

Theoretische Informatik: Berechenbarkeit und Komplexität

Prof. Dr. Sibylle Schwarz
HTWK Leipzig, Fakultät IM
Gustav-Freytag-Str. 42a, 04277 Leipzig
Zimmer ZU 411 (Zuse-Bau)
<https://informatik.htwk-leipzig.de/schwarz>
sibylle.schwarz@htwk-leipzig.de

Wintersemester 2026/27

Informatik

Informatik Lehre von der **symbolischen Darstellung** und **Verarbeitung** von **Information** durch **Algorithmen**

Teilgebiete der Informatik:

theoretisch

- ▶ Sprachen zur Formulierung von Information und Algorithmen,
- ▶ Möglichkeiten und Grenzen der Berechenbarkeit durch Algorithmen,
- ▶ Grundlagen für technische und praktische (und angewandte) Informatik

technisch

- ▶ maschinelle Darstellung von Information
- ▶ Mittel zur Ausführung von Algorithmen

(Rechnerarchitektur, Hardware-Entwurf, Netzwerk, ...)

praktisch Entwurf und Implementierung von Algorithmen
(Betriebssysteme, Compilerbau, SE, ...)

angewandt Anwendung von Algorithmen
(Text- und Bildverarbeitung, Datenbanken, KI, Medizin-, Bio-, Wirtschafts-, Medien-Informatik, ...)

Teilgebiete der theoretischen Informatik

Formale Sprachen

(bekannt aus INB-Modul)

- ▶ Repräsentation von Informationen und Aufgaben in maschinenlesbarer Form (Mensch-Maschine-Kommunikation)
- ▶ Ausdrucksstärke und Flexibilität von Programmiersprachen
- ▶ Übersetzung von Programmiersprachen (z.B. in ausführbaren Code)

Maschinenmodelle (Automaten)

(bekannt aus INB-Modul)

- ▶ Möglichkeiten und Grenzen verschiedener Modelle zur (maschinellen) Ausführung von Algorithmen

Berechenbarkeitstheorie

- ▶ Welche Aufgaben sind überhaupt algorithmisch (mit Hilfe verschiedener Maschinenmodelle) lösbar?
Auch negative Antworten sind sehr hilfreich
(sparen Aufwand für ungeeignete Lösungsansätze)

Komplexitätstheorie

- ▶ Welche Aufgaben sind mit beschränkten Ressourcen (z.B. Zeit, Speicherplatz) lösbar?
- ▶ Für welche Aufgaben können schnelle Algorithmen existieren?

Prinzipien der theoretischen Informatik

ältester Zweig der Informatik (lange vor dem ersten Computer)

Mathematische Prinzipien:

- ▶ Abstraktion
 - ▶ ermöglicht verallgemeinerte Aussagen und breit einsetzbare Verfahren,
 - ▶ Ergebnisse und Verfahren oft nicht sofort praktisch anwendbar, müssen auf spezielle Situationen angepasst werden.
- ▶ Beweisbarkeit
 - ▶ erfordert präzise Modellierung der Aufgabe
 - ▶ Nachweis der Korrektheit von Hard- und Software (Tests können dies nicht !)

Wissen aus der theoretischen Informatik

- ▶ veraltet über viele Jahre kaum
- ▶ Grundlage für Verständnis von (schnelllebigem) Spezialwissen, z.B. konkrete Programmiersprachen, Domain-spezifische Sprachen, Transformationen verschiedener Darstellungen

Aus der Modulbeschreibung

C144 Theoretische Informatik: Berechenbarkeit und Komplexität

Arbeitsaufwand: Präsenzzeit 56 SWS (= 2 SWS V + 2 SWS S)
Vor- und Nachbereitungszeit 94 h (≈ 6 h je Woche)

Voraussetzungen: anwendungsbereite Kenntnisse auf den Gebieten
Modellierung, Logik, Formale Sprachen,
Maschinenmodelle, Algorithmen und Datenstrukturen,
Aufwandsabschätzungen

Lernziele: Nach erfolgreichem Abschluss des Moduls sind die Studierenden in der Lage, fundiert die prinzipiellen Möglichkeiten und Grenzen der verschiedenen Berechenbarkeitsmodelle einzuschätzen.
Sie besitzen ein Grundverständnis grundlegender Komplexitätsklassen.
Sie können die Komplexität ausgewählter Problembeispiele beurteilen und algorithmisch unlösbar oder schwer handhabbare Probleme als solche erkennen.

Inhalt der Lehrveranstaltung

- ▶ **Berechenbarkeit** von Funktionen
- ▶ Algorithmenmodelle und ihre Rolle bei der Untersuchung von Grenzen der Berechenbarkeit
- ▶ Codierung von Aufgaben (Problemen) als Mengen
- ▶ **Entscheidbarkeit** von Mengen
- ▶ Unentscheidbare Probleme
- ▶ Komplexitätsmaße
- ▶ Zusammenhang zwischen Ausdrucksstärke von Algorithmenmodellen und **Komplexität** von Problemen
- ▶ Komplexitätsklassen (P, NP, PSPACE, ...)

jeweils mit vielen Beispielen und praktischen Folgerungen

Lehrveranstaltungen

Folien, Übungsserien, aktuelle Informationen unter

<https://informatik.htwk-leipzig.de/schwarz/lehre/ws26/tim>

Vorlesung normalerweise jede Woche (2 SWS)

Selbststudium (Hausaufgaben): (6 h / Woche)

schriftliche Übungsserien (zu jeder Vorlesung)

Besprechung im folgenden Seminar

Autotool (ca. eine Woche Bearbeitungszeit)

Seminar (2 SWS)

Besprechung der Übungsserien

Präsentation der Lösungen (Vorrechnen)

Alle Folien, Aufgaben, Lösungen schnell auffindbar
mitbringen !

Prüfung

Prüfungsvorleistungen:

- ▶ $\geq 50\%$ aller Punkte für Autotool-Pflichtaufgaben und
- ▶ ≥ 3 Punkte für Vorrechnen im Seminar

Prüfung: Klausur 90 min

Aufgabentypen ähnlich Übungsaufgaben

(Hilfsmittel: beidseitig handbeschriebenes A4-Blatt)

Entscheidbarkeit: Eigenschaften von Grammatiken

- ▶ E_3 : Sprach-Äquivalenz von Typ-3-Grammatiken
- ▶ E_2 : Sprach-Äquivalenz von Typ-2-Grammatiken

gesucht ist jeweils ein Algorithmus mit

Eingabe: Paar (G_1, G_2) von Grammatiken,

Ausgabe: 1, falls $L(G_1) = L(G_2)$, sonst 0

praktische Motivation: Test bzw. Verkleinerung von regulären Ausdrücken, von Grammatiken (automatische Bewertung von Übungsaufgaben zu formalen Sprachen)

- ▶ E_3 ist entscheidbar,
bekannt aus INB-Modul Automaten und formale Sprachen
- ▶ E_2 nicht entscheidbar (Nachweis später im Semester hier)

Methode:

Reduktion: wenn E_2 entscheidbar, dann auch ...

Sind verschiedene Aufgaben gleich schwer?

(eine typische Frage der Komplexitätstheorie)

- ▶ Eine (zulässige) k -Knoten-Färbung eines Graphen $G = (V, E)$ ist eine Funktion $f : V \rightarrow \{1, 2, \dots, k\}$ mit $\forall uv \in E : f(u) \neq f(v)$.
- ▶ $k\text{COL}$ = die Menge der Graphen, die eine k -Knoten-Färbung besitzen.
- ▶ Aufgaben: Gilt für einen gegebenen Graphen G
 - ▶ $G \in 2\text{COL}$?
 - ▶ $G \in 3\text{COL}$?
- ▶ beide Aufgaben sind entscheidbar (warum?)
- ▶ für 2COL ist ein effizienter Algorithmus bekannt, für 3COL nicht.

Methode:

Reduktion: wenn man 3COL effizient lösen könnte, dann auch ...

Praktische Problemstellungen

Berechenbarkeitsmodell = Programmierparadigma

- ▶ Registermaschine: imperative Programmierung
- ▶ Loop- und While-Programme:
strukturierte (imperative) Programmierung
- ▶ primitiv/allgemein-rekursive Funktionen:
funktionale Programmierung
- ▶ (uniforme) Schaltkreise: parallele Programmierung
- ▶ nichtdeterministische Maschinen: Suchverfahren

für jede dieser Definitionen:

- ▶ exakte Beschreibung (Spezifikation)
von (abstrakter) Syntax und Semantik (Interpreter)

zwischen diesen Definitionen:

- ▶ semantik-erhaltende Übersetzung (Compiler)

Geschichte des Algorithmenbegriffs

Suche nach **Lösungsverfahren** für mathematische Aufgaben
(symbolische Differentiation, Integration, Gleichungssysteme)

Beispiele:

1. Allgemeingültigkeit gegebener prädikatenlogischer Formeln
2. das 10. Hilbertsche Problem (1900):
Lösbarkeit gegebener Polynomgleichungen in ganzen Zahlen
z.B. $a^2 + 1 = 0$, $a^2 + b = 0$, $a^2 + b^2 - 1 = 0$, $a^2 - b^2 = 0$,
 $a^2 + b^2 - c^2 = 0$, $a^5 = 0$, $a^5 + b^5 - c^5 = 0$

bzw. nach **Beweisen für deren Nicht-Existenz**

1. Gödel, Church, Turing (1936,...): nicht entscheidbar
2. Matiassevich (1970): nicht entscheidbar

Bedeutung des Algorithmenbegriffs

(nach K. Wagner: Theoretische Informatik, Springer 2003)

- ▶ Die Bedeutung des Algorithmenbegriffs für Mathematik und Informatik entspricht der Bedeutung des Begriffes der natürlichen Zahlen.
- ▶ Die mathematische Präzisierung des Algorithmenbegriffs und die Erkenntnis der Grenzen des algorithmisch Machbaren gehören zu den wichtigsten intellektuellen Leistungen des 20. Jahrhunderts.

Literatur

- ▶ Juraj Hromkovic: *Theoretische Informatik – Formale Sprachen, Berechenbarkeit, Komplexitätstheorie, Algorithmik, Kommunikation und Kryptographie*, Vieweg 2014
- ▶ Ingo Wegener: *Theoretische Informatik* Teubner 2005
- ▶ Klaus Wagner: *Theoretische Informatik* Springer 2003
- ▶ Gottfried Vossen, Kurt-Ulrich Witt: *Grundkurs Theoretische Informatik* Vieweg 2016
- ▶ Uwe Schöning: *Theoretische Informatik – kurzgefasst* Spektrum 2001
- ▶ Renate Winter: *Theoretische Informatik* Oldenbourg 2002
- ▶ John E. Hopcroft, Jeffrey D. Ullman *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie* Addison-Wesley 1990
- ▶ Hartley Rogers Jr.: *Theory of Recursive Functions and Effective Computability*, 1987
- ▶ Michael Garey, David S. Johnson: *Computers and Intractability* Freeman 1979

Wiederholung Grundlagen

aus den INB-Modulen

- ▶ Theoretische Informatik: Automaten und formale Sprachen

`http:`

`//informatik.htwk-leipzig.de/schwarz/lehre/ss26/tib`

- ▶ Modellierung

`http://informatik.htwk-leipzig.de/schwarz/lehre/ws25/
modellierung`

WH: Alphabet, Wort, Sprache

Operationen auf

Wörtern: $\circ, *, ^R$

Sprachen: $\cup, \cap, \circ, *, ^R$,

Bsp.:

► z.z. $\forall u \in A^* : (u^R)^R = u$

► z.z. $\forall u, v \in A^* : (u \circ v)^R = v^R \circ u^R$

ÜA: Zeigen:

1. $\forall u, v \in A^* : (u^R \circ v^R) = (v \circ u)^R$

2. $\forall L \subseteq A^* : L \circ \{\varepsilon\} = L$

3. $\forall L \subseteq A^* : L \circ \emptyset = \emptyset$

4. $\forall L, L' \subseteq A^* : L \circ (L' \circ L)^* = (L \circ L')^* \circ L$

Repräsentation formaler Sprachen

- ▶ Reguläre Ausdrücke (algebraisch, Terme)
- ▶ Grammatiken, Ableitungen (konstruktiv)
- ▶ Maschinenmodelle mit verschiedenen Akzeptanzbedingungen
 - ▶ NFA, DFA
 - ▶ PDA
 - ▶ LBA
 - ▶ TM

WH: Reguläre Ausdrücke

Syntax: Terme (Bäume)

Induktive Definition der Menge **RegExp(A)** aller (einfachen) **regulären Ausdrücke** über Alphabet A :

IA: $A \subseteq \text{RegExp}(A)$, $\varepsilon \in \text{RegExp}(A)$, $\emptyset \in \text{RegExp}(A)$

IS: Für alle $E, F \in \text{RegExp}(A)$ gilt auch

- ▶ $E + F \in \text{RegExp}(A)$
- ▶ $EF \in \text{RegExp}(A)$
- ▶ $E^* \in \text{RegExp}(A)$

Semantik: $L(E) \subseteq A^*$, Äquivalenz

Erweiterte reguläre Ausdrücke: mit zusätzlich $\cap, \setminus, \overline{}, +, ^n, ^R$
ÜA:

1. Zeige $ab(bab)^* \equiv (abb)^*ab$
2. Zeige $\forall E, F \in \text{RegExp}(A) : E(FE)^* \equiv (EF)^*E$
3. Finde äquivalenten RegExp zu $\overline{(a + b)^*ab(a + b)^*}$
(mit Nachweis)

Achtung: Es gibt Sprachen, die **nicht** durch einen regulären Ausdruck dargestellt werden können. (Warum?)

WH: Mächtigkeit unendlicher Mengen – Abzählbarkeit

Eine Menge A heißt

abzählbar gdw. $\exists f : \mathbb{N} \rightarrow A$ surjektiv

überabzählbar sonst

Abschlusseigenschaften der Menge aller abzählbaren Mengen:

Sind die Mengen A_1 und A_2 abzählbar, dann sind auch

- ▶ jede Teilmenge von A_1 abzählbar
- ▶ $A_1 \cup A_2$ abzählbar
- ▶ $A_1 \times A_2$ abzählbar (nach Cantor 1)
- ▶ A_1^* abzählbar

Überabzählbar sind z.B. (nach Cantor 2):

- ▶ $2^{\mathbb{N}}$,
- ▶ $\mathbb{R}$,
- ▶ 2^{A^*} für $A \neq \emptyset$

Wortersetzung und Grammatiken

Syntax: Grammatik $G = (N, T, P, S)$ mit

- ▶ Alphabet $N \cup T$ (N enthält Hilfssymbole)
Startsymbol $S \in N$
- ▶ Wortersetzungssystem $P \subseteq (N \cup T)^+ \times (N \cup T)^*$

Semantik:

- ▶ Ableitung von Wörtern in Grammatiken
- ▶ $L(G) \subseteq T^*$
- ▶ Äquivalenz
- ▶ Palindrom-, Dyck-, Łukasiewicz-Sprachen

ÜA:

1. Ableitungen von $\varepsilon, b, ab, ba, aba$ in
 $G = (\{S, T\}, \{a, b\}, \{S \rightarrow T|aS a|bSa, T \rightarrow b|\varepsilon\}, S)$
2. Mengen-Darstellung von $L(G)$
3. Finde Grammatik G mit $L(G) = \{a^m b^n a^{m+n} \mid m, n \in \mathbb{N}\}$
4. Finde Grammatik G mit $L(G) = L(b(a^*b)^*)$

Chomsky-Hierarchie

Eine **Grammatik** $G = (N, T, P, S)$ ist vom Chomsky-Typ

0 immer,

1, falls für jede Regel $(l \rightarrow r) \in P$ gilt: $|l| \leq |r|$
(monoton, kontextsensitiv)

2, falls Typ 1 und für jede Regel $(l \rightarrow r) \in P$ gilt: $l \in N$
(kontextfrei)

3, falls Typ 2 und für jede Regel $(l \rightarrow r) \in P$ gilt:
 $l \in N$ und $r \in (T \cup (T \circ N))$
(regulär)

Eine **Sprache** $L \subseteq T^*$ heißt **vom (Chomsky-)Typ i** für $i \in \{0, \dots, 3\}$, falls i die größte Zahl ist, für die eine Grammatik G vom Typ i mit $L \setminus \{\varepsilon\} = L(G)$ existiert.

$\mathcal{L}_i$ bezeichnet die Menge aller Sprachen vom Typ i .

ÜA: Einordnung von $\{a^n b^m c^k \mid n, m, k \in \mathbb{N}\}$, $\{a^{2^n} \mid n \in \mathbb{N}\}$,
 $\{a^{n^2} \mid n \in \mathbb{N}\}$, $\{a^n b^m a^n b^m \mid n \in \mathbb{N}\}$, $\{a^n b^n \mid n, m \in \mathbb{N}\}$,
 $\{a^n b^n c^n \mid n \in \mathbb{N}\}$

Achtung: Es gibt Sprachen, die **nicht** durch eine Grammatik erzeugt werden können, also **keinen** Chomsky-Typ haben (Warum?)

Maschinenmodelle

Definition (endliche Beschreibung) durch

- ▶ interne Steuerung
(z.B. endliche Menge von Zuständen)
- ▶ externen Speicher mit speziellen Zugriffsmöglichkeiten
 - ▶ Typ des Speicherinhaltes
(z.B. endliches Wort über endlichem Alphabet)
 - ▶ Zugriffsmethode (z.B. Lesen / Schreiben, einmal / wiederholt, feste Reihenfolge)

Konfigurationen (Momentaufnahmen)

und endliche Menge zulässiger lokaler Übergänge zwischen Konfigurationen

schrittweise Berechnung : Folge von Konfigurationen von einer Startkonfiguration über zulässige Konfigurationsübergänge

Akzeptanz einer Eingabe durch akzeptierende Berechnung:
endliche Konfigurationenfolge zu einer akzeptierenden Konfiguration

WH: Endliche Automaten

Syntax: NFA] (nondeterministic finite automaton)

$A = (X, Q, \delta, I, F)$ mit

X endliches Alphabet,

Q endliche Menge von Zuständen,

δ Übergangsrelationen $\delta : X \rightarrow 2^{(Q \times Q)}$,

$I \subseteq Q$ Startzustände,

$F \subseteq Q$ akzeptierende Zustände.

- ▶ Eigenschaften: vollständig, deterministisch (DFA)
- ▶ Akzeptanz von Wörtern, akzeptierte Sprache
- ▶ Transformationen zu Sprachoperationen
- ▶ Transformation von / zu Typ-3-Grammatik, RegExp
- ▶ nicht NFA-akzeptierbare Sprachen

ÜA:

1. $L(A)$ für NFA $A = (\{p, q\}, \{a, b\}, \delta, \{p\}, \{q\})$ mit $\delta(a) = \{(p, p), (p, q), (q, p)\}$, $\delta(b) = \{(p, p), (q, q)\}$
2. Finde NFA B mit $L(B) = \overline{L(A)}$
3. Zeige: $\{a^n b^m \mid n < m\}$ nicht NFA-akzeptierbar.
4. Verfahren zum Nachweis von $L(A) = L(B)$ für NFA A, B

WH: Kellerautomat (PDA)

Erweiterung von NFA um Stack als internen Speicher
nichtdeterministischer **Kellerautomat** (PDA)

$A = (X, Q, \Gamma, \delta, q_0, F, \perp)$ mit

X Alphabet (endlich, nichtleer)

Q Menge von Zuständen (endlich, nichtleer)

Γ Kelleralphabet (endlich, nichtleer)

$q_0 \in Q$ Startzustand

$F \subseteq Q$ Menge der akzeptierenden Zustände

$\delta : (X \cup \{\varepsilon\}) \rightarrow (Q \times Q \times \Gamma \times \Gamma^*)$
für jedes $s \in X \cup \{\varepsilon\}$ eine Übergangsrelation
(mit Änderung des Kellerinhaltes)

$\perp \in \Gamma$ Kellerboden-Symbol

- ▶ Akzeptanz durch Zustand / leeren Keller
- ▶ Äquivalenz
- ▶ Transformation von / zu Typ-2-Grammatik