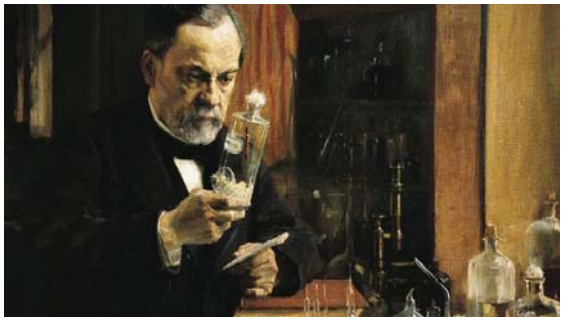


Die Theorie ist die Mutter der Praxis, denn ohne sie ist die Praxis nur Routine (...); es ist allein die Theorie, die den Geist der Erfindung entstehen und sich entwickeln lässt. — Louis Pasteur

Grundlagen der Theoretischen Informatik

Till Mossakowski

Fakultät für Informatik
Otto-von-Guericke Universität
Magdeburg

Wintersemester 2014/15

Organisatorisches

Vorlesung $3 \times 45 = 135$ Minuten, zwischendurch 15 Minuten Pause !

Anmeldung zur Übung ab 14.10. 7:00 Uhr

Erfolgreiche Teilnahme an den Übungen: Insgesamt für mindestens 66% der Aufgaben votieren, mindestens zweimal vortragen und die Belegaufgabe bestehen (mindestens 50% der erreichbaren Punkte erzielen).

Leistungsnachweis: Erfolgreich an den Übungen teilnehmen (siehe oben) und die Leistungsnachweisklausur nach dem aktuellen Semester oder deren Wiederholung nach dem darauffolgenden Semester bestehen (mindestens 50% der erreichbaren Punkte erzielen).

1

Einleitung

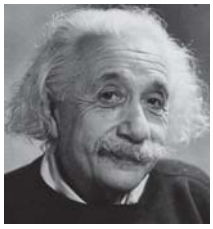
Theorie ?

logisch-systematische Zusammenfassung und Verallgemeinerung von Erkenntnissen über einen Bereich der Wirklichkeit, deren Zusammenhänge erklärend, begründend widergespiegelt werden

- Spezielle Relativitätstheorie
- Newtonsche Theorie (klassische Mechanik)
- Quantentheorie
- Keynesche Theorie
- Neoklassische Theorie
- Wissenschaftstheorie

Der Unterschied zwischen Theorie und Praxis ist in der Praxis weit höher als in der Theorie.

Ernst Ferstl



Es gibt nichts Praktischeres als eine gute Theorie!

Die *theoretische Informatik* beschüftigt sich mit der Abstraktion, Modellbildung und grundlegenden Fragestellungen, die mit der Struktur, Verarbeitung, Übertragung und Wiedergabe von Informationen in Zusammenhang stehen.

Wikipedia

Historisches

Beginn des 20. Jahrhunderts:
Ziel: Axiomatisierung der Mathematik

Diese Überzeugung von der Löslichkeit eines jeden mathematischen Problems ist uns ein kräftiger Ansporn während der Arbeit; wir hören in uns den steten Zuruf: Da ist das Problem, suche die Lösung. Du kannst sie durch reines Denken finden; denn in der Mathematik gibt es kein Ignorabimus!

David Hilbert, 1900

Emil du Bois-Reymond 1872: ignoramus et ignorabimus

2. Hilbertsches Problem

2. Die Widerspruchslosigkeit der arithmetischen Axiome.
Wenn es sich darum handelt, die Grundlagen einer Wissenschaft zu untersuchen, so hat man ein System von Axiomen aufzustellen, welche eine genaue und vollständige Beschreibung derjenigen Beziehungen enthalten, die zwischen den elementaren Begriffen jener Wissenschaft stattfinden. Die aufgestellten Axiome sind zugleich die Definitionen jener elementaren Begriffe und jede Aussage innerhalb des Bereiches der Wissenschaft, deren Grundlagen wir prüfen, gilt uns nur dann als richtig, falls sie sich mittelst einer endlichen Anzahl logischer Schlüsse aus den aufgestellten Axiomen ableiten läßt. Bei näherer Betrachtung entsteht die Frage, ob etwa gewisse Aussagen einzelner Axiome sich untereinander bedingen und ob nicht somit die Axiome noch gemeinsame Bestandteile enthalten, die man beseitigen muß, wenn man zu einem System von Axiomen gelangen will, die völlig von einander unabhängig sind.

Vor Allem aber möchte ich unter den zahlreichen Fragen, welche hinsichtlich der Axiome gestellt werden können, dies als das wichtigste Problem bezeichnen, zu beweisen, daß dieselben untereinander widerspruchslos sind, d.h. daß man auf Grund derselben mittelst einer endlichen Anzahl von logischen Schlüssen niemals zu Resultaten gelangen kann, die miteinander in Widerspruch stehen.

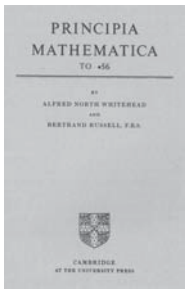
. . .

David Hilbert, 1900

Axiomatisierung

Whitehead, A. N., and B. Russell.
Principia Mathematica,
3 vols, Cambridge University Press,
1910, 1912, and 1913.

Second edition, 1925 (Vol. 1), 1927 (Vols 2, 3).
Abridged as Principia Mathematica to *56,
Cambridge University Press, 1962.



Russel-Zermelo Paradoxon (Russelsche Antinomie, 1903):

„Menge“ aller Mengen, die sich nicht selbst enthalten

$$\begin{aligned} R &= \{M \mid M \notin M\} \\ R \notin R &\Rightarrow R \in R \\ R \in R &\Rightarrow R \notin R \end{aligned}$$

→ Zermelo-Fraenkel-Mengenlehre

xkcd.com

Gödelscher Unvollständigkeitssatz

Satz: [Kurt Gödel, 1931]
Jedes Beweissystem für die Menge der wahren arithmetischen Formeln ist notwendigerweise unvollständig.

Es bleiben also immer wahre arithmetische Formeln übrig,
die nicht beweisbar sind.

10. Hilbertsches Problem

10. Entscheidung der Lösbarkeit einer diophantischen Gleichung.
Eine diophantische Gleichung mit irgendwelchen Unbekannten und mit ganzen rationalen Zahlencoefficienten sei vorgelegt: man soll ein Verfahren angeben, nach welchem sich mittelst einer endlichen Anzahl von Operationen entscheiden lässt, ob die Gleichung in ganzen rationalen Zahlen lösbar ist.

David Hilbert, 1900

Beispiel: $x^{10} + y^{10} = z^{10}$

David Hilbert, 1889

Entscheidungsprobleme

Entscheidungsproblem:

$$\mathcal{P} : \mathcal{I} \to \{\text{wahr, falsch}\}$$

Das Entscheidungsproblem:
Gibt es einen Algorithmus, der für jede beliebige Formel der Prädikatenlogik erster Stufe entscheidet, ob die Formel allgemeingültig ist?

D. Hilbert, W. Ackermann, 1928

Formalisierung des Berechenbarkeitsbegriffs

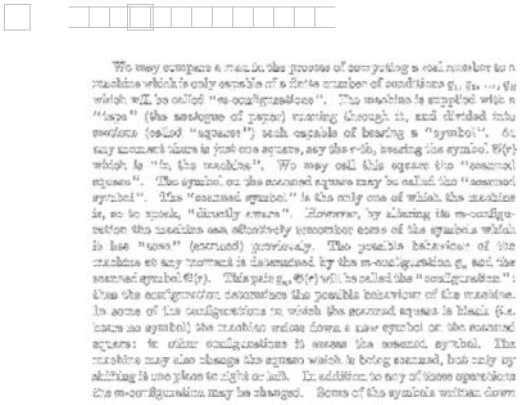
1920er, 1930er: "Effective Computability"

Berechnungsmodelle:

- λ-Kalkül
- Allgemeine rekursive Funktionen
- Turing-Maschinen
- μ-rekursive Funktionen
- Postsche Systeme

Church, Gödel, Herbrand, Kleene, Post, Rosser, Turing, ...

Turings Maschine



We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions $q_1, q_2, \ldots, q_R$ which will be called "m-configurations". The machine is supplied with a "tape" (the analogue of paper) running through it, and divided into sections (called "squares") each capable of bearing a "symbol". At any moment there is just one square, say the r-th, bearing the symbol $\mathfrak{S}(r)$ which is "in the machine". We may call this square the "scanned square". The symbol on the scanned square may be called the "scanned symbol". The "scanned symbol" is the only one of which the machine is, so to speak, "directly aware". However, by altering its m-configuration the machine can effectively remember some of the symbols which it has "seen" (scanned) previously. The possible behaviour of the machine at any moment is determined by the m-configuration q_n and the scanned symbol $\mathfrak{S}(r)$. This pair $q_n, \mathfrak{S}(r)$ will be called the "configuration": thus the configuration determines the possible behaviour of the machine. In some of the configurations in which the scanned square is blank (i.e. contains no symbol) the machine writes down a new symbol on the scanned square: in other configurations it erases the scanned symbol. The machine may also change the square which is being scanned, but only by shifting it one place to right or left. In addition to any of these operations the m-configuration may be changed. Some of the symbols written down

Churchsche These

Churchsche These [1936]:
Die Klasse der im intuitiven Sinne berechenbaren Funktionen stimmt genau mit der Klasse der Turing-berechenbaren Funktionen überein.

1940er, 1950er, 1960er, ...

- Computer (Z3, ENIAC, ...) und Programmiersprachen
- Markov-Algorithmen
- Registermaschinen
- WHILE-Programme

- Supercomputer

- Quantencomputer

... die Churchsche These hält stand ...

Unentscheidbarkeit

Satz: [Church, 1936], [Turing, 1937]
Die Prädikatenlogik erster Stufe ist unentscheidbar.

Satz: [Yuri Matiyasevich, 1970]
Das 10. Hilbertsche Problem ist unlösbar.

Vorarbeiten von Julia Robinson, Martin Davis, Hilary Putnam, ...

Halteproblem

Halteproblem

Gegeben eine Turing-Maschine (Beschreibung eines Programms) und eine endliche Eingabe für die Turing-Maschine, entscheide, ob die Turing-Maschine (das Programm) bei dieser Eingabe anhalten oder unendlich lange rechnen wird.


```
public static void main (String[] args) {
  int n = Integer.valueOf(args[0]).intValue();
  int total = 3;
  while (true) {
    for ( int x = 1; x <= total-2; ++x) {
      for ( int y = 1; y <= total-x-1; ++y) {
        int z = total - x - y;
        if ( exp(x,n) + exp(y,n) == exp(z,n) ) {
          return;
        }
      }
    }
    ++total;
  }
}
```

$$x^n + y^n = z^n$$

Satz: *Das Halteproblem ist nicht entscheidbar.*

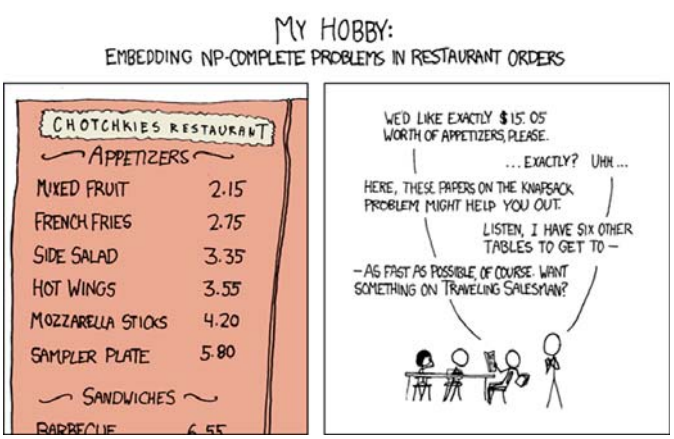
Satz von Cook

entscheidbar $\not\Rightarrow$ handhabbar

1970er: NP-Vollständigkeit

Wenn jemand einen Polynomialzeitalgorithmus für ein NP-vollständiges Problem findet, so hat er damit auch Polynomialzeitalgorithmen für alle anderen Probleme in NP gefunden.

Satz: [Cook, 1971], [Levin, 1973]
Das Erfüllbarkeitsproblem der Aussagenlogik ist NP*-vollständig.*



xkcd.com

Grundlagen

Notation:

Wir schreiben

$\neg$	für	*nicht*
$\vee$	für	*oder*
$\wedge$	für	*und*
$\Rightarrow$	für	*wenn . . . dann*
$\Leftrightarrow$	für	*genau dann wenn*
$\exists$	für	*es existiert*
$\forall$	für	*für alle*

Mengenlehre

Eine *Menge* ist eine Zusammenfassung von *verschiedenen* Objekten zu einer Gesamtheit. Die Objekte in einer Menge werden *Elemente* der Menge genannt. Die Elemente sind ungeordnet, aber wohlunterschieden. Falls x ein Element einer Menge M ist, schreiben wir $x \in M$, sonst $x \notin M$.

$\emptyset$	leere Menge
$\mathbb{N}$	Menge der natürlichen Zahlen
$\mathbb{Z}$	Menge der ganzen Zahlen
$\mathbb{Q}$	Menge der rationalen Zahlen
$\mathbb{R}$	Menge der reellen Zahlen
$\mathbb{N}_0$	Menge der natürlichen Zahlen und Null

Darstellung von Mengen:

Aufzählung der Elemente:
$\{3,5,8,9,17\}$,
$\{a,b,c,d,e,f,g,h\}$,
$\{\text{grün}, \bullet, 61, \heartsuit, \{g,h\}\}$,
$\{\emptyset,\{\emptyset\},\{\emptyset,\{\emptyset\}\}\}$,
$\{0,1,4,9,16,25,\ldots\}$

Auswahl durch Eigenschaft:
$\{x \in M \mid$ und x hat Eigenschaft $E\}$
$\{x \in \mathbb{N} \mid$ und x ist ein Vielfaches von $3\}$,
$\{x \in \mathbb{Z} \mid$ und x ist gerade$\}$,
$\{x \in \mathbb{R} \mid$ und x ist algebraisch$\}$

Eine Menge A heißt *Teilmenge* einer Menge B, wir schreiben $A \subseteq B$, falls jedes Element von A auch ein Element von B ist.

Eine Menge A heißt *Obermenge* einer Menge B, wir schreiben $A \supseteq B$, genau dann wenn $B \subseteq A$.

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

Eine Menge A heißt *echte Teilmenge* einer Menge B, wir schreiben $A \subset B$, genau dann wenn $A \subseteq B$ und $A \neq B$.

$A \cup B = \{x \mid x \in A \text{ oder } x \in B\}$ heißt *Vereinigung* von A und B

$A \cap B = \{x \mid x \in A \text{ und } x \in B\}$ heißt *Durchschnitt* von A und B

$A - B = \{x \mid x \in A \text{ und } x \notin B\}$ heißt *Differenz* von A und B

$2^A = \{M \mid M \subseteq A\}$, die Menge aller Teilmengen von A,
heißt *Potenzmenge* von A.

Satz: *Seien A, B und C Mengen. Dann gilt*

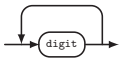
$$\begin{aligned}
A \cup A &= A \\
A \cap A &= A \\
A \cup B &= B \cup A \\
A \cap B &= B \cap A \\
(A \cup B) \cup C &= A \cup (B \cup C) \\
(A \cap B) \cap C &= A \cap (B \cap C) \\
(A \cup B) \cap C &= (A \cap C) \cup (B \cap C) \\
(A \cap B) \cup C &= (A \cup C) \cap (B \cup C) \\
(A \cup B) \cap A &= A \\
(A \cap B) \cup A &= A \\
A - (B \cup C) &= (A - B) \cap (A - C) \\
A - (B \cap C) &= (A - B) \cup (A - C)
\end{aligned}$$

Formalisierung von Sprachen und Grammatiken

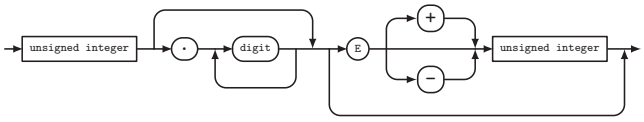
$$\begin{aligned}
\langle \textit{Satz} \rangle &\rightarrow \langle \textit{Subjekt} \rangle \langle \textit{Prädikat} \rangle \langle \textit{Objekt} \rangle \\
\langle \textit{Subjekt} \rangle &\rightarrow \langle \textit{Artikel} \rangle \langle \textit{Attribut} \rangle \langle \textit{Substantiv} \rangle \\
\langle \textit{Artikel} \rangle &\rightarrow \text{der} \\
\langle \textit{Artikel} \rangle &\rightarrow \text{die} \\
\langle \textit{Artikel} \rangle &\rightarrow \ldots \\
\ldots &\rightarrow \ldots
\end{aligned}$$

Blockdiagramme aus Beschreibung der PASCAL-Syntax

`unsigned integer`



`unsigned real`



Ausschnitt aus Beschreibung der C++-Syntax

```
class-name:
        identifier
        template-id
template-id:
        template-name < template-argument-list >
template-name:
        identifier
template-argument-list:
        template-argument
        template-argument-list , template-argument
identifier:
        nondigit
        identifier nondigit
        identifier digit
```


Wörter und Sprachen

Im Zusammenhang mit formalen Sprachen wird jede nicht-leere endliche Menge als *Alphabet* bezeichnet. Die Elemente eines Alphabets werden *Symbole* genannt.

Ein *Wort* über einem Alphabet Σ ist eine endliche Folge von Symbolen aus Σ.

Das *leere Wort* ist die aus 0 Symbolen bestehende Folge und wird mit ε bezeichnet.

Die Menge aller Wörter über einem Alphabet Σ wird mit Σ^* bezeichnet. Ferner ist $\Sigma^+ = \Sigma^* - \{\varepsilon\}$.

Eine *Sprache* über einem Alphabet Σ ist eine Teilmenge von Σ^*.

Beispiele:

$\Sigma = \{a, b, c\}$

$$\begin{aligned}
c &\in \Sigma^* \\
abba &\in \Sigma^* \\
cbc &\in \Sigma^* \\
abcca &\in \Sigma^* \\
\varepsilon &\in \Sigma^*
\end{aligned}$$

$\{w \in \Sigma^* \mid w \text{ beginnt mit } a\}$
$\{w \in \Sigma^* \mid w \text{ enthält kein } c\}$
$\{w \in \Sigma^* \mid w \text{ enthält kein } a \text{ oder kein } b\}$
$\{w \in \Sigma^* \mid \text{die Anzahl der } a \text{ in } w \text{ ist eine Primzahl}\}$
$\emptyset \subseteq \Sigma^*$

Operationen auf Wörtern

Die *Länge* eines Wortes w ist die Länge der Folge der Symbole. Die Länge des Wortes w bezeichnen wir mit $|w|$.

Seien x und y Wörter über dem gleichen Alphabet. Dann bezeichnet $x \circ y$ das Wort, das durch Hintereinanderschreiben von x und y entsteht. Diese Operation wird als *Konkatenation* oder auch als Verkettung bezeichnet. Statt $x \circ y$ schreiben wir meist einfach xy.

Beispiel: $\textit{kupfer} \circ \textit{kessel} = \textit{kupferkessel}$

Für alle Wörter u, v, w gilt

$$\begin{aligned}
w \circ \varepsilon = \varepsilon \circ w &= w \\
(u \circ v) \circ w &= u \circ (v \circ w)
\end{aligned}$$

v ist *Teilwort* eines Wortes w, wenn es Wörter x und y gibt, so dass $w = xvy$.

v ist *Präfix* eines Wortes w, wenn es ein Wort y gibt, so dass $w = vy$.

v ist *Suffix* eines Wortes w, wenn es ein Wort x gibt, so dass $w = xv$.

Für ein Wort w und eine natürliche Zahl n bezeichnet w^n die $(n-1)$-fache Konkatenation von w:

$$w^1 = w$$
$$w^n = w^{n-1}w$$

und insbesondere

$$w^0 = \varepsilon$$

Beispiel: $(da)^3 = dadada$

w^n wird auch als n-te Potenz von w bezeichnet.

Für ein Wort w bezeichnet w^R das Wort w in umgekehrter Symbolfolge:

$$w = \sigma_1\sigma_2\cdots\sigma_k \qquad w^R = \sigma_k\sigma_{k-1}\cdots\sigma_1$$

Beispiel: $(magdeburg)^R = grubedgam$

w^R wird auch als Spiegelung von w bezeichnet.
R steht hier für die englische Bezeichnung *Reversal*.

Operationen auf Sprachen

Wie auf allen Mengen sind auch auf Sprachen die Operationen $\cup, \cap, -$ (Vereinigung, Schnitt, Differenz) definiert.

Das *Komplement* einer Sprache L über Σ enthält genau die Wörter aus Σ^*, die nicht zu L gehören. Wir bezeichnen das Komplement von L mit $\overline{L}$.

Seien L_1 und L_2 Sprachen über Σ. Dann ist $L_1 \circ L_2 = L_1L_2 = \{w \in \Sigma^* \mid$ es gibt $x \in L_1$ und $y \in L_2$ so dass $w = xy\}$ die *Konkatenation* von L_1 und L_2. Mit $L^n = \{w \in \Sigma^* \mid$ es gibt $w_1, w_2, \ldots, w_n \in L$ so dass $w = w_1w_2\cdots w_n\}$ bezeichnen wir die n-Potenz der Sprache L. Insbesondere ist $L^0 = \{\varepsilon\}$.

Der *Kleene Star* Abschluss einer Sprache L ist

$$L^* = \bigcup_{i \geq 0} L^i$$

Ferner definieren wir

$$L^+ = \bigcup_{i \geq 1} L^i$$

Schließlich definieren wir die Spiegelung einer Sprache L:

$$L^R = \{w^R \mid w \in L\}$$

Grammatik

Eine Grammatik generiert Wörter durch einen Termersetzungsprozess.

Definition:
Eine Grammatik ist ein 4-Tupel (V, Σ, R, S), wobei gilt:

- *$V \neq \emptyset$ ist eine endliche Menge, die Menge der Nichtterminal(symbol)e oder Variablen,*
- *Σ ist ein Alphabet, das Terminalalphabet, $\Sigma \cap V = \emptyset$,*
- *$R \subset (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$ ist eine endliche Menge von Regeln oder auch Produktionen,*
- *$S \in V$ ist das Startsymbol.*

Falls $G = (V, \Sigma, R, S)$ eine Grammatik ist und $(u, v) \in R$, so schreiben wir $u \to_G v$.

Beispiel: $G = (\{S\}, \{a, b\}, \{S \to \varepsilon,\ \ S \to SS,\ \ S \to aSb\}, S)$

	S	Regel
$\Rightarrow_G$	SS	$S \to_G SS$
$\Rightarrow_G$	$aSbS$	$S \to_G aSb$
$\Rightarrow_G$	$aSSbS$	$S \to_G SS$
$\Rightarrow_G$	$aSSbaSb$	$S \to_G aSb$
$\Rightarrow_G$	$aaSbSbaSb$	$S \to_G aSb$
$\Rightarrow_G$	$aaSbSbab$	$S \to_G \varepsilon$
$\Rightarrow_G$	$aabSbab$	$S \to_G \varepsilon$
$\Rightarrow_G$	$aabaSbbab$	$S \to_G aSb$
$\Rightarrow_G$	$aababbab$	$S \to_G \varepsilon$

Zu einer gegebenen Grammatik $G = (V, \Sigma, R, S)$ definieren wir eine Relation $\Rightarrow_G$ auf $(V \cup \Sigma)^* \times (V \cup \Sigma)^*$ wie folgt: Seien $x, y \in (V \cup \Sigma)^*$.

$x \Rightarrow_G y$ genau dann wenn es $u, v, w, z \in (V \cup \Sigma)^*$ gibt, so dass

$$x = uvw, \quad y = uzw, \quad \text{und} \quad v \to_G z$$

Wir sagen, *x geht unter G unmittelbar in y über*, falls $x \Rightarrow_G y$.

Die *von* einer Grammatik G *erzeugte Sprache* ist

$$L(G) = \{w \in \Sigma^* \mid \exists w_1, \ldots, w_k : S \Rightarrow_G w_1 \Rightarrow_G w_2 \Rightarrow_G \cdots \Rightarrow_G w_k = w\}$$

Beispiel:

$G = (\{S\}, \{a,b\}, \{S \to \varepsilon,\ \ S \to SS,\ \ S \to aSb\}, S)$

$$\begin{aligned} ab &\in L(G) \\ ba &\notin L(G) \\ abab &\in L(G) \\ aabbab &\in L(G) \\ aabbba &\notin L(G) \end{aligned}$$

$L(G)$: korrekt verschachtelte „Klammern“

Beispiel:

$G = (\{S,A,B,C\}, \{a,b,c\}, R, S)$ mit

$$\begin{aligned} R \ = \ \{&S \to ABCS,\ \ S \to \varepsilon, \\ &CA \to AC,\ \ AC \to CA, \\ &BA \to AB,\ \ AB \to BA, \\ &CB \to BC,\ \ BC \to CB, \\ &A \to a,\ B \to b, C \to c\} \end{aligned}$$

$L(G) = \{w \in \{a,b,c\}^* \mid w$ enthält gleichviele a,b und $c\}$

	S	Regel
$\Rightarrow_G$	$ABCS$	$S \to ABCS$
$\Rightarrow_G$	$ABCABCS$	$S \to ABCS$
$\Rightarrow_G$	$ABCABC$	$S \to \varepsilon$
$\Rightarrow_G$	$ABCBAC$	$AB \to BA$
$\Rightarrow_G$	$ABCBCA$	$AC \to CA$
$\Rightarrow_G$	$ABCCBA$	$BC \to CB$
$\Rightarrow_G$	$aBCCBA$	$A \to a$
$\Rightarrow_G$	$abCCBA$	$B \to b$
	$\vdots$	
$\Rightarrow_G$	$abccba$	

Chomsky-Hierarchie

Klassifikation von Grammatiken (und daraus resultierend Klassifikation von formalen Sprachen)

Typ 0 *allgemeine Grammatiken*

Typ 1 *kontextsensitive Grammatiken*

Typ 2 *kontextfreie Grammatiken*

Typ 3 *reguläre Grammatiken*

Anwendungen

Formale Sprachen und Automatentheorie

- Textmustererkennung (z.B. *grep*)
- eingebettete Systeme
- Programmiersprachendesign und Übersetzerbau
- Computergrafik: Pflanzengenerierung

Komplexitätstheorie

- Kryptographie

Autor: Bernd Lintermann

Auswirkungen

Berechenbarkeitstheorie

- Programmverifikation

Komplexitätstheorie

- Heuristiken und Approximationsalgorithmen

Überblick

1. Einleitung
2. Automatentheorie und Formale Sprachen
3. Berechenbarkeitstheorie
4. Komplexitätstheorie

Literaturhinweise

J. Hopcroft, R. Motwani, J. Ullmann.
Introduction to Automata Theory, Languages, and Computation (3rd Edition.).
Pearson – Addison-Wesley.

H. Lewis, C. Papadimitriou.
Elements of the Theory of Computation (2nd Edition).
Pearson – Prentice Hall.

M. Sipser.
Introduction to the Theory of Computation (2nd Edition).
Cengage Learning – Thomson.

J. Hopcroft, R. Motwani, J. Ullmann.
Einführung in der Automatentheorie, Formale Sprachen und Komplexitätstheorie (2., überarbeitete Auflage).
Pearson Studium.

U. Schöning.
Theoretische Informatik - kurzgefasst.
Spektrum.

Populärwissenschaftliches zum Thema:

D. Hofstadter.
Gödel, Escher, Bach: Ein Endloses Geflochtenes Band.
DTV.

D. Harel.
Das Affenpuzzle und weitere bad news aus der Computerwelt.
Springer Verlag.