Script generated by TTT

Title: Nipkow: Theo (25.04.2019)

Date: Thu Apr 25 14:14:05 CEST 2019

Duration: 88:24 min

Pages: 138

Einführung in die Theoretische Informatik

Tobias Nipkow

Fakultät für Informatik
TU München

Sommersemester 2019

Organisatorisches

http://www21.in.tum.de/teaching/theo/SS19/

Fragen (von Ihnen), vor allem zu Übungen,
und Ankündigungen (von uns)
auf Piazza:
https://piazza.com/tum.de/summer2019/in0011

Termine

- Vorlesung:
  - Mo 14:15–15:45 im MW 0001
  - Do 14:15–15:45 im MW 0001
- Sprechstunde: Do 16:00–16:45 im MI HS1
- Klausur: ~~09.08.2019~~

Übungskonzept

- Erfahrung aus letzten Jahren:
  - Sehr unterschiedliche Wissenstände unter den Studierenden
  - Zu wenig Zeit, um Aufgaben ausreichend zu bearbeiten

Inhalt und Gliederung der Vorlesung

- Endliche Automaten und reguläre Ausdrücke

Inhalt und Gliederung der Vorlesung

- Endliche Automaten und reguläre Ausdrücke
  - Grundlagen der Textanalyse, der lexikalischen Analyse von Programmiersprachen, der Spezifikation und Analyse von Kommunikationsprotokollen, ...

Inhalt und Gliederung der Vorlesung

- Endliche Automaten und reguläre Ausdrücke
  - Grundlagen der Textanalyse, der lexikalischen Analyse von Programmiersprachen, der Spezifikation und Analyse von Kommunikationsprotokollen, ...
- Kontextfreie Grammatiken

Inhalt und Gliederung der Vorlesung

- Endliche Automaten und reguläre Ausdrücke
  - Grundlagen der Textanalyse, der lexikalischen Analyse von Programmiersprachen, der Spezifikation und Analyse von Kommunikationsprotokollen, ...
- Kontextfreie Grammatiken
  - Grundlagen der syntaktische Analyse von Programmiersprachen, Parsing, Compilerbau

Inhalt und Gliederung der Vorlesung

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

Inhalt und Gliederung der Vorlesung

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

Inhalt und Gliederung der Vorlesung

- Endliche Automaten und reguläre Ausdrücke
  - Grundlagen der Textanalyse, der lexikalischen Analyse von Programmiersprachen, der Spezifikation und Analyse von Kommunikationsprotokollen, ...
- Kontextfreie Grammatiken
  - Grundlagen der syntaktische 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.

Inhalt und Gliederung der Vorlesung

- Endliche Automaten und reguläre Ausdrücke
  - Grundlagen der Textanalyse, der lexikalischen Analyse von Programmiersprachen, der Spezifikation und Analyse von Kommunikationsprotokollen, ...
- Kontextfreie Grammatiken
  - Grundlagen der syntaktische 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.

Universalwerkzeuge der Informatik

Wichtige abstrakte Ziele

Abstraktion von *irrelevanten* Details:

Wichtige abstrakte Ziele

Abstraktion von *irrelevanten* Details:
zB nicht x86 sondern Turingmaschine.

Wichtige abstrakte Ziele

Abstraktion von *irrelevanten* Details:
zB nicht x86 sondern Turingmaschine.

Formalisierung durch mathematische Objekte (Mengen, Funktionen, Relationen)

Wichtige abstrakte Ziele

Abstraktion von *irrelevanten* Details:
zB nicht x86 sondern Turingmaschine.

Formalisierung durch mathematische Objekte (Mengen, Funktionen, Relationen)

Simulation eines Formalismus durch einen anderen:

Wichtige abstrakte Ziele

Abstraktion von *irrelevanten* Details:
zB nicht x86 sondern Turingmaschine.

Formalisierung durch mathematische Objekte (Mengen, Funktionen, Relationen)

Simulation eines Formalismus durch einen anderen:
zB deterministische durch nichtdeterministische Maschinen

Wichtige abstrakte Ziele

Abstraktion von *irrelevanten* Details:
zB nicht x86 sondern Turingmaschine.

Formalisierung durch mathematische Objekte (Mengen, Funktionen, Relationen)

Simulation eines Formalismus durch einen anderen:
zB deterministische durch nichtdeterministische Maschinen

Äquivalenz von Formalismen:

Wichtige abstrakte Ziele

Abstraktion von *irrelevanten* Details:
zB nicht x86 sondern Turingmaschine.

Formalisierung durch mathematische Objekte (Mengen, Funktionen, Relationen)

Simulation eines Formalismus durch einen anderen:
zB deterministische durch nichtdeterministische Maschinen

Äquivalenz von Formalismen:
zB endliche Automaten und reguläre Ausdrücke.

Wichtige abstrakte Ziele

Abstraktion von *irrelevanten* Details:
zB nicht x86 sondern Turingmaschine.

Formalisierung durch mathematische Objekte (Mengen, Funktionen, Relationen)

Simulation eines Formalismus durch einen anderen:
zB deterministische durch nichtdeterministische Maschinen

Äquivalenz von Formalismen:
zB endliche Automaten und reguläre Ausdrücke.

Reduktion von einem Problem auf ein anderes:

Wichtige abstrakte Ziele

Abstraktion von *irrelevanten* Details:
zB nicht x86 sondern Turingmaschine.

Formalisierung durch mathematische Objekte (Mengen, Funktionen, Relationen)

Simulation eines Formalismus durch einen anderen:
zB deterministische durch nichtdeterministische Maschinen

Äquivalenz von Formalismen:
zB endliche Automaten und reguläre Ausdrücke.

Reduktion von einem Problem auf ein anderes:
zB Formeln auf Automaten, ...

Wichtige abstrakte Ziele

Abstraktion von *irrelevanten* Details:
zB nicht x86 sondern Turingmaschine.

Formalisierung durch mathematische Objekte (Mengen, Funktionen, Relationen)

Simulation eines Formalismus durch einen anderen:
zB deterministische durch nichtdeterministische Maschinen

Äquivalenz von Formalismen:
zB endliche Automaten und reguläre Ausdrücke.

Reduktion von einem Problem auf ein anderes:
zB Formeln auf Automaten, ...

Endliche Beschreibungen unendlicher Mengen

Geschichte:

1936 Berechenbarkeitstheorie: Church & Turing

Geschichte:

1936 Berechenbarkeitstheorie: Church & Turing

1956 Automaten und reguläre Ausdrücke: Kleene

Geschichte:

1936 Berechenbarkeitstheorie: Church & Turing

1956 Automaten und reguläre Ausdrücke: Kleene

1956 Grammatiken: Chomsky

14

Geschichte:

1936 Berechenbarkeitstheorie: Church & Turing

1956 Automaten und reguläre Ausdrücke: Kleene

1956 Grammatiken: Chomsky

1971 Komplexitätstheorie: Cook

14

Vorkenntnisse und weiterführende Vorlesungen

- Vorkenntnisse:
  - Einführung in die Informatik 1
  - Diskrete Strukturen

Vorkenntnisse und weiterführende Vorlesungen

- Vorkenntnisse:
  - Einführung in die Informatik 1
  - Diskrete Strukturen
- Weiterführende Vorlesungen:
  - Automata und Formal Languages
  - Complexity Theory
  - Logic
  - Model Checking
  - Quantitative Verification
  - Compiler Construction
  - ...

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.

1. Historischer Hintergrund

- Erste höhere Programmiersprachen:
  Fortran (1954), Lisp (1959), COBOL (1959)

1. Historischer Hintergrund

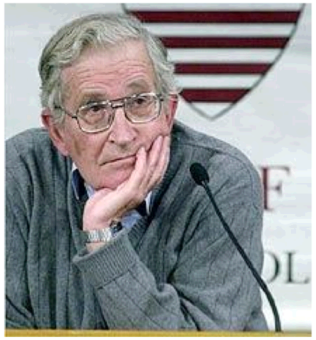
- Erste höhere Programmiersprachen:
  Fortran (1954), Lisp (1959), COBOL (1959)
- Die Geburt formaler Grammatiken:

Noam Chomsky.
Three Models for the Description of Language.
Transactions on Information Theory, 113-124, 1956.

1. Historischer Hintergrund

- Erste höhere Programmiersprachen:
  Fortran (1954), Lisp (1959), COBOL (1959)
- Die Geburt formaler Grammatiken:

  Noam Chomsky.
  Three Models for the Description of Language.
  Transactions on Information Theory, 113-124, 1956.
- Seitdem wird die Syntax von Programmiersprachen durch formale Grammatiken beschrieben.



Noam Chomsky (1928) ist Professor für Linguistik am MIT und ein bedeutender Sprachwissenschaftler.
Neben seiner linguistischen Arbeit gilt Chomsky als einer der bedeutendsten Intellektuellen Nordamerikas und ist als scharfer Kritiker der US-amerikanischen Außenpolitik bekannt.
[Quelle: Wikipedia]

- Beispiele von Regeln für den Bau von Sätzen:

$$\begin{aligned} \text{Satz} &\rightarrow \text{Nominalphrase Verbalphrase} \\ \text{Verbalphrase} &\rightarrow \text{Verb Nominalphrase} \\ \text{Nominalphrase} &\rightarrow \text{Artikel Nomen} \end{aligned}$$

- Beispiele von Regeln für den Bau von Sätzen:

$$\begin{aligned} \text{Satz} &\rightarrow \text{Nominalphrase Verbalphrase} \\ \text{Verbalphrase} &\rightarrow \text{Verb Nominalphrase} \\ \text{Nominalphrase} &\rightarrow \text{Artikel Nomen} \end{aligned}$$

- Beispiele von Regeln für den Bau von mathematischen Ausdrücken:

$$\begin{aligned} \text{Expression} &\rightarrow \text{Identifier} \\ \text{Expression} &\rightarrow \text{Expression} + \text{Expression} \\ \text{Expression} &\rightarrow \text{Expression} * \text{Expression} \end{aligned}$$

- Das Wortproblem: Kann eine gegebene Zeichenkette ("Wort") von den Regeln einer Grammatik erzeugt werden, d.h. ist es eine legale Zeichenkette bzgl. der Grammatik?

- Beispiele von Regeln für den Bau von Sätzen:

$$\begin{aligned} \text{Satz} &\rightarrow \text{Nominalphrase Verbalphrase} \\ \text{Verbalphrase} &\rightarrow \text{Verb Nominalphrase} \\ \text{Nominalphrase} &\rightarrow \text{Artikel Nomen} \end{aligned}$$

- Beispiele von Regeln für den Bau von mathematischen Ausdrücken:

$$\begin{aligned} \text{Expression} &\rightarrow \text{Identifier} \\ \text{Expression} &\rightarrow \text{Expression} + \text{Expression} \\ \text{Expression} &\rightarrow \text{Expression} * \text{Expression} \end{aligned}$$

- Das Wortproblem: Kann eine gegebene Zeichenkette ("Wort") von den Regeln einer Grammatik erzeugt werden, d.h. ist es eine legale Zeichenkette bzgl. der Grammatik?

- Das Wortproblem: Kann eine gegebene Zeichenkette ("Wort") von den Regeln einer Grammatik erzeugt werden, d.h. ist es eine legale Zeichenkette bzgl. der Grammatik?
- Recognizer: Programm, welches das Wortproblem für eine gegebene Grammatik löst.
  Recognizer → Lexer/Parser → Compiler
- Hauptfragen:

- Das Wortproblem: Kann eine gegebene Zeichenkette ("Wort") von den Regeln einer Grammatik erzeugt werden, d.h. ist es eine legale Zeichenkette bzgl. der Grammatik?
- Recognizer: Programm, welches das Wortproblem für eine gegebene Grammatik löst.
  Recognizer $\rightarrow$ Lexer/Parser $\rightarrow$ 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.

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 is 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 is 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.
- 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 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.
- Sind u und v Wörter, so ist uv ihre Konkatenation.
- Ist w ein Wort, so ist w^n definiert durch $w^0 = \epsilon$ und $w^{n+1} = ww^n$.

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.
- 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 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.
- 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 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.
- 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
- $L_1 = \{\epsilon, ab, abab, ababab, \ldots\} = \{(ab)^n \mid n \in \mathbb{N}\}$
  $(\Sigma_1 = \{a, b\})$

Beispiel 2.2 (Formale Sprachen)

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

Beispiel 2.2 (Formale Sprachen)

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

Definition 2.3 (Operationen auf Sprachen)

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

Definition 2.3 (Operationen auf Sprachen)

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

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

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

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

- Konkatenation: $AB = \{uv \mid u \in A \wedge v \in B\}$
  Bsp: $\{ab, b\}\{a, bb\} = \{aba, abbb, ba, bbb\}$
  NB: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \cdots w_n \mid w_1, \ldots, w_n \in A\}$

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: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \cdots w_n \mid w_1, \ldots, w_n \in A\} = \underbrace{A \cdots A}_{n}$

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: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \cdots w_n \mid w_1, \ldots, w_n \in A\} = \underbrace{A \cdots A}_{n}$

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

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

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: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \cdots w_n \mid w_1, \ldots, w_n \in A\} = \underbrace{A \cdots A}_{n}$
  Bsp: $\{ab, ba\}^2 = \{abab, abba, baab, baba\}$
  Rekursiv: $A^0 = \{\epsilon\}$ und $A^{n+1} = AA^n$
- $A^* = \{w_1 \cdots w_n \mid n \geq 0 \wedge w_1, \ldots, w_n \in A\} = \bigcup_{n \in \mathbb{N}} A^n$

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: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \cdots w_n \mid w_1, \ldots, w_n \in A\} = \underbrace{A \cdots A}_{n}$
  Bsp: $\{ab, ba\}^2 = \{abab, abba, baab, baba\}$
  Rekursiv: $A^0 = \{\epsilon\}$ und $A^{n+1} = AA^n$
- $A^* = \{w_1 \cdots w_n \mid n \geq 0 \wedge w_1, \ldots, w_n \in A\} = \bigcup_{n \in \mathbb{N}} A^n$
  Bsp: $\{01\}^* = \{\epsilon, 01, 0101, 010101, \ldots\}$

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: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \cdots w_n \mid w_1, \ldots, w_n \in A\} = \underbrace{A \cdots A}_{n}$
  Bsp: $\{ab, ba\}^2 = \{abab, abba, baab, baba\}$
  Rekursiv: $A^0 = \{\epsilon\}$ und $A^{n+1} = AA^n$
- $A^* = \{w_1 \cdots w_n \mid n \geq 0 \wedge w_1, \ldots, w_n \in A\} = \bigcup_{n \in \mathbb{N}} A^n$
  Bsp: $\{01\}^* = \{\epsilon, 01, 0101, 010101, \ldots\} \subsetneq \{0, 1\}^*$

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: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \cdots w_n \mid w_1, \ldots, w_n \in A\} = \underbrace{A \cdots A}_{n}$
  Bsp: $\{ab, ba\}^2 = \{abab, abba, baab, baba\}$
  Rekursiv: $A^0 = \{\epsilon\}$ und $A^{n+1} = AA^n$
- $A^* = \{w_1 \cdots w_n \mid n \geq 0 \wedge w_1, \ldots, w_n \in A\} = \bigcup_{n \in \mathbb{N}} A^n$
  Bsp: $\{01\}^* = \{\epsilon, 01, 0101, 010101, \ldots\} \neq \{0, 1\}^*$
- $A^+ = AA^* = \bigcup_{n \geq 1} A^n$

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: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \cdots w_n \mid w_1, \ldots, w_n \in A\} = \underbrace{A \cdots A}_{n}$
  Bsp: $\{ab, ba\}^2 = \{abab, abba, baab, baba\}$
  Rekursiv: $A^0 = \{\epsilon\}$ und $A^{n+1} = AA^n$
- $A^* = \{w_1 \cdots w_n \mid n \geq 0 \wedge w_1, \ldots, w_n \in A\} = \bigcup_{n \in \mathbb{N}} A^n$
  Bsp: $\{01\}^* = \{\epsilon, 01, 0101, 010101, \ldots\} \neq \{0, 1\}^*$
- $A^+ = AA^* = \bigcup_{n \geq 1} A^n$
  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: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \cdots w_n \mid w_1, \ldots, w_n \in A\} = \underbrace{A \cdots A}_{n}$
  Bsp: $\{ab, ba\}^2 = \{abab, abba, baab, baba\}$
  Rekursiv: $A^0 = \{\epsilon\}$ und $A^{n+1} = AA^n$
- $A^* = \{w_1 \cdots w_n \mid n \geq 0 \wedge w_1, \ldots, w_n \in A\} = \bigcup_{n \in \mathbb{N}} A^n$
  Bsp: $\{01\}^* = \{\epsilon, 01, 0101, 010101, \ldots\} \neq \{0, 1\}^*$
- $A^+ = AA^* = \bigcup_{n \geq 1} A^n$
  Bsp: $\Sigma^+ =$ Menge aller nicht-leeren Wörter über Σ

Achtung:

- Für alle A: $\epsilon \in A^*$

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: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \cdots w_n \mid w_1, \ldots, w_n \in A\} = \underbrace{A \cdots A}_{n}$
  Bsp: $\{ab, ba\}^2 = \{abab, abba, baab, baba\}$
  Rekursiv: $A^0 = \{\epsilon\}$ und $A^{n+1} = AA^n$
- $A^* = \{w_1 \cdots w_n \mid n \geq 0 \wedge w_1, \ldots, w_n \in A\} = \bigcup_{n \in \mathbb{N}} A^n$
  Bsp: $\{01\}^* = \{\epsilon, 01, 0101, 010101, \ldots\} \neq \{0, 1\}^*$
- $A^+ = AA^* = \bigcup_{n \geq 1} A^n$
  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: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \cdots w_n \mid w_1, \ldots, w_n \in A\} = \underbrace{A \cdots A}_{n}$
  Bsp: $\{ab, ba\}^2 = \{abab, abba, baab, baba\}$
  Rekursiv: $A^0 = \{\epsilon\}$ und $A^{n+1} = AA^n$
- $A^* = \{w_1 \cdots w_n \mid n \geq 0 \wedge w_1, \ldots, w_n \in A\} = \bigcup_{n \in \mathbb{N}} A^n$
  Bsp: $\{01\}^* = \{\epsilon, 01, 0101, 010101, \ldots\} \neq \{0, 1\}^*$
- $A^+ = AA^* = \bigcup_{n \geq 1} A^n$
  Bsp: $\Sigma^+ =$ Menge aller nicht-leeren Wörter über Σ

Achtung:

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

Einige Rechenregeln:

Lemma 2.4

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

Lemma 2.5

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

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

$= \{uv \mid u \in A \wedge v \in B \cup C\}$

$= (v \in B \vee v \in C)$

$= \{uv \mid u \in A \wedge v \in B \vee u \in A \wedge v \in C\}$

$= \{uv \mid \ldots\} \cup \{uv \mid \ldots\}$

$= AB \cup AC$

Einige Rechenregeln:

Lemma 2.4

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

Lemma 2.5

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

$\emptyset$ / $A \cap \{a, aa\}$

$A = \{\epsilon, a\}$ $B = \{\epsilon\}$ $C = \{a\}$

Einige Rechenregeln:

Lemma 2.4

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

Lemma 2.5

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

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

Lemma 2.6
$A^*A^* = A^*$

Definition 2.7
Eine Grammatik ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei

Definition 2.7
Eine Grammatik ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei

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

Definition 2.7
Eine Grammatik ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei

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

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

Definition 2.7
Eine Grammatik ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei

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

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

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

Definition 2.7
Eine Grammatik ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei

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

Σ ist eine endliche Menge von Terminalzeichen (oder Terminalen), 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.

Definition 2.7
Eine Grammatik ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei

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

Σ ist eine endliche Menge von Terminalzeichen (oder Terminalen), 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:

- $A, B, C, \ldots$ sind Nichtterminale,

Definition 2.7
Eine Grammatik ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei

- V ist eine endliche Menge von Nichtterminalzeichen (oder Nichtterminalen, oder Variablen),
- Σ ist eine endliche Menge von Terminalzeichen (oder Terminalen), 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:

- $A, B, C, \ldots$ sind Nichtterminale,
- $a, b, c, \ldots$ (und Sonderzeichen wie $+, *, \ldots$) sind Terminale,

Definition 2.7
Eine Grammatik ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei

- V ist eine endliche Menge von Nichtterminalzeichen (oder Nichtterminalen, oder Variablen),
- Σ ist eine endliche Menge von Terminalzeichen (oder Terminalen), 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:

- $A, B, C, \ldots$ sind Nichtterminale,
- $a, b, c, \ldots$ (und Sonderzeichen wie $+, *, \ldots$) sind Terminale,
- $\alpha, \beta, \gamma, \ldots \in (V \cup \Sigma)^*$

Definition 2.7
Eine Grammatik ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei

- V ist eine endliche Menge von Nichtterminalzeichen (oder Nichtterminalen, oder Variablen),
- Σ ist eine endliche Menge von Terminalzeichen (oder Terminalen), 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:

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

Definition 2.7
Eine Grammatik ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei

- V ist eine endliche Menge von Nichtterminalzeichen (oder Nichtterminalen, oder Variablen),
- Σ ist eine endliche Menge von Terminalzeichen (oder Terminalen), 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:

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

Beispiel 2.8 (Arithmetische Ausdrücke)

Beispiel 2.8 (Arithmetische Ausdrücke)

- $V = \{\langle\text{Expr}\rangle, \langle\text{Term}\rangle, \langle\text{Faktor}\rangle\}$
- $\Sigma = \{a, b, c, +, *, (,)\}$

Beispiel 2.8 (Arithmetische Ausdrücke)

- $V = \{\langle\text{Expr}\rangle, \langle\text{Term}\rangle, \langle\text{Faktor}\rangle\}$
- $\Sigma = \{a, b, c, +, *, (,)\}$
- $S = \langle\text{Expr}\rangle$

Beispiel 2.8 (Arithmetische Ausdrücke)

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

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

Definition 2.9
Eine Grammatik $G = (V, \Sigma, P, S)$ induziert eine Ableitungsrelation $\to_G$ auf Wörtern über $V \cup \Sigma$:

$$\alpha \to_G \alpha'$$

Definition 2.9
Eine Grammatik $G = (V, \Sigma, P, S)$ induziert eine Ableitungsrelation $\to_G$ auf Wörtern über $V \cup \Sigma$:

$$\alpha \to_G \alpha'$$

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

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

Definition 2.9
Eine Grammatik $G = (V, \Sigma, P, S)$ induziert eine Ableitungsrelation $\to_G$ auf Wörtern über $V \cup \Sigma$:

$$\alpha \to_G \alpha'$$

gdw es eine Regel $\beta \to \beta'$ 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

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

Definition 2.9
Eine Grammatik $G = (V, \Sigma, P, S)$ induziert eine Ableitungsrelation $\to_G$ auf Wörtern über $V \cup \Sigma$:

$$\alpha \to_G \alpha'$$

gdw es eine Regel $\beta \to \beta'$ 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

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

Eine Sequenz $\alpha_1 \to_G \cdots \to_G \alpha_n$ ist eine Ableitung von α_n aus α_1.

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 $\alpha_1 \rightarrow_G \cdots \rightarrow_G \alpha_n$ 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.

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 $\alpha_1 \rightarrow_G \cdots \rightarrow_G \alpha_n$ 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)

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

Beispiel 2.10 (Arithmetische Ausdrücke)

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

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

Beispiel 2.10 (Arithmetische Ausdrücke)

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

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

Eine Ableitung:

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

$\to a * (b + c)$

Beispiel 2.10 (Arithmetische Ausdrücke)

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

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

Eine Ableitung:

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

$\to a * (b + c)$

Beispiel 2.10 (Arithmetische Ausdrücke)

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

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

Eine Ableitung:

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

$\to a * (b + c)$

Beispiel 2.10 (Arithmetische Ausdrücke)

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

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

Eine Ableitung:

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

$\to a * (b + c)$

Beispiel 2.10 (Arithmetische Ausdrücke)

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

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

Eine Ableitung:

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

$\to a * (b + c)$

Beispiel 2.10 (Arithmetische Ausdrücke)

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

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

Eine Ableitung:

$\langle \text{Expr} \rangle \to \langle \text{Term} \rangle \to \langle \text{Term} \rangle * \langle \text{Factor} \rangle$
$\to \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \to a * \langle \text{Factor} \rangle$
$\to a * (\langle \text{Expr} \rangle) \to a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle)$
$\to a * (\langle \text{Term} \rangle + \langle \text{Term} \rangle) \to a * (\langle \text{Factor} \rangle + \langle \text{Term} \rangle)$
$\to a * (b + \langle \text{Term} \rangle) \to a * (b + \langle \text{Factor} \rangle)$
$\to a * (b + c)$

Beispiel 2.11

Eine Grammatik für $\{a^n b^n c^n \mid n \geq 0\}$

Welche ist richtig?

$$
\begin{array}{rcl}
S & \to & abcS \mid \epsilon \\
ba & \to & ab \\
cb & \to & bc
\end{array}
\qquad
\begin{array}{rcl}
S & \to & aBSc \mid \epsilon \\
Ba & \to & aB \\
Bb & \to & bb \\
Bc & \to & bc
\end{array}
$$

Definition 2.12 (Iteration von $\to_G$)

$$\alpha \to_G^0 \alpha$$

Definition 2.12 (Iteration von $\to_G$)

$$\alpha \to_G^0 \alpha$$

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

Definition 2.12 (Iteration von $\to_G$)

$$\alpha \to_G^0 \alpha$$

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

Reflexive transitive Hülle:

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

Definition 2.12 (Iteration von $\to_G$)

$$\alpha \to_G^0 \alpha$$

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

Reflexive transitive Hülle:

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

Transitive Hülle:

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

Definition 2.12 (Iteration von $\to_G$)

$$\alpha \to_G^0 \alpha$$

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

Reflexive transitive Hülle:

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

Transitive Hülle:

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

Beispiel: $\langle \text{Expr} \rangle \to_G^{11} a * (b + c)$

Definition 2.12 (Iteration von $\to_G$)

$$\alpha \to_G^0 \alpha$$

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

Reflexive transitive Hülle:

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

Transitive Hülle:

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

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

Definition 2.12 (Iteration von $\to_G$)

$$\alpha \to_G^0 \alpha$$

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

Reflexive transitive Hülle:

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

Transitive Hülle:

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

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

Nach Definition: $L(G) = \{w \in \Sigma^* \mid S \to_G^* w\}$

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 $\alpha \to \beta$ außer $S \to \epsilon$ gilt $|\alpha| \leq |\beta|$

2.2 Die Chomsky-Hierarchie

Eine Grammatik G ist vom

Typ 0 immer.

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

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

32

2.2 Die Chomsky-Hierarchie

Eine Grammatik G ist vom

Typ 0 immer.

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

32

Beispiel 2.10 (Arithmetische Ausdrücke)

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

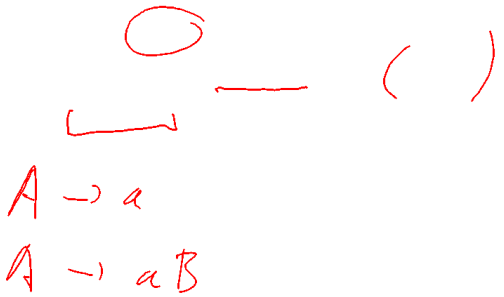
Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung:

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

Definition 2.12 (Iteration von $\to_G$)

$$\alpha \to_G^0 \alpha$$



2.2 Die Chomsky-Hierarchie

Eine Grammatik G ist vom

- Typ 0 immer.
- Typ 1 falls für jede Produktion $\alpha \to \beta$ außer $S \to \epsilon$ gilt $|\alpha| \leq |\beta|$
- Typ 2 falls G vom Typ 1 ist und für jede Produktion $\alpha \to \beta$ gilt $\alpha \in V$.
- Typ 3 falls G vom Typ 2 ist und für jede Produktion $\alpha \to \beta$ außer $S \to \epsilon$ 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

Satz 2.13

L(Typ 3) $\subset$ L(Typ 2) $\subset$ L(Typ 1) $\subset$ L(Typ 0)

Grammatiken und Sprachklassen:

Typ 3	Rechtslineare Grammatik	Reguläre Sprachen

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.

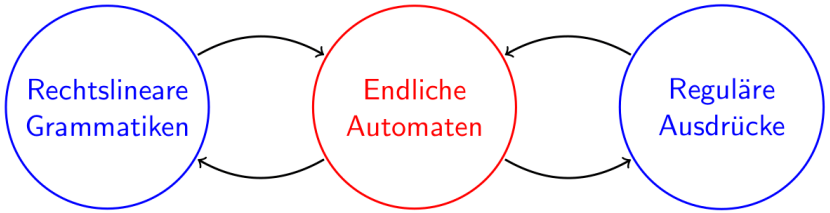
Das Wortproblem:

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.

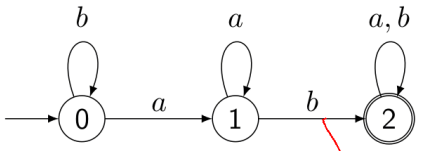
3. Reguläre Sprachen



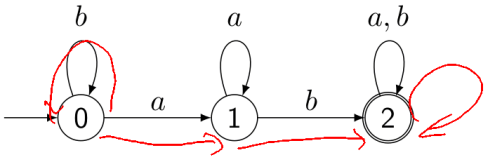
$A \to a$

$A \to bC$

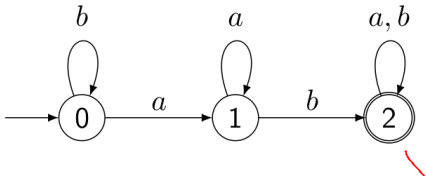
3.1 Deterministische endliche Automaten



Zustände Übergang

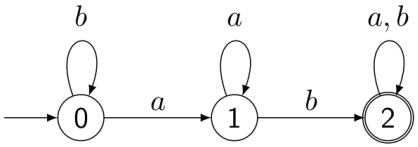


Eingabewort $baba \rightsquigarrow$ Zustandsfolge 0,0,1,2,2.



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

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



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

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

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

Definition 3.1
Ein deterministischer endlicher Automat (*deterministic finite automaton*, DFA) $M = (Q, \Sigma, \delta, q_0, F)$ besteht aus

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,

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 $\delta : \ Q \times \Sigma \to Q$,
- einem Startzustand $q_0 \in Q$, und

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 $\delta : \ Q \times \Sigma \to Q$,
- einem Startzustand $q_0 \in Q$, und
- einer Menge $F \subseteq Q$ von Endzuständen (akzeptierenden Zust.)

Die von M akzeptierte Sprache ist

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

beginnend in q_0 liefert w in einen Endzustand.