Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

AUTOMATENTHEORIE UND FORMALE SPRACHEN

Sommersemester 2021

11. April 2021

Prof. Dr. Steffen Reith

Theoretische Informatik
Studienbereich Angewandte Informatik
Hochschule **RheinMain**

ADMINISTRATIVES

TERMINE

Vorlesung:

Donnerstag 8^{15} - 9^{45} via BBB (ZAPP)

TERMINE

Vorlesung:

Donnerstag 8^{15} - 9^{45} via BBB (ZAPP)

Übung (Montag):

Gruppe	Zeit	Raum
AI Grp A	Mo 16^{00} - 17^{30}	UDE-C037 (Reinhard)
AI Grp B	Mo 17^{45} - 19^{15}	UDE-C035 (Reinhard)

ÜBER DEN DOZENTEN

- → Prof. Dr. Steffen Reith, geboren 1968, verheiratet, eine Tochter
- → Seit Sommersemester 2006 an der Hochschule RheinMain (FH Wiesbaden)

ÜBER DEN DOZENTEN

- → Prof. Dr. Steffen Reith, geboren 1968, verheiratet, eine Tochter
- → Seit Sommersemester 2006 an der Hochschule RheinMain (FH Wiesbaden)
- → Früher: Entwickler für kryptographische und mathematische Algorithmen für tief eingebettete System in KFZs.
- → Spezialgebiete: Komplexitätstheorie, Logik in der Informatik, eingebettete Systeme und Kryptographie (Zahlentheorie)

ÜBER DEN DOZENTEN

→ Prof. Dr. Steffen Reith, geboren 1968, verheiratet, eine Tochter
→ Seit Sommersemester 2006 an der Hochschule RheinMain (FH Wiesbaden)
→ Früher: Entwickler für kryptographische und mathematische Algorithmen für tief eingebettete System in KFZs.
→ Spezialgebiete: Komplexitätstheorie, Logik in der Informatik, eingebettete Systeme und Kryptographie (Zahlentheorie)
→ Abschlussarbeiten: Kryptographie, Kryptographie für eingebettete Systeme, paralleles Rechnen, Komplexitätstheorie, Logik in der Informatik

ÜBER DEN DOZENTEN

- → Prof. Dr. Steffen Reith, geboren 1968, verheiratet, eine Tochter
- → Seit Sommersemester 2006 an der Hochschule RheinMain (FH Wiesbaden)
- → Früher: Entwickler für kryptographische und mathematische Algorithmen für tief eingebettete System in KFZs.
- → Spezialgebiete: Komplexitätstheorie, Logik in der Informatik, eingebettete Systeme und Kryptographie (Zahlentheorie)
- → Abschlussarbeiten: Kryptographie, Kryptographie für eingebettete Systeme, paralleles Rechnen, Komplexitätstheorie, Logik in der Informatik

EMail: `Steffen.Reith@hs-rm.de`

Skype: `Steffen.Reith`

Büro: Raum C202

Sprechzeiten: Nach Vereinbarung (per BBB: immer!)

WEITERE INFORMATIONEN ZUR VORLESUNG

Webseite: `http://www.cs.hs-rm.de/~reith`

Auf der Webseite werden auch die Übungsblätter veröffentlicht (das erste **heute**!). Üblicherweise eine Woche Bearbeitungszeit. Bei **Feiertagschaos** bitte Dozenten befragen!

Literatur:

→ John E. Hopcroft and Rajeev Motwani and Jeffrey D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison Wesley Publishing Company, 2001

WEITERE INFORMATIONEN ZUR VORLESUNG

Webseite: `http://www.cs.hs-rm.de/~reith`

Auf der Webseite werden auch die Übungsblätter veröffentlicht (das erste **heute**!). Üblicherweise eine Woche Bearbeitungszeit. Bei **Feiertagschaos** bitte Dozenten befragen!

Literatur:

- → John E. Hopcroft and Rajeev Motwani and Jeffrey D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison Wesley Publishing Company, 2001
- → Uwe Schöning, Theoretische Informatik - kurzgefasst, Spektrum Akademischer Verlag, 2001

WEITERE INFORMATIONEN ZUR VORLESUNG

Webseite: `http://www.cs.hs-rm.de/~reith`

Auf der Webseite werden auch die Übungsblätter veröffentlicht (das erste **heute**!). Üblicherweise eine Woche Bearbeitungszeit. Bei **Feiertagschaos** bitte Dozenten befragen!

Literatur:

- → John E. Hopcroft and Rajeev Motwani and Jeffrey D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison Wesley Publishing Company, 2001
- → Uwe Schöning, Theoretische Informatik - kurzgefasst, Spektrum Akademischer Verlag, 2001
- → **Michael Sipser, Introduction to the Theory of Computation, Wadsworth Inc Fulfillment, 2. Auflage, 2005**

WEITERE INFORMATIONEN ZUR VORLESUNG

Webseite: `http://www.cs.hs-rm.de/~reith`

Auf der Webseite werden auch die Übungsblätter veröffentlicht (das erste **heute**!). Üblicherweise eine Woche Bearbeitungszeit. Bei **Feiertagschaos** bitte Dozenten befragen!

Literatur:

→ John E. Hopcroft and Rajeev Motwani and Jeffrey D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison Wesley Publishing Company, 2001

→ Uwe Schöning, Theoretische Informatik - kurzgefasst, Spektrum Akademischer Verlag, 2001

→ **Michael Sipser, Introduction to the Theory of Computation, Wadsworth Inc Fulfillment, 2. Auflage, 2005**

→ Juraj Hromkovič, Theoretische Informatik - Formale Sprachen, Berechenbarkeit, Komplexitätstheorie, Algorithmik, Kommunikation und Kryptographie, 3. Auflage, Teubner, 2007

WEITERE INFORMATIONEN ZUR VORLESUNG (II)

Ersatztermine:
Werden mit Hörern abgestimmt

WEITERE INFORMATIONEN ZUR VORLESUNG (II)

Ersatztermine:
Werden mit Hörern abgestimmt

Skript:
Wird in (sehr) unregelmäßigen Abständen auf der Webseite der Vorlesung verbessert und erweitert (eine alte Version ist heute online).

WEITERE INFORMATIONEN ZUR VORLESUNG (II)

Ersatztermine:
Werden mit Hörern abgestimmt

Skript:
Wird in (sehr) unregelmäßigen Abständen auf der Webseite der Vorlesung verbessert und erweitert (eine alte Version ist heute online).

Folien:
Einzelne (kleine) Teile der Vorlesung werden in Folienform zur Verfügung stehen. Folien, die vom Skript abweichen, werden auf der Webseite (nachträglich) zur Verfügung stehen.

WEITERE INFORMATIONEN ZUR VORLESUNG (II)

Ersatztermine:
Werden mit Hörern abgestimmt

Skript:
Wird in (sehr) unregelmäßigen Abständen auf der Webseite der Vorlesung verbessert und erweitert (eine alte Version ist heute online).

Folien:
Einzelne (kleine) Teile der Vorlesung werden in Folienform zur Verfügung stehen. Folien, die vom Skript abweichen, werden auf der Webseite (nachträglich) zur Verfügung stehen.

Eine eigene Mitschrift sollte *angefertigt* werden!

EIN ROTER FADEN

In der Vorlesung werden die folgenden Themen untersucht:

1. Einleitung (grundlegende Begriffe, L-Systeme)

EIN ROTER FADEN

In der Vorlesung werden die folgenden Themen untersucht:

1. Einleitung (grundlegende Begriffe, L-Systeme)
2. Die Chomsky-Hierarchie (Sprachklassen, Wortproblem)

EIN ROTER FADEN

In der Vorlesung werden die folgenden Themen untersucht:

1. Einleitung (grundlegende Begriffe, L-Systeme)
2. Die Chomsky-Hierarchie (Sprachklassen, Wortproblem)
3. Endliche Automaten und reguläre Sprachen (Pumping Lemma)

EIN ROTER FADEN

In der Vorlesung werden die folgenden Themen untersucht:

1. Einleitung (grundlegende Begriffe, L-Systeme)
2. Die Chomsky-Hierarchie (Sprachklassen, Wortproblem)
3. Endliche Automaten und reguläre Sprachen (Pumping Lemma)
4. Kontextfreie Sprachen (Normalformen, Kellerautomaten)

EIN ROTER FADEN

In der Vorlesung werden die folgenden Themen untersucht:

1. Einleitung (grundlegende Begriffe, L-Systeme)
2. Die Chomsky-Hierarchie (Sprachklassen, Wortproblem)
3. Endliche Automaten und reguläre Sprachen (Pumping Lemma)
4. Kontextfreie Sprachen (Normalformen, Kellerautomaten)
5. Kontextsensitive- und Typ0-Sprachen (Turingmaschinen, Unentscheidbarkeit)

EIN ROTER FADEN

In der Vorlesung werden die folgenden Themen untersucht:

1. Einleitung (grundlegende Begriffe, L-Systeme)
2. Die Chomsky-Hierarchie (Sprachklassen, Wortproblem)
3. Endliche Automaten und reguläre Sprachen (Pumping Lemma)
4. Kontextfreie Sprachen (Normalformen, Kellerautomaten)
5. Kontextsensitive- und Typ0-Sprachen (Turingmaschinen, Unentscheidbarkeit)
6. Komplexität ($\mathbf{P}$ vs. $\mathbf{NP}$)

WARUM THEORETISCHE INFORMATIK?

Theoretische Informatik wird (wohl aufgrund der mathematischen Prägung) oft als **„schwach motiviert“**, **„langweilig“** und **„nutzlos“** beschrieben.

WARUM THEORETISCHE INFORMATIK?

Theoretische Informatik wird (wohl aufgrund der mathematischen Prägung) oft als **„schwach motiviert“**, **„langweilig“** und **„nutzlos“** beschrieben.

Warum lohnt sich die Theoretische Informatik?

WARUM THEORETISCHE INFORMATIK?

Theoretische Informatik wird (wohl aufgrund der mathematischen Prägung) oft als **„schwach motiviert“**, **„langweilig“** und **„nutzlos“** beschrieben.

Warum lohnt sich die Theoretische Informatik?

→ Konkrete Technologien ändern sich sehr schnell, deshalb sollte man die extrem langlebigen Konzepte verstehen.

WARUM THEORETISCHE INFORMATIK?

Theoretische Informatik wird (wohl aufgrund der mathematischen Prägung) oft als **„schwach motiviert“**, **„langweilig“** und **„nutzlos“** beschrieben.

Warum lohnt sich die Theoretische Informatik?

- → Konkrete Technologien ändern sich sehr schnell, deshalb sollte man die extrem langlebigen Konzepte verstehen.
- → Hintergrundinformationen ermöglichen Chancen und Grenzen von Technologien zu verstehen.

WARUM THEORETISCHE INFORMATIK?

Theoretische Informatik wird (wohl aufgrund der mathematischen Prägung) oft als **„schwach motiviert“**, **„langweilig“** und **„nutzlos“** beschrieben.

Warum lohnt sich die Theoretische Informatik?

- → Konkrete Technologien ändern sich sehr schnell, deshalb sollte man die extrem langlebigen Konzepte verstehen.
- → Hintergrundinformationen ermöglichen Chancen und Grenzen von Technologien zu verstehen.
- → Theoretische Informatik gibt Hinweise, welche Wege zu keiner Lösung führen werden.

WARUM THEORETISCHE INFORMATIK?

Theoretische Informatik wird (wohl aufgrund der mathematischen Prägung) oft als **„schwach motiviert“**, **„langweilig“** und **„nutzlos“** beschrieben.

Warum lohnt sich die Theoretische Informatik?

→ Konkrete Technologien ändern sich sehr schnell, deshalb sollte man die extrem langlebigen Konzepte verstehen.

→ Hintergrundinformationen ermöglichen Chancen und Grenzen von Technologien zu verstehen.

→ Theoretische Informatik gibt Hinweise, welche Wege zu keiner Lösung führen werden.

→ Verbesserung des strukturierten Denkens und der Problemlösungskompetenz.

WARUM THEORETISCHE INFORMATIK?

Theoretische Informatik wird (wohl aufgrund der mathematischen Prägung) oft als **„schwach motiviert“**, **„langweilig“** und **„nutzlos“** beschrieben.

Warum lohnt sich die Theoretische Informatik?

- → Konkrete Technologien ändern sich sehr schnell, deshalb sollte man die extrem langlebigen Konzepte verstehen.
- → Hintergrundinformationen ermöglichen Chancen und Grenzen von Technologien zu verstehen.
- → Theoretische Informatik gibt Hinweise, welche Wege zu keiner Lösung führen werden.
- → Verbesserung des strukturierten Denkens und der Problemlösungskompetenz.
- → (Tiefgreifende) Ideen führen zu schnellen Algorithmen

PROGRAMMIERSPRACHEN - FORTRAN77

```
c     Erstellt von Gilberto E. Urroz
c000000000111111111122222222223333333333444444444455555555556666666666777
c234567890123456789012345678901234567890123456789012345678901234567890123
      program function
c     --- declaration of variables
      real x, y
c     --- show function
      print*,"========================================="
      print*,"Calculate the function y = f(x) defined as"
      print*,"           y = x+1  if x <  1"
      print*,"           y = 2-x  if x >= 1"
      print*,"========================================="
c     --- request x as input
      print*,"enter a value of x:"
      read*,x
c     --- evaluation of function
      if (x.lt.1.0) then
      y = x+1
      else
      y = 2-x
      end if
c     --- print result
      print*,"the corresponding value of y is: ",y
c     --- end program
      end
```

PROGRAMMIERSPRACHEN - FORTRAN77

```
c     Erstellt von Gilberto E. Urroz
c0000000001111111111222222222233333333334444444444555555555566666666667777
c2345678901234567890123456789012345678901234567890123456789012345678901234567890123
      program function
c     --- declaration of variables
      real x, y
c     --- show function
      print*,"=========================================="
      print*,"Calculate the function y = f(x) defined as"
      print*,"           y = x+1  if x <  1"
      print*,"           y = 2-x  if x >= 1"
      print*,"=========================================="
c     --- request x as input
      print*,"enter a value of x:"
      read*,x
c     --- evaluation of function
      if (x.lt.1.0) then
      y = x+1
      else
      y = 2-x
      end if
c     --- print result
      print*,"the corresponding value of y is: ",y
c     --- end program
      end
^^I^^I^^I
```

Warum haben wir heute Programmiersprachen mit „schönerer“ Syntax?

SPIELREGELN

→ *Rechner* und Handys sind zu Beginn der Veranstaltung **aus**

SPIELREGELN

- → *Rechner* und Handys sind zu Beginn der Veranstaltung **aus**
- → Wir (Dozent + Hörer) sind **pünktlich**

SPIELREGELN

- → *Rechner* und Handys sind zu Beginn der Veranstaltung **aus**
- → Wir (Dozent + Hörer) sind **pünktlich**
- → Es *redet nur eine Person*

SPIELREGELN

- → *Rechner* und Handys sind zu Beginn der Veranstaltung **aus**
- → Wir (Dozent + Hörer) sind **pünktlich**
- → Es *redet nur eine Person*
- → Bei Fragen und Problemen **sofort melden / fragen**

SPIELREGELN

- → *Rechner* und Handys sind zu Beginn der Veranstaltung **aus**
- → Wir (Dozent + Hörer) sind **pünktlich**
- → Es *redet nur eine Person*
- → Bei Fragen und Problemen **sofort melden / fragen**
- → Es wird Eigeninitiative und selbstständiges Arbeiten erwartet

SPIELREGELN

- *Rechner* und Handys sind zu Beginn der Veranstaltung **aus**
- Wir (Dozent + Hörer) sind **pünktlich**
- Es *redet nur eine Person*
- Bei Fragen und Problemen **sofort melden / fragen**
- Es wird Eigeninitiative und selbstständiges Arbeiten erwartet
- Eine Vorlesung ist keine (wöchentliche) Fernsehserie
  - Eine Vorlesung wird von **den Hörern** und vom Dozenten **gestaltet**
  - aktive Mitarbeit erwünscht und erforderlich
  - Der Dozent will motiviert werden
  - Umfangreiche Vor- und Nachbereitung notwendig
  - Lernen kurz vor der Klausur ist tötlich! (kontinuierliches Lernen)

SPIELREGELN

- *Rechner* und Handys sind zu Beginn der Veranstaltung **aus**
- Wir (Dozent + Hörer) sind **pünktlich**
- Es *redet nur eine Person*
- Bei Fragen und Problemen **sofort melden / fragen**
- Es wird Eigeninitiative und selbstständiges Arbeiten erwartet
- Eine Vorlesung ist keine (wöchentliche) Fernsehserie
  - Eine Vorlesung wird von **den Hörern** und vom Dozenten **gestaltet**
  - aktive Mitarbeit erwünscht und erforderlich
  - Der Dozent will motiviert werden
  - Umfangreiche Vor- und Nachbereitung notwendig
  - Lernen kurz vor der Klausur ist tötlich! (kontinuierliches Lernen)
- Vergessen Sie den (angeblichen) Konflikt von Theorie und Praxis

SPIELREGELN

- → *Rechner* und Handys sind zu Beginn der Veranstaltung **aus**
- → Wir (Dozent + Hörer) sind **pünktlich**
- → Es *redet nur eine Person*
- → Bei Fragen und Problemen **sofort melden / fragen**
- → Es wird Eigeninitiative und selbstständiges Arbeiten erwartet
- → Eine Vorlesung ist keine (wöchentliche) Fernsehserie
  - → Eine Vorlesung wird von **den Hörern** und vom Dozenten **gestaltet**
  - → aktive Mitarbeit erwünscht und erforderlich
  - → Der Dozent will motiviert werden
  - → Umfangreiche Vor- und Nachbereitung notwendig
  - → Lernen kurz vor der Klausur ist tötlich! (kontinuierliches Lernen)
- → Vergessen Sie den (angeblichen) Konflikt von Theorie und Praxis

Was wünschen Sie sich?

NATÜRLICHE VS. FORMALE SPRACHEN

NATÜRLICHE SPRACHEN

Natürliche Sprachen legen ihre Struktur durch

→ die Regeln einer **Grammatik**

NATÜRLICHE SPRACHEN

Natürliche Sprachen legen ihre Struktur durch

- → die Regeln einer **Grammatik**
- → und eine Menge von **erlaubten Worten** ($\triangleq$ Strings gebildet aus Buchstaben)

NATÜRLICHE SPRACHEN

Natürliche Sprachen legen ihre Struktur durch

- → die Regeln einer **Grammatik**
- → und eine Menge von **erlaubten Worten** ($\triangleq$ Strings gebildet aus Buchstaben)

fest.

Allerdings müssen syntaktisch korrekte Sätze einer natürlichen Sprache keinen Sinn tragen:

- → Wiesbaden wohnt weiterhin weich

NATÜRLICHE SPRACHEN

Natürliche Sprachen legen ihre Struktur durch

→ die Regeln einer **Grammatik**
→ und eine Menge von **erlaubten Worten** ($\triangleq$ Strings gebildet aus Buchstaben)

fest.

Allerdings müssen syntaktisch korrekte Sätze einer natürlichen Sprache keinen Sinn tragen:

→ Wiesbaden wohnt weiterhin weich
→ Der bissige Student jagt die verschlafene Mensa

NATÜRLICHE SPRACHEN

Natürliche Sprachen legen ihre Struktur durch

- → die Regeln einer **Grammatik**
- → und eine Menge von **erlaubten Worten** ($\triangleq$ Strings gebildet aus Buchstaben)

fest.

Allerdings müssen syntaktisch korrekte Sätze einer natürlichen Sprache keinen Sinn tragen:

- → Wiesbaden wohnt weiterhin weich
- → Der bissige Student jagt die verschlafene Mensa

$\Rightarrow$ syntakisch korrekte Sätze müssen keinen Sinn ($\triangleq$ Semantik) tragen.

NATÜRLICHE SPRACHEN

Natürliche Sprachen legen ihre Struktur durch

- → die Regeln einer **Grammatik**
- → und eine Menge von **erlaubten Worten** ($\triangleq$ Strings gebildet aus Buchstaben)

fest.

Allerdings müssen syntaktisch korrekte Sätze einer natürlichen Sprache keinen Sinn tragen:

- → Wiesbaden wohnt weiterhin weich
- → Der bissige Student jagt die verschlafene Mensa

$\Rightarrow$ syntakisch korrekte Sätze müssen keinen Sinn ($\triangleq$ Semantik) tragen.

Wie kann man diese Beobachtungen in der Informatik ausnutzen?

FORMALE REGELN ZUR ERZEUGUNG EINER SPRACHE

Der Linguist Noam Chomsky hatte folgende Idee:

Korrekte Sätze einer (natürlichen) Sprache sollen durch ein **(endliches System) von formalen Regeln** erzeugt werden.

FORMALE REGELN ZUR ERZEUGUNG EINER SPRACHE

Der Linguist Noam Chomsky hatte folgende Idee:

Korrekte Sätze einer (natürlichen) Sprache sollen durch ein **(endliches System) von formalen Regeln** erzeugt werden.

Bis heute ist diese Idee

→ in der Linguistik umstritten, aber

FORMALE REGELN ZUR ERZEUGUNG EINER SPRACHE

Der Linguist Noam Chomsky hatte folgende Idee:

Korrekte Sätze einer (natürlichen) Sprache sollen durch ein **(endliches System) von formalen Regeln** erzeugt werden.

Bis heute ist diese Idee

→ in der Linguistik umstritten, aber

→ extrem bedeutsam in der Informatik.

FORMALE REGELN ZUR ERZEUGUNG EINER SPRACHE

Der Linguist Noam Chomsky hatte folgende Idee:

Korrekte Sätze einer (natürlichen) Sprache sollen durch ein **(endliches System) von formalen Regeln** erzeugt werden.

Bis heute ist diese Idee

- → in der Linguistik umstritten, aber
- → extrem bedeutsam in der Informatik.

Basis für z.B. alle Programmiersprachen / Compilerbau, Auszeichnungssprachen (SGML, XML, HTML, . . .).

FORMALE REGELN ZUR ERZEUGUNG EINER SPRACHE

Der Linguist Noam Chomsky hatte folgende Idee:

Korrekte Sätze einer (natürlichen) Sprache sollen durch ein **(endliches System) von formalen Regeln** erzeugt werden.

Bis heute ist diese Idee

→ in der Linguistik umstritten, aber

→ extrem bedeutsam in der Informatik.

Basis für z.B. alle Programmiersprachen / Compilerbau, Auszeichnungssprachen (SGML, XML, HTML, . . .).

Ähnlich sind die sogenannten **(Semi) Thue Systeme**, die heute z.B. in Spezialformen in der Computergraphik Bedeutung erlangt haben.

EINIGE GRUNDLEGENDE BEGRIFFE

Eine endliche Menge Σ heißt **Alphabet**.

EINIGE GRUNDLEGENDE BEGRIFFE

Eine endliche Menge Σ heißt **Alphabet**. Die **Elemente** von Σ werden **Buchstaben** genannt.

EINIGE GRUNDLEGENDE BEGRIFFE

Eine endliche Menge Σ heißt **Alphabet**. Die **Elemente** von Σ werden **Buchstaben** genannt. Eine Folge von Buchstaben nennt man **Wort** (über Σ).

EINIGE GRUNDLEGENDE BEGRIFFE

Eine endliche Menge Σ heißt **Alphabet**. Die **Elemente** von Σ werden **Buchstaben** genannt. Eine Folge von Buchstaben nennt man **Wort** (über Σ). Eine beliebige Menge von Worten über Σ nennt man dann eine **(formale) Sprache**.

EINIGE GRUNDLEGENDE BEGRIFFE

Eine endliche Menge Σ heißt **Alphabet**. Die **Elemente** von Σ werden **Buchstaben** genannt. Eine Folge von Buchstaben nennt man **Wort** (über Σ). Eine beliebige Menge von Worten über Σ nennt man dann eine **(formale) Sprache**.

Beispiel (arithmetische Ausdrücke)

Sei $\Sigma = \{), (, +, -, *, /, x\}$ und EXPR die Menge aller korrekten arithmetischen Ausdrücke.

EINIGE GRUNDLEGENDE BEGRIFFE

Eine endliche Menge Σ heißt **Alphabet**. Die **Elemente** von Σ werden **Buchstaben** genannt. Eine Folge von Buchstaben nennt man **Wort** (über Σ). Eine beliebige Menge von Worten über Σ nennt man dann eine **(formale) Sprache**.

Beispiel (arithmetische Ausdrücke)

Sei $\Sigma = \{), (, +, -, *, /, x\}$ und EXPR die Menge aller korrekten arithmetischen Ausdrücke. Damit gilt

- $(x - x) \in$ EXPR
- $((x + x) * x)/x \in$ EXPR
- $))(x-) * x \notin$ EXPR

EXPR ist eine Menge von Worten über Σ, also kann man EXPR als **formale Sprache** (über $\{), (, +, -, *, /, x\}$) bezeichnen.

WEITERE BEISPIELE FÜR FORMALE SPRACHEN

Beispiel (Zahlenmengen)

Sei $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, dann sind die folgenden Mengen auch formale Sprachen über Σ:

WEITERE BEISPIELE FÜR FORMALE SPRACHEN

Beispiel (Zahlenmengen)

Sei $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, dann sind die folgenden Mengen auch formale Sprachen über Σ:

- → PRIMES $= \{2, 3, 5, 7, 11, 13, 17, 19, 23, \ldots\}$
- → EVEN $= \{0, 2, 4, 6, 8, 10, 12, 14, 16, 18, \ldots\}$
- → 2POT $= \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, \ldots\}$

WEITERE BEISPIELE FÜR FORMALE SPRACHEN (II))

Beispiel (Wortmengen über $\{a, b\}$)

Sei $\Sigma = \{a, b\}$, dann sind die folgenden Mengen auch formale Sprachen über Σ:

WEITERE BEISPIELE FÜR FORMALE SPRACHEN (II))

Beispiel (Wortmengen über $\{a, b\}$)

Sei $\Sigma = \{a, b\}$, dann sind die folgenden Mengen auch formale Sprachen über Σ:

- → BRACKET $= \{ab, aabb, aaabbb, aaaabbbb, \ldots\}$
- → UODD $= \{a, aaa, aaaaa, aaaaaaa, aaaaaaaaa, \ldots\}$
- → $\Sigma^* =$ ALL $= \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, \ldots\}$

GRAMMATIKEN UND AUTOMATEN

(Formale) Sprachen enthalten meist unendlich viele Wörter

GRAMMATIKEN UND AUTOMATEN

(Formale) Sprachen enthalten meist unendlich viele Wörter

→ Wir brauchen endlich viele **Erzeugungsregeln**, um (algorithmisch) mit formalen Sprachen umgehen zu können. Die Rolle der Regeln übernehmen **Grammatiken**.

GRAMMATIKEN UND AUTOMATEN

(Formale) Sprachen enthalten meist unendlich viele Wörter

- → Wir brauchen endlich viele **Erzeugungsregeln**, um (algorithmisch) mit formalen Sprachen umgehen zu können. Die Rolle der Regeln übernehmen **Grammatiken**.
- → Weiterhin werden **Erkenner** benötigt, die entscheiden, ob ein **Wort zu einer Sprache gehört**. Die Rolle der Erkenner spielen die **Automaten**, die wir in dieser Vorlesung studieren.

TEIL EINER NATÜRLICHEN SPRACHE

Beispiel (Eine Grammatik)

$$
\begin{array}{rcl}
\textbf{Satz} & \rightarrow & \textbf{Subjekt Prädikat Objekt} \\
\textbf{Subjekt} & \rightarrow & \textbf{Artikel Attribut Substantiv} \\
\textbf{Artikel} & \rightarrow & \epsilon \mid \text{der} \mid \text{die} \mid \text{das} \\
\textbf{Attribut} & \rightarrow & \epsilon \\
\textbf{Attribut} & \rightarrow & \textbf{Adjektiv} \\
\textbf{Attribut} & \rightarrow & \textbf{Adjektiv Attribut} \\
\textbf{Adjektiv} & \rightarrow & \text{kleine} \mid \text{bissige} \mid \text{verschlafene} \\
\textbf{Substantiv} & \rightarrow & \text{Student} \mid \text{Katze} \\
\textbf{Prädikat} & \rightarrow & \text{jagt} \mid \text{betritt} \\
\textbf{Objekt} & \rightarrow & \textbf{Artikel Attribut Substantiv}
\end{array}
$$

TEIL EINER NATÜRLICHEN SPRACHE

Beispiel (Eine Grammatik)

$$
\begin{array}{rcl}
\textbf{Satz} & \to & \textbf{Subjekt Prädikat Objekt} \\
\textbf{Subjekt} & \to & \textbf{Artikel Attribut Substantiv} \\
\textbf{Artikel} & \to & \epsilon \mid \text{der} \mid \text{die} \mid \text{das} \\
\textbf{Attribut} & \to & \epsilon \\
\textbf{Attribut} & \to & \textbf{Adjektiv} \\
\textbf{Attribut} & \to & \textbf{Adjektiv Attribut} \\
\textbf{Adjektiv} & \to & \text{kleine} \mid \text{bissige} \mid \text{verschlafene} \\
\textbf{Substantiv} & \to & \text{Student} \mid \text{Katze} \\
\textbf{Prädikat} & \to & \text{jagt} \mid \text{betritt} \\
\textbf{Objekt} & \to & \textbf{Artikel Attribut Substantiv}
\end{array}
$$

Das Symbol „|“ markiert eine Alternative, d.h. $\mathbf{A} \to \mathbf{B} \mid \mathbf{C}$ ist Abkürzung für die beiden Regeln $\mathbf{A} \to \mathbf{B}$ und $\mathbf{A} \to \mathbf{C}$

TEIL EINER NATÜRLICHEN SPRACHE (II)

Durch Anwendung der Regeln und Ersetzung der fett gedruckten Wörter können z.B. die folgenden Sätze gebildet werden:

TEIL EINER NATÜRLICHEN SPRACHE (II)

Durch Anwendung der Regeln und Ersetzung der fett gedruckten Wörter können z.B. die folgenden Sätze gebildet werden:

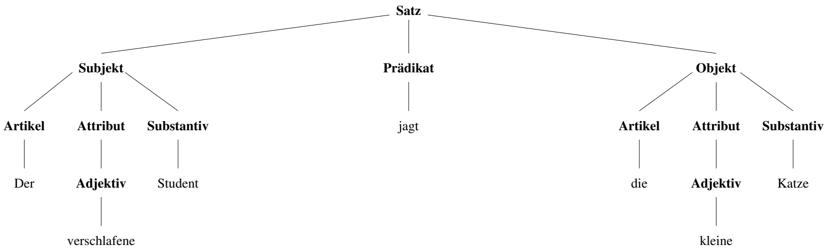
→ Der kleine bissige Student betritt die verschlafene Mensa
→ Der verschlafene Student jagt die kleine Katze

TEIL EINER NATÜRLICHEN SPRACHE (II)

Durch Anwendung der Regeln und Ersetzung der fett gedruckten Wörter können z.B. die folgenden Sätze gebildet werden:

- → Der kleine bissige Student betritt die verschlafene Mensa
- → Der verschlafene Student jagt die kleine Katze

Mit **Syntaxbäumen** man man die **Ableitungschritte graphisch** verdeutlichen:



L-SYSTEME

L-SYSTEME

→ Die **L-Systeme** wurden 1968 durch **Aristid Lindenmeyer** als mathematisches Modell des Pflanzenwachstums eingeführt.

L-SYSTEME

→ Die **L-Systeme** wurden 1968 durch **Aristid Lindenmeyer** als mathematisches Modell des Pflanzenwachstums eingeführt.

→ L-Systeme werden heute in der Computergraphik benutzt, um natürlich wirkende Pflanzen schnell generieren zu können.

L-SYSTEME

- Die **L-Systeme** wurden 1968 durch **Aristid Lindenmeyer** als mathematisches Modell des Pflanzenwachstums eingeführt.
- L-Systeme werden heute in der Computergraphik benutzt, um natürlich wirkende Pflanzen schnell generieren zu können.
- Hier betrachten wir die einfachste Klasse von L-Systemen, die so genannten D0L-System.

L-SYSTEME

- Die **L-Systeme** wurden 1968 durch **Aristid Lindenmeyer** als mathematisches Modell des Pflanzenwachstums eingeführt.
- L-Systeme werden heute in der Computergraphik benutzt, um natürlich wirkende Pflanzen schnell generieren zu können.
- Hier betrachten wir die einfachste Klasse von L-Systemen, die so genannten D0L-System.
  - Die Regeln sind deterministisch, d.h. für jeden Buchstaben gibt es **genau eine Regel**.

L-SYSTEME

- Die **L-Systeme** wurden 1968 durch **Aristid Lindenmeyer** als mathematisches Modell des Pflanzenwachstums eingeführt.
- L-Systeme werden heute in der Computergraphik benutzt, um natürlich wirkende Pflanzen schnell generieren zu können.
- Hier betrachten wir die einfachste Klasse von L-Systemen, die so genannten D0L-System.
  - Die Regeln sind deterministisch, d.h. für jeden Buchstaben gibt es **genau eine Regel**.
  - Die Regeln sind kontextfrei, d.h. **Ersetzungen hängen nicht** von den **umgebenden Buchstaben** ($\triangleq$ Kontext) ab.

GRUNDLEGENDE BEGRIFFE UND EIGENSCHAFTEN

Definition (0L-Systeme)

→ Mit Σ^* bezeichnen wir die Menge **aller Wörter** über Σ.

GRUNDLEGENDE BEGRIFFE UND EIGENSCHAFTEN

Definition (0L-Systeme)

- Mit Σ^* bezeichnen wir die Menge **aller Wörter** über Σ.
- Ein **0L-System** G ist ein Tripel $G = (\Sigma, \omega, P)$, wobei

GRUNDLEGENDE BEGRIFFE UND EIGENSCHAFTEN

Definition (0L-Systeme)

- Mit Σ^* bezeichnen wir die Menge **aller Wörter** über Σ.
- Ein **0L-System** G ist ein Tripel $G = (\Sigma, \omega, P)$, wobei
  - Σ das **Alphabet**,

GRUNDLEGENDE BEGRIFFE UND EIGENSCHAFTEN

Definition (0L-Systeme)

- Mit Σ^* bezeichnen wir die Menge **aller Wörter** über Σ.
- Ein **0L-System** G ist ein Tripel $G = (\Sigma, \omega, P)$, wobei
  - Σ das **Alphabet**, ω das **Axiom** und
  - $P \subseteq \Sigma \times \Sigma^*$ die Menge der **Produktionen**.

GRUNDLEGENDE BEGRIFFE UND EIGENSCHAFTEN

Definition (0L-Systeme)

- Mit Σ^* bezeichnen wir die Menge **aller Wörter** über Σ.
- Ein **0L-System** G ist ein Tripel $G = (\Sigma, \omega, P)$, wobei
  - Σ das **Alphabet**, ω das **Axiom** und
  - $P \subseteq \Sigma \times \Sigma^*$ die Menge der **Produktionen**.
- Eine Produktion $(a, \chi) \in P$ wird als $a \to \chi$ geschrieben.

GRUNDLEGENDE BEGRIFFE UND EIGENSCHAFTEN

Definition (0L-Systeme)

- Mit Σ^* bezeichnen wir die Menge **aller Wörter** über Σ.
- Ein **0L-System** G ist ein Tripel $G = (\Sigma, \omega, P)$, wobei
  - Σ das **Alphabet**, ω das **Axiom** und
  - $P \subseteq \Sigma \times \Sigma^*$ die Menge der **Produktionen**.
- Eine Produktion $(a, \chi) \in P$ wird als $a \rightarrow \chi$ geschrieben. Der Buchstabe a heißt **Vorgänger** und χ **Nachfolger** dieser Produktion.

GRUNDLEGENDE BEGRIFFE UND EIGENSCHAFTEN

Definition (0L-Systeme)

- Mit Σ^* bezeichnen wir die Menge **aller Wörter** über Σ.
- Ein **0L-System** G ist ein Tripel $G = (\Sigma, \omega, P)$, wobei
  - Σ das **Alphabet**, ω das **Axiom** und
  - $P \subseteq \Sigma \times \Sigma^*$ die Menge der **Produktionen**.
- Eine Produktion $(a, \chi) \in P$ wird als $a \to \chi$ geschrieben. Der Buchstabe a heißt **Vorgänger** und χ **Nachfolger** dieser Produktion.
- Für **jeden** Buchstaben $a \in \Sigma$ **existiert eine Produktion** $(a, \chi) \in P$.

GRUNDLEGENDE BEGRIFFE UND EIGENSCHAFTEN

Definition (0L-Systeme)

- Mit Σ^* bezeichnen wir die Menge **aller Wörter** über Σ.
- Ein **0L-System** G ist ein Tripel $G = (\Sigma, \omega, P)$, wobei
  - Σ das **Alphabet**, ω das **Axiom** und
  - $P \subseteq \Sigma \times \Sigma^*$ die Menge der **Produktionen**.
- Eine Produktion $(a, \chi) \in P$ wird als $a \to \chi$ geschrieben. Der Buchstabe a heißt **Vorgänger** und χ **Nachfolger** dieser Produktion.
- Für **jeden** Buchstaben $a \in \Sigma$ **existiert eine Produktion** $(a, \chi) \in P$.
- Ein $0L$-System heißt **deterministisch**, wenn es für jeden Buchstaben $a \in \Sigma$ nur **genau eine** Produktion $(a, \chi) \in P$ gibt.

D0L-SYSTEME (II)

Definition

Deterministische 0L-Systeme heißen **D0L**-Systeme.

D0L-SYSTEME (II)

Definition

Deterministische 0L-Systeme heißen **D0L**-Systeme.

Definition (Ableitung)

Sei $\mu = a_1 \ldots a_m$ ein beliebiges Wort über Σ, dann kann $\nu = \chi_1 \ldots \chi_m$ aus μ **abgeleitet** werden, wenn

D0L-SYSTEME (II)

Definition

Deterministische 0L-Systeme heißen **D0L**-Systeme.

Definition (Ableitung)

Sei $\mu = a_1 \dots a_m$ ein beliebiges Wort über Σ, dann kann $\nu = \chi_1 \dots \chi_m$ aus μ **abgeleitet** werden, wenn

→ **für alle** $i = 1, \dots, m$ $(a_i, \chi_i) \in P$ gilt, wobei

D0L-SYSTEME (II)

Definition

Deterministische 0L-Systeme heißen **D0L**-Systeme.

Definition (Ableitung)

Sei $\mu = a_1 \dots a_m$ ein beliebiges Wort über Σ, dann kann $\nu = \chi_1 \dots \chi_m$ aus μ **abgeleitet** werden, wenn

- **für alle** $i = 1, \dots, m$ $(a_i, \chi_i) \in P$ gilt, wobei
- man $\mu \vdash \nu$ schreibt.

DOL-SYSTEME (II)

Definition

Deterministische 0L-Systeme heißen **D0L**-Systeme.

Definition (Ableitung)

Sei $\mu = a_1 \dots a_m$ ein beliebiges Wort über Σ, dann kann $\nu = \chi_1 \dots \chi_m$ aus μ **abgeleitet** werden, wenn

- **für alle** $i = 1, \dots, m$ $(a_i, \chi_i) \in P$ gilt, wobei
- man $\mu \longmapsto \nu$ schreibt.
- Ein Wort ν heißt **von** G **generiert**, wenn es in **endlich** vielen Schritten aus dem Axiom abgeleitet werden kann.

D0L-SYSTEME (III)

Geben wir aus **Bequemlichkeitsgründen** für einen Buchstaben a keine Produktion an, dann gilt **implizit** $(a, a) \in P$.

DOL-SYSTEME (III)

Geben wir aus **Bequemlichkeitsgründen** für einen Buchstaben a keine Produktion an, dann gilt **implizit** $(a, a) \in P$.

Achtung: Alle Regeln aus P werden **gleichzeitig** angewendet.

D0L-SYSTEME (III)

Geben wir aus **Bequemlichkeitsgründen** für einen Buchstaben a keine Produktion an, dann gilt **implizit** $(a, a) \in P$.

Achtung: Alle Regeln aus P werden **gleichzeitig** angewendet.

Wird ein Wort ν von $G = (\Sigma, \omega, P)$ generiert, dann können wir also

$$\omega \longmapsto \mu_1 \longmapsto \mu_2 \longmapsto \ldots \longmapsto \mu_n = \nu$$

schreiben (kurz: $\omega \overset{*}{\longmapsto} \nu$).

EIN BEISPIEL

Sei $G = (\Sigma, \omega, P)$, wobei

- $\Sigma = \{a, b, c\}$,
- $\omega = abc$ und
- $P = \{a \to aa, b \to bb, c \to cc\}$.

Mit Hilfe dieses D0L-Systems können Worte der Form

EIN BEISPIEL

Sei $G = (\Sigma, \omega, P)$, wobei

- $\Sigma = \{a, b, c\}$,
- $\omega = abc$ und
- $P = \{a \rightarrow aa, b \rightarrow bb, c \rightarrow cc\}$.

Mit Hilfe dieses D0L-Systems können Worte der Form

$$a^{2^n} b^{2^n} c^{2^n}$$

für $n \geq 0$ abgeleitet werden.

EIN BEISPIEL

Sei $G = (\Sigma, \omega, P)$, wobei

- $\Sigma = \{a, b, c\}$,
- $\omega = abc$ und
- $P = \{a \rightarrow aa, b \rightarrow bb, c \rightarrow cc\}$.

Mit Hilfe dieses D0L-Systems können Worte der Form

$$a^{2^n} b^{2^n} c^{2^n}$$

für $n \geq 0$ abgeleitet werden.

Bemerkung: a^n ist die Abkürzung für $\underbrace{aaa \ldots a}_{n-\text{mal}}$

TURTLE-GRAPHIK

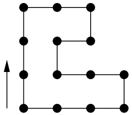
Sei δ **ein beliebiger Winkel**, dann werden die Buchstaben $\mathtt{F}$, $\mathtt{f}$, $+$ und $-$ wie folgt interpretiert:

TURTLE-GRAPHIK

Sei δ **ein beliebiger Winkel**, dann werden die Buchstaben $\mathtt{F}$, $\mathtt{f}$, $+$ und $-$ wie folgt interpretiert:

$\mathtt{F}$	**Bewege** den **Stift** um die Länge d und **zeichne** eine **Linie**
$\mathtt{f}$	**Bewege** den **Stift** um die Länge d und **zeichne keine Linie**
$-$	**drehe** um δ Grad nach **rechts**
$+$	**drehe** um δ Grad nach **links**

Mit $\delta = 90°$ wird $\mathtt{FFF} - \mathtt{FF} - \mathtt{F} - \mathtt{F} + \mathtt{F} + \mathtt{FF} - \mathtt{F} - \mathtt{FFF}$ in die Graphik



umgesetzt.

EIN BEISPIEL

Beispiel (Kochsche Schneeflocke)

Gegeben sei $G = (\Sigma, \omega, P)$ mit Alphabet $\Sigma = \{\mathtt{F}, +, -\}$, Axiom $\omega = \mathtt{F}$ und der Menge der Produktionen $\{\mathtt{F} \to \mathtt{F} + \mathtt{F} - -\mathtt{F} + \mathtt{F}\}$

EIN BEISPIEL

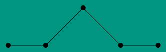
Beispiel (Kochsche Schneeflocke)

Gegeben sei $G = (\Sigma, \omega, P)$ mit Alphabet $\Sigma = \{\mathtt{F}, +, -\}$, Axiom $\omega = \mathtt{F}$ und der Menge der Produktionen $\{\mathtt{F} \rightarrow \mathtt{F} + \mathtt{F} - -\mathtt{F} + \mathtt{F}\}$
Wir legen $\delta = 45°$ fest. Für die Anzahl der Schritte n ergibt sich:

EIN BEISPIEL

Beispiel (Kochsche Schneeflocke)

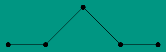
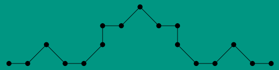
Gegeben sei $G = (\Sigma, \omega, P)$ mit Alphabet $\Sigma = \{\mathtt{F}, +, -\}$, Axiom $\omega = \mathtt{F}$ und der Menge der Produktionen $\{\mathtt{F} \to \mathtt{F}+\mathtt{F}--\mathtt{F}+\mathtt{F}\}$
Wir legen $\delta = 45°$ fest. Für die Anzahl der Schritte n ergibt sich:

$n = 1$	$\mathtt{F}+\mathtt{F}--\mathtt{F}+\mathtt{F}$	

EIN BEISPIEL

Beispiel (Kochsche Schneeflocke)

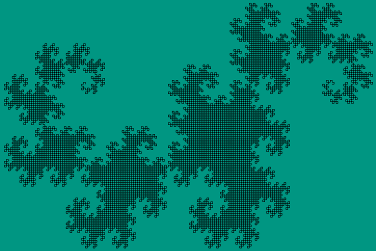
Gegeben sei $G = (\Sigma, \omega, P)$ mit Alphabet $\Sigma = \{\mathtt{F}, +, -\}$, Axiom $\omega = \mathtt{F}$ und der Menge der Produktionen $\{\mathtt{F} \to \mathtt{F} + \mathtt{F} - -\mathtt{F} + \mathtt{F}\}$
Wir legen $\delta = 45°$ fest. Für die Anzahl der Schritte n ergibt sich:

$n = 1$	$\mathtt{F} + \mathtt{F} - -\mathtt{F} + \mathtt{F}$	
$n = 2$	$\mathtt{F} + \mathtt{F} - -\mathtt{F} + \mathtt{F} + \mathtt{F} + \mathtt{F}$ $- - \mathtt{F} + \mathtt{F} - -\mathtt{F} + \mathtt{F} - -$ $\mathtt{F} + \mathtt{F} + \mathtt{F} + \mathtt{F} - -\mathtt{F} + \mathtt{F}$	

EIN ZWEITES BEISPIEL

Beispiel (Drachenkurve)

Sei $\delta = 90°$ und das L-System $G = (\{\mathtt{F_r}, \mathtt{F_l}, +, -\}, \mathtt{F_l}, \{\mathtt{F_l} \rightarrow \mathtt{F_l} + \mathtt{F_r}+, \mathtt{F_r} \rightarrow -\mathtt{F_l} - \mathtt{F_r}\})$, dann ergibt sich



Sowohl $\mathtt{F_l}$ als auch $\mathtt{F_r}$ werden als „Bewege den Stift einen Schritt der Länge d und zeichne eine Linie“ interpretiert.

WEITERFÜHRENDE LITERATUR

Przemyslaw Prusinkiewicz und Aristid Lindenmayer, The Algorithmic Beauty of Plants

unter

`http://algorithmicbotany.org/papers/#abop`

WEITERFÜHRENDE LITERATUR

Przemyslaw Prusinkiewicz und Aristid Lindenmayer, The Algorithmic Beauty of Plants

unter

`http://algorithmicbotany.org/papers/#abop`

Die folgenden Graphiken wurden diesem Buch entnommen: