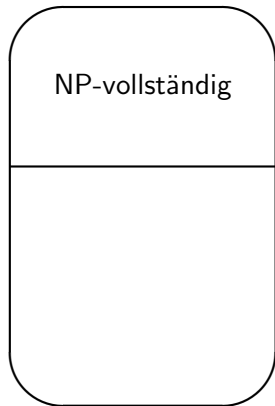
Man weiß nicht ob $P = NP$. Aber man weiß (Ladner 1975)

$P \neq NP \implies NP =$



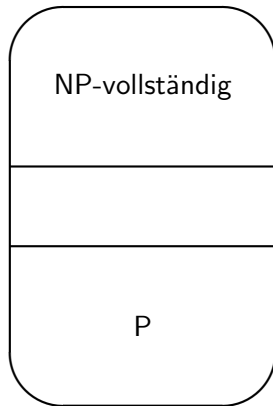
Man weiß nicht ob $P = NP$. Aber man weiß (Ladner 1975)

$P \neq NP \implies NP =$



Man weiß nicht ob $P = NP$. Aber man weiß (Ladner 1975)

$P \neq NP \implies NP =$





Kurt Gödel.

Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatshefte für Mathematik, 1931.



Kurt Gödel.

Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatshefte für Mathematik, 1931.

Kurt Gödel
1906 (Brünn) –
1978 (Princeton)



6.5 Die Unvollständigkeit der Arithmetik

6.5 Die Unvollständigkeit der Arithmetik

Syntax der Arithmetik:

Variablen: V $x y z \dots$

6.5 Die Unvollständigkeit der Arithmetik

Syntax der Arithmetik:

Variablen: V $x y z \dots$

Zahlen: N $0 1 2 \dots$

6.5 Die Unvollständigkeit der Arithmetik

Syntax der Arithmetik:

Variablen: V $x y z \dots$

Zahlen: N $0 1 2 \dots$

Terme: T $V N (T + T) (T * T)$

6.5 Die Unvollständigkeit der Arithmetik

Syntax der Arithmetik:

Variablen: V $x y z \dots$

Zahlen: N $0 1 2 \dots$

Terme: T $V N (T + T) (T * T)$

Formeln: F $(T = T) \neg F (F \wedge F) (F \vee F)$
 $\exists V. F$

6.5 Die Unvollständigkeit der Arithmetik

Syntax der Arithmetik:

Variablen: V $x y z \dots$

Zahlen: N $0 1 2 \dots$

Terme: T $V N (T + T) (T * T)$

Formeln: F $(T = T) \neg F (F \wedge F) (F \vee F)$
 $\exists V. F$

Wir betrachten $\forall x. F$ als Abk. für $\neg \exists x. \neg F$.

6.5 Die Unvollständigkeit der Arithmetik

Syntax der Arithmetik:

Variablen:	V	$x y z \dots$
Zahlen:	N	$0 1 2 \dots$
Terme:	T	$V N (T + T) (T * T)$
Formeln:	F	$(T = T) \neg F (F \wedge F) (F \vee F)$ $\exists V. F$

Wir betrachten $\forall x. F$ als Abk. für $\neg \exists x. \neg F$.

Definition 6.37

Ein Vorkommen einer Variablen x in einer Formel F ist **gebunden** gdw das Vorkommen in einer Teilformel der Form $\exists x. F'$ oder $\forall x. F'$ in der Teilformel F' liegt.

6.5 Die Unvollständigkeit der Arithmetik

Syntax der Arithmetik:

Variablen:	V	$x y z \dots$
Zahlen:	N	$0 1 2 \dots$
Terme:	T	$V N (T + T) (T * T)$
Formeln:	F	$(T = T) \neg F (F \wedge F) (F \vee F)$ $\exists V. F$

Wir betrachten $\forall x. F$ als Abk. für $\neg \exists x. \neg F$.

Definition 6.37

Ein Vorkommen einer Variablen x in einer Formel F ist **gebunden** gdw das Vorkommen in einer Teilformel der Form $\exists x. F'$ oder $\forall x. F'$ in der Teilformel F' liegt.

Ein Vorkommen ist **frei** gdw es nicht gebunden ist.

Notation: $F(x_1, \dots, x_k)$ bezeichnet eine Formel F , in der höchstens x_1, \dots, x_k frei vorkommen.

Notation: $F(x_1, \dots, x_k)$ bezeichnet eine Formel F , in der höchstens x_1, \dots, x_k frei vorkommen.

Sind $n_1, \dots, n_k \in \mathbb{N}$ so ist $F(n_1, \dots, n_k)$ das Ergebnis der Substitution von n_1, \dots, n_k für die freien Vorkommen von x_1, \dots, x_k .

Notation: $F(x_1, \dots, x_k)$ bezeichnet eine Formel F , in der höchstens x_1, \dots, x_k frei vorkommen.

Sind $n_1, \dots, n_k \in \mathbb{N}$ so ist $F(n_1, \dots, n_k)$ das Ergebnis der Substitution von n_1, \dots, n_k für die freien Vorkommen von x_1, \dots, x_k .

Beispiel 6.38

$$F(x, y) = (x = y \wedge \exists x. x = y)$$

Notation: $F(x_1, \dots, x_k)$ bezeichnet eine Formel F , in der höchstens x_1, \dots, x_k frei vorkommen.

Sind $n_1, \dots, n_k \in \mathbb{N}$ so ist $F(n_1, \dots, n_k)$ das Ergebnis der Substitution von n_1, \dots, n_k für die freien Vorkommen von x_1, \dots, x_k .

Beispiel 6.38

$$F(x, y) = (x = y \wedge \exists x. x = y)$$

$$F(5, 7) = (5 = 7 \wedge \exists x. x = 7)$$

Notation: $F(x_1, \dots, x_k)$ bezeichnet eine Formel F , in der höchstens x_1, \dots, x_k frei vorkommen.

Sind $n_1, \dots, n_k \in \mathbb{N}$ so ist $F(n_1, \dots, n_k)$ das Ergebnis der Substitution von n_1, \dots, n_k für die freien Vorkommen von x_1, \dots, x_k .

Beispiel 6.38

$$F(x, y) = (x = y \wedge \exists x. x = y)$$

$$F(5, 7) = (5 = 7 \wedge \exists x. x = 7)$$

Ein **Satz** ist eine Formel ohne freie Variablen.

Notation: $F(x_1, \dots, x_k)$ bezeichnet eine Formel F , in der höchstens x_1, \dots, x_k frei vorkommen.

Sind $n_1, \dots, n_k \in \mathbb{N}$ so ist $F(n_1, \dots, n_k)$ das Ergebnis der Substitution von n_1, \dots, n_k für die freien Vorkommen von x_1, \dots, x_k .

Beispiel 6.38

$$F(x, y) = (x = y \wedge \exists x. x = y)$$

$$F(5, 7) = (5 = 7 \wedge \exists x. x = 7)$$

Ein **Satz** ist eine Formel ohne freie Variablen.

Beispiel 6.39

$$\exists x. \exists y. x = y \qquad \exists y. \exists x. x = y$$

Notation: $F(x_1, \dots, x_k)$ bezeichnet eine Formel F , in der höchstens x_1, \dots, x_k frei vorkommen.

Sind $n_1, \dots, n_k \in \mathbb{N}$ so ist $F(n_1, \dots, n_k)$ das Ergebnis der Substitution von n_1, \dots, n_k für die freien Vorkommen von x_1, \dots, x_k .

Beispiel 6.38

$$F(x, y) = (x = y \wedge \exists x. x = y)$$

$$F(5, 7) = (5 = 7 \wedge \exists x. x = 7)$$

Ein **Satz** ist eine Formel ohne freie Variablen.

Beispiel 6.39

$$\exists x. \exists y. x = y \quad \exists y. \exists x. x = y$$

Mit S bezeichnen wir die Menge der arithmetischen Sätze.

Die Menge der **wahren** Sätze der Arithmetik nennen wir W :

Definition 6.40

Die Menge der wahren Sätze der Arithmetik nennen wir W :

Definition 6.40

$(t_1 = t_2) \in W$ gdw t_1 und t_2 den selben Wert haben.

Die Menge der **wahren** Sätze der Arithmetik nennen wir W :

Definition 6.40

$(t_1 = t_2) \in W$ gdw t_1 und t_2 den selben Wert haben.

$\neg F \in W$ gdw $F \in W$

Die Menge der **wahren** Sätze der Arithmetik nennen wir W :

Definition 6.40

$(t_1 = t_2) W$ gdw t_1 und t_2 den selben Wert haben.

$\neg F W$ gdw $F W$

$(F \wedge G) W$ gdw $F W$ und $G W$

Die Menge der **wahren** Sätze der Arithmetik nennen wir W :

Definition 6.40

$(t_1 = t_2) W$ gdw t_1 und t_2 den selben Wert haben.

$\neg F W$ gdw $F W$

$(F \wedge G) W$ gdw $F W$ und $G W$

$(F \vee G) W$ gdw $F W$ oder $G W$

Die Menge der **wahren** Sätze der Arithmetik nennen wir W :

Definition 6.40

$(t_1 = t_2) W$ gdw t_1 und t_2 den selben Wert haben.

$\neg F W$ gdw $F W$

$(F \wedge G) W$ gdw $F W$ und $G W$

$(F \vee G) W$ gdw $F W$ oder $G W$

$\exists x. F(x)$ gdw es $n \in \mathbb{N}$ gibt, so dass $F(n) W$

Die Menge der **wahren** Sätze der Arithmetik nennen wir W :

Definition 6.40

$(t_1 = t_2) W$ gdw t_1 und t_2 den selben Wert haben.

$\neg F W$ gdw $F W$

$(F \wedge G) W$ gdw $F W$ und $G W$

$(F \vee G) W$ gdw $F W$ oder $G W$

$\exists x. F(x)$ gdw es $n \in \mathbb{N}$ gibt, so dass $F(n) W$

Fakt 6.41

Für jeden Satz F gilt entweder $F W$ oder $\neg F W$

Die Menge der **wahren** Sätze der Arithmetik nennen wir W :

Definition 6.40

$(t_1 = t_2) W$ gdw t_1 und t_2 den selben Wert haben.

$\neg F W$ gdw $F W$

$(F \wedge G) W$ gdw $F W$ und $G W$

$(F \vee G) W$ gdw $F W$ oder $G W$

$\exists x. F(x)$ gdw es $n \in \mathbb{N}$ gibt, so dass $F(n) W$

Fakt 6.41

Für jeden Satz F gilt entweder $F W$ oder $\neg F W$

NB Ob eine Formel mit freien Variablen wahr ist, kann vom Wert der freien Variablen abhängen:

Die Menge der **wahren** Sätze der Arithmetik nennen wir W :

Definition 6.40

$(t_1 = t_2) W$ gdw t_1 und t_2 den selben Wert haben.

$\neg F W$ gdw $F W$

$(F \wedge G) W$ gdw $F W$ und $G W$

$(F \vee G) W$ gdw $F W$ oder $G W$

$\exists x. F(x)$ gdw es $n \in \mathbb{N}$ gibt, so dass $F(n) W$

Fakt 6.41

Für jeden Satz F gilt entweder $F W$ oder $\neg F W$

NB Ob eine Formel mit freien Variablen wahr ist, kann vom Wert der freien Variablen abhängen:

$$\exists x. x + x = y$$

Die Menge der **wahren** Sätze der Arithmetik nennen wir W :

Definition 6.40

$(t_1 = t_2) W$ gdw t_1 und t_2 den selben Wert haben.

$\neg F W$ gdw $F W$

$(F \wedge G) W$ gdw $F W$ und $G W$

$(F \vee G) W$ gdw $F W$ oder $G W$

$\exists x. F(x)$ gdw es $n \in \mathbb{N}$ gibt, so dass $F(n) W$

Fakt 6.41

Für jeden Satz F gilt entweder $F W$ oder $\neg F W$

NB Ob eine Formel mit freien Variablen wahr ist, kann vom Wert der freien Variablen abhängen:

$$\exists x. x + x = y$$

Daher haben wir Wahrheit nur für Sätze definiert.

Definition 6.42

Eine partielle Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist **arithmetisch repräsentierbar** gdw es eine Formel $F(x_1, \dots, x_k, y)$ gibt, so dass für alle $n_1, \dots, n_k, m \in \mathbb{N}$ gilt:

$$f(n_1, \dots, n_k) = m \quad \text{gdw} \quad F(n_1, \dots, n_k, m) \text{ W}$$

Definition 6.42

Eine partielle Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist **arithmetisch repräsentierbar** gdw es eine Formel $F(x_1, \dots, x_k, y)$ gibt, so dass für alle $n_1, \dots, n_k, m \in \mathbb{N}$ gilt:

$$f(n_1, \dots, n_k) = m \quad \text{gdw} \quad F(n_1, \dots, n_k, m) \text{ W}$$

Satz 6.43

Jede WHILE-berechenbare Funktion ist arithmetisch repräsentierbar.

Satz 6.44

W ist nicht entscheidbar.

Korollar 6.45

W ist nicht semi-entscheidbar.

Wir kodieren Beweise als Zahlen.

Definition 6.46

Ein **Beweissystem** für die Arithmetik ist ein entscheidbares Prädikat

$$Bew : \mathbb{N} \times S \rightarrow \{0, 1\}$$

wobei wir $Bew(b, F)$ lesen als „ b ist Beweis für Formel F “.

Ein Beweissystem Bew ist **korrekt** gdw

$$Bew(b, F) \implies F \text{ W.}$$

Ein Beweissystem Bew ist **vollständig** gdw

$$F \text{ W} \implies \text{es gibt } b \text{ mit } Bew(b, F).$$

Satz 6.47 (Gödel)

Es gibt kein korrektes und vollständiges Beweissystem für die Arithmetik.

Beweis:

Denn mit jedem korrekten und vollständigen Beweissystem kann man $\chi'_W(F)$ programmieren:

$b := 0$

while $Bew(b, F) = 0$ **do** $b := b + 1$

output(1)



6.6 Die Entscheidbarkeit der Presburger Arithmetik

6.6 Die Entscheidbarkeit der Presburger Arithmetik

Syntax der Presburger Arithmetik:

Variablen: $V \rightarrow x \mid y \mid z \mid \dots$

Zahlen: $N \rightarrow 0 \mid 1 \mid 2 \mid \dots$

Terme: $T \rightarrow V \mid N \mid T + T$

Formeln: $F \rightarrow (T = T) \mid \neg F \mid (F \wedge F) \mid (F \vee F)$
 $\mid \exists V. F$

6.6 Die Entscheidbarkeit der Presburger Arithmetik

Syntax der Presburger Arithmetik:

Variablen: $V \rightarrow x \mid y \mid z \mid \dots$

Zahlen: $N \rightarrow 0 \mid 1 \mid 2 \mid \dots$

Terme: $T \rightarrow V \mid N \mid T + T$

Formeln: $F \rightarrow (T = T) \mid \neg F \mid (F \wedge F) \mid (F \vee F)$
 $\mid \exists V. F$

Wir betrachten $\forall x. F$ als Abk. für $\neg \exists x. \neg F$.

6.6 Die Entscheidbarkeit der Presburger Arithmetik

Syntax der Presburger Arithmetik:

Variablen: $V \rightarrow x \mid y \mid z \mid \dots$

Zahlen: $N \rightarrow 0 \mid 1 \mid 2 \mid \dots$

Terme: $T \rightarrow V \mid N \mid T + T$

Formeln: $F \rightarrow (T = T) \mid \neg F \mid (F \wedge F) \mid (F \vee F)$
 $\mid \exists V. F$

Wir betrachten $\forall x. F$ als Abk. für $\neg \exists x. \neg F$.

Satz 6.48

Für Sätze der Presburger Arithmetik ist entscheidbar, ob sie wahr sind.



Mojzesz Presburger.

Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. 1929.



Mojzesz Presburger.

Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. 1929.

Mojzesz Presburger 1904 – 1943



Mojzesz Presburger.

Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. 1929.

Mojzesz Presburger 1904 – 1943

YAD VASHEM

The Holocaust Martyrs' and Heroes' Remembrance Authority
Hall of Names - P.O.B. 3477, Jerusalem 91034 www.yadvashem.org



יד ושם

רשות הזיכרון לשואה ולגבורה
היכל השמות - ת.ד. 3477, ירושלים 91034

Page of Testimony דף עד

דף עד לרישום והנצחה של הנספים בשואה; נא למלא דף עבור כל נספה בנפרד, בכתב ברור ובאותיות דפוס.
Page of Testimony for commemoration of the Jews who perished during the Shoah; please submit a separate form for each victim, in block capitals



רוק יצירן השואה והגבורה - משיג 1953 קובץ מסעי מס' 2 כי יתקדו של יד ושם הוא לסמן אל השולח את זכרם של כל אלה בני העם היהודי שנבלע וסורו את נפשם, לתמו וחדד כביש האנטי ובמיוחד ולהביא עם זכרם לתב, לקהילות, לטענות ולמסמכת שנרצחו בגלל השתייכותם לעם היהודי.
The Martyrs' and Heroes' Remembrance Law 5713-1953 determines in section 2 that: "The task of Yad Vashem is to gather into the homeland material regarding all those members of the Jewish people who laid down their lives, who fought and rebelled against the Nazi enemy and his collaborators, and to perpetuate their names and those of the communities, organizations and institutions which were destroyed because they were Jewish".

Maiden name: שם משפחה לפני הנישואין; Victim's family name: שם משפחה של הנספה;

PRESBURGER
Previous/other family name: שם משפחה קודם/אחר; First name (also nickname): שם פרטי (גם שם חיבה/כינוי);

MOJZESZ O MIETEK ?

Approx. age at death: גיל משוער בעת המוות: 43	Date of birth: תאריך לידה: 1904	Gender: מין: M/F	Title: תואר: philospher
Nationality: תזיונות: POLISH	Country: ארץ: POLAND	Region: מחוז: ?	Place of birth: מקום לידה: ?

Victim's father: שם פרטי: אב הנספה:	Family name: שם משפחה:	First name: שם פרטי: אב הנספה:
Victim's mother: שם פרטי: אב הנספה:	Maiden name: שם לפני הנישואין:	First name: שם פרטי: אב הנספה:

Presburger Arithmetik = normale Arithmetik
ohne **Multiplikation**

Arithmetik : hochgradig unentscheidbar :-(
sogar **sogar unvollständig** :-((

⇒ Hilberts 10tes Problem

⇒ Gödels Theorem

Vereinfachte Presburger Formeln:

$$\begin{aligned} \phi \quad ::= & \quad x + y = z \quad | \quad x = n \quad | \\ & \quad \phi_1 \wedge \phi_2 \quad | \quad \neg \phi \quad | \\ & \quad \exists x. \phi \end{aligned}$$

Vereinfachte Presburger Formeln:

$$\begin{aligned} \phi \quad ::= & \quad x + y = z \quad | \quad x = n \quad | \\ & \quad \phi_1 \wedge \phi_2 \quad | \quad \neg \phi \quad | \\ & \quad \exists x. \phi \end{aligned}$$

Bemerkungen

- Durch sukzessive Addition kann man Multiplikation mit Konstanten simulieren.
(Wie?)

Vereinfachte Presburger Formeln:

$$\phi ::= x + y = z \mid x = n \mid \\ \phi_1 \wedge \phi_2 \mid \neg \phi \mid \\ \exists x. \phi$$

Bemerkungen

- Durch sukzessive Addition kann man Multiplikation mit Konstanten simulieren.
(Wie?)
- Das Rucksackproblem lässt sich in PA ausdrücken.
(Wie?)

Allgemeineres Ziel:

PSAT

Finde Werte in \mathbb{N} für die freien Variablen , so dass ϕ gilt ...

Allgemeineres Ziel:

PSAT

Finde Werte in \mathbb{N} für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Allgemeineres Ziel:

PSAT

Finde Werte in \mathbb{N} für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Allgemeineres Ziel:

PSAT

Finde Werte in \mathbb{N} für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Allgemeineres Ziel:

PSAT

Finde Werte in \mathbb{N} für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Allgemeineres Ziel:

PSAT

Finde Werte in \mathbb{N} für die freien Variablen, so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Allgemeineres Ziel:

PSAT

Finde Werte in \mathbb{N} für die freien Variablen, so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Allgemeineres Ziel:

PSAT

Finde Werte in \mathbb{N} für die freien Variablen, so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Allgemeineres Ziel:

PSAT

Finde Werte in \mathbb{N} für die freien Variablen, so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Allgemeineres Ziel:

PSAT

Finde Werte in \mathbb{N} für die freien Variablen, so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Allgemeineres Ziel:

PSAT

Finde Werte in \mathbb{N} für die freien Variablen, so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Allgemeineres Ziel:

PSAT

Finde Werte in \mathbb{N} für die freien Variablen, so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Beobachtung:

Die Menge der erfüllenden Variablenbelegungen ist regulär !!

Beobachtung:

Die Menge der erfüllenden Variablenbelegungen ist regulär !!

$$\begin{array}{lll} \phi_1 \wedge \phi_2 & \implies & \mathcal{L}(\phi_1) \cap \mathcal{L}(\phi_2) \quad (\text{Durchschnitt}) \\ \neg \phi & \implies & \overline{\mathcal{L}(\phi)} \quad (\text{Komplement}) \\ \exists x : \phi & \implies & \pi_x(\mathcal{L}(\phi)) \quad (\text{Projektion}) \end{array}$$

Beobachtung:

Die Menge der erfüllenden Variablenbelegungen ist **regulär** !!

$$\begin{array}{lll} \phi_1 \wedge \phi_2 & \implies & \mathcal{L}(\phi_1) \cap \mathcal{L}(\phi_2) \quad (\text{Durchschnitt}) \\ \neg \phi & \implies & \overline{\mathcal{L}(\phi)} \quad (\text{Komplement}) \\ \exists x : \phi & \implies & \pi_x(\mathcal{L}(\phi)) \quad (\text{Projektion}) \end{array}$$

Achtung:

- Ein akzeptiertes Tupel kann immer durch führende Nullen verlängert werden !
- bleibt unter Vereinigung, Durchschnitt und Komplement erhalten !!

Beobachtung:

Die Menge der erfüllenden Variablenbelegungen ist **regulär** !!

$$\begin{array}{lll} \phi_1 \wedge \phi_2 & \implies & \mathcal{L}(\phi_1) \cap \mathcal{L}(\phi_2) \quad (\text{Durchschnitt}) \\ \neg \phi & \implies & \overline{\mathcal{L}(\phi)} \quad (\text{Komplement}) \\ \exists x : \phi & \implies & \pi_x(\mathcal{L}(\phi)) \quad (\text{Projektion}) \end{array}$$

Achtung:

- Ein akzeptiertes Tupel kann immer durch führende Nullen verlängert werden !
- bleibt unter Vereinigung, Durchschnitt und Komplement erhalten !!
- Wie sieht das mit der Projektion aus ?

Weg-Projizierung der x -Komponente:

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Weg-Projizierung der x -Komponente:

213

t

42

z

89

y

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

Weg-Projizierung der x -Komponente:

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0

- Nun werden möglicherweise Tupel von Zahlen erst nach Anhängen von geeignet vielen führenden Nullen akzeptiert !

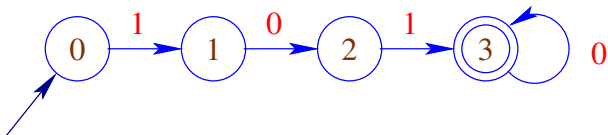
Weg-Projizierung der x -Komponente:

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0

- Nun werden möglicherweise Tupel von Zahlen erst nach Anhängen von geeignet vielen führenden Nullen akzeptiert !
- Der Automat muss so komplettiert werden, dass q bereits akzeptierend ist, wenn von q aus mit Nullen ein akzeptierender Zustand erreicht werden kann.

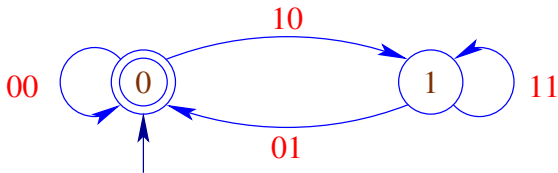
Automaten für Basis-Prädikate:

$$x = 5$$



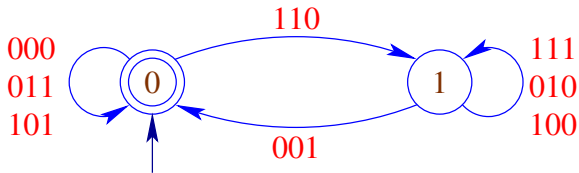
Automaten für Basis-Prädikate:

$$x+x = y$$



Automaten für Basis-Prädikate:

$$x+y = z$$



Ergebnisse:

Ferrante, Rackoff, 1973 : $\text{PSAT} \leq \text{DSPACE}(2^{2^{c \cdot n}})$

Ergebnisse:

Ferrante, Rackoff, 1973 : $\text{PSAT} \leq \text{DSPACE}(2^{2^{c \cdot n}})$

Fischer, Rabin, 1974 : $\text{PSAT} \geq \text{NTIME}(2^{2^{c \cdot n}})$

Informatik im Jahr 2100: Eine Vorhersage

Informatik im Jahr 2100: Eine Vorhersage

Java 7



Informatik im Jahr 2100: Eine Vorhersage

Java 7

~> **Taiwan (Red Dragon Edition)**

Informatik im Jahr 2100: Eine Vorhersage

Java 7

Python



Taiwan (Red Dragon Edition)



Informatik im Jahr 2100: Eine Vorhersage

Java 7

Python

~> Taiwan (Red Dragon Edition)

~> Gnu-Boa++# 18.3

Informatik im Jahr 2100: Eine Vorhersage

Java 7
Python
XML

~> Taiwan (Red Dragon Edition)
~> Gnu-Boa++# 18.3
~>

Informatik im Jahr 2100: Eine Vorhersage

Java 7
Python
XML

~> Taiwan (Red Dragon Edition)
~> Gnu-Boa++# 18.3
~> Medea

Informatik im Jahr 2100: Eine Vorhersage

Java 7	~>	Taiwan (Red Dragon Edition)
Python	~>	Gnu-Boa++# 18.3
XML	~>	Medea
Reguläre Sprachen	~>	

Informatik im Jahr 2100: Eine Vorhersage

Java 7	~>	Taiwan (Red Dragon Edition)
Python	~>	Gnu-Boa++# 18.3
XML	~>	Medea
Reguläre Sprachen	~>	Reguläre Sprachen

Informatik im Jahr 2100: Eine Vorhersage

Java 7	~>	Taiwan (Red Dragon Edition)
Python	~>	Gnu-Boa++# 18.3
XML	~>	Medea
Reguläre Sprachen	~>	Reguläre Sprachen
Berechenbare Funktionen	~>	

Informatik im Jahr 2100: Eine Vorhersage

Java 7	~>	Taiwan (Red Dragon Edition)
Python	~>	Gnu-Boa++# 18.3
XML	~>	Medea
Reguläre Sprachen	~>	Reguläre Sprachen
Berechenbare Funktionen	~>	Berechenbare Funktionen

Informatik im Jahr 2100: Eine Vorhersage

Java 7	↪	Taiwan (Red Dragon Edition)
Python	↪	Gnu-Boa++# 18.3
XML	↪	Medea
Reguläre Sprachen	↪	Reguläre Sprachen
Berechenbare Funktionen	↪	Berechenbare Funktionen
P	↪	

Informatik im Jahr 2100: Eine Vorhersage

Java 7	↪	Taiwan (Red Dragon Edition)
Python	↪	Gnu-Boa++# 18.3
XML	↪	Medea
Reguläre Sprachen	↪	Reguläre Sprachen
Berechenbare Funktionen	↪	Berechenbare Funktionen
P	↪	P