Theoretische Informatik I

Prof. Dr. Carsten Lutz
AG Theorie der künstlichen Intelligenz
Cartesium 2.59

Homepage der Vorlesung:
http://www.informatik.uni-bremen.de/tdki/lehre/ws13/theoinf

Organisatorisches

Vorlesung: Mo 14:00 – 16:00 NW1 H1

Hauptsächlich Folien,
ausgesuchte Beispiele + Beweise an der Tafel

Skript:

- Verfügbar auf Webseite
- Teile I + II: VL Theoretische Informatik I
- Teile III + IV: VL Theoretische Informatik II
- Zusätzliches Material in der VL wird angekündigt

  Mitschreiben!

Literatur

- Skript zur Vorlesung (Webseite)
- Dexter Kozen, Automata and Computability, Springer Verlag 2007
- John Hopcroft, Rajeev Motwani, Jeff Ullmann, Introduction to Automata Theory, Languages, and Computation (3rd edition), Addison Wesley, 2006
- Uwe Schöning, Theoretische Informatik-kurzgefasst, Spektrum Akademischer Verlag, 2001
- Ingo Wegener, Theoretische Informatik-Eine algorithmenorientierte Einführung, Teubner, 1999.

Übungsgruppen

- 8 Gruppen zu unterschiedlichen Terminen, Zuordnung über zentrales Webtool (PI1)
- Beginn kommende Woche
- Jede Woche ein Aufgabenblatt auf VL-Homepage, das in der Übungsgruppe gemeinsam gelöst wird
- Jede zweite Woche werden die Aufgaben abgegeben (bis Mo. 12:00 ins Postfach des Tutors) und korrigiert
- In der kommenden Woche muss nichts abgegeben werden.
- Die Bearbeitung der Aufgaben erfolgt in Gruppen von 2-3 Personen

Scheine / Prüfungen

Prüfungsmodalitäten

- Es gibt 6 gewertete Übungsblätter, insgesamt müssen 50% der Punkte erreicht werden
- Note wird über alle Blätter gemittelt,
  geht in Bachelor-Note ein!
- Zusätzlich Fachgespräch am Ende des Semesters
  (Prüfungsleistung, Änderung der Note möglich)
- Der prüfungsrelevante Stoff wird bestimmt durch
  Vorlesung und Übungen (nicht das Skript)!

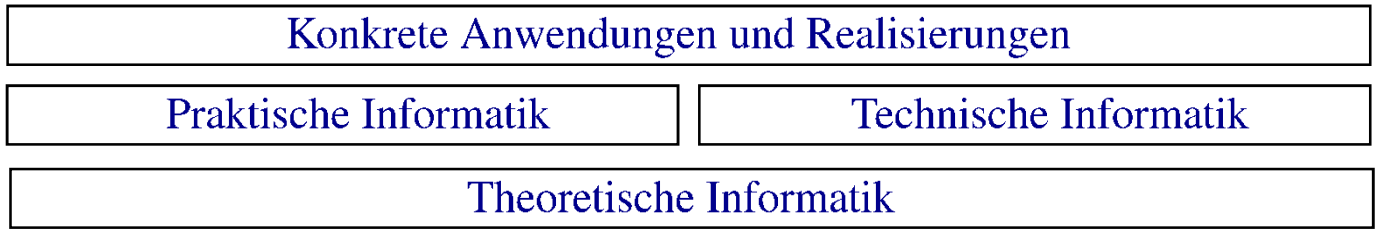
Theoretische Informatik – Eine kurze Einführung

Theoretische Informatik

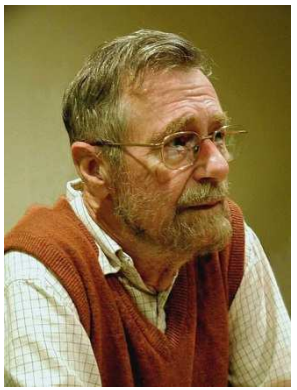
Schafft formale und kulturelle Grundlage für die Informatik

Kultur:

- Gemeinsames Grundwissen: welche allgemeinen Konzepte und Methoden sind zentral für die Disziplin und “common knowledge”?
- Gemeinsame Sprache: welche zentralen Begriffe werden von allen verstanden?



Theoretische Informatik



Dijkstra: In der Informatik geht es genauso wenig um Computer, wie in der Astronomie um Teleskope

Schwerpunkte:

- Schaffen von mathematischen Modellen und Abstraktionen

  Unwichtiges Ausblenden, Wesentliches klar herausstellen
  Grundlage für exakte, mathematische Behandlung informatischer Fragestellungen

- Bereitstellen von Berechnungsmodellen und algorithmischen Techniken

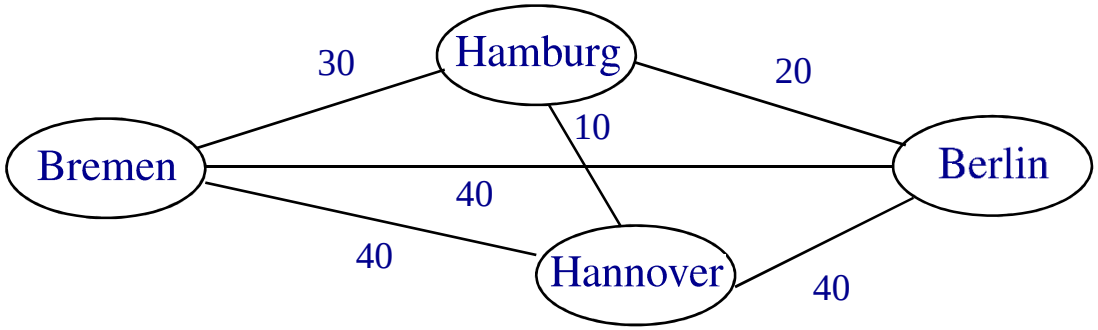
  Abstrakte Modelle von Computern, Programmiersprachen, etc
  Wie unterscheiden sich PC, Tablet, das Web als "großer Computer", einem DNA Computer und einem Quantencomputer?

- Verständnis der Grenzen der (effizienten) Berechenbarkeit

  Kann ich alles berechnen, was ich beschreiben kann (bei vollständiger Information)? Wie effizient kann ich Dinge berechnen?

Theoretische Informatik – Beispiel 1

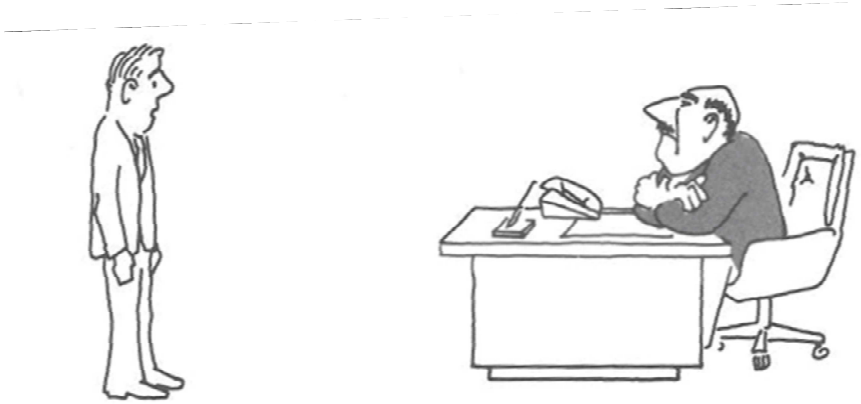
Aufgabe: Finde den günstigsten Weg für eine Rundreise



Das Programm ist nicht sehr effizient? Das liegt nicht am Programmierer!

Denn:

- unbekannt, ob dieses Problem (Travelling Salesman) effizient lösbar
- Frage äquivalent zum wichtigsten offenen Problem in der Informatik/Mathematik



"I can't find an efficient algorithm, I guess I'm just too dumb."

"I can't find an efficient algorithm, but neither can all these famous people."

Theoretische Informatik – Beispiel 2

Aufgabe: Entwerfe eine Raketensteuerung

Theoretische Informatik – Beispiel 2

Aufgabe: Entwerfe eine Raketensteuerung

Problem: Fehler in der Steuersoftware

Theoretische Informatik – Beispiel 2

Klassische Methode zum Finden von Bugs:

Testen! (nach Möglichkeit systematisch)

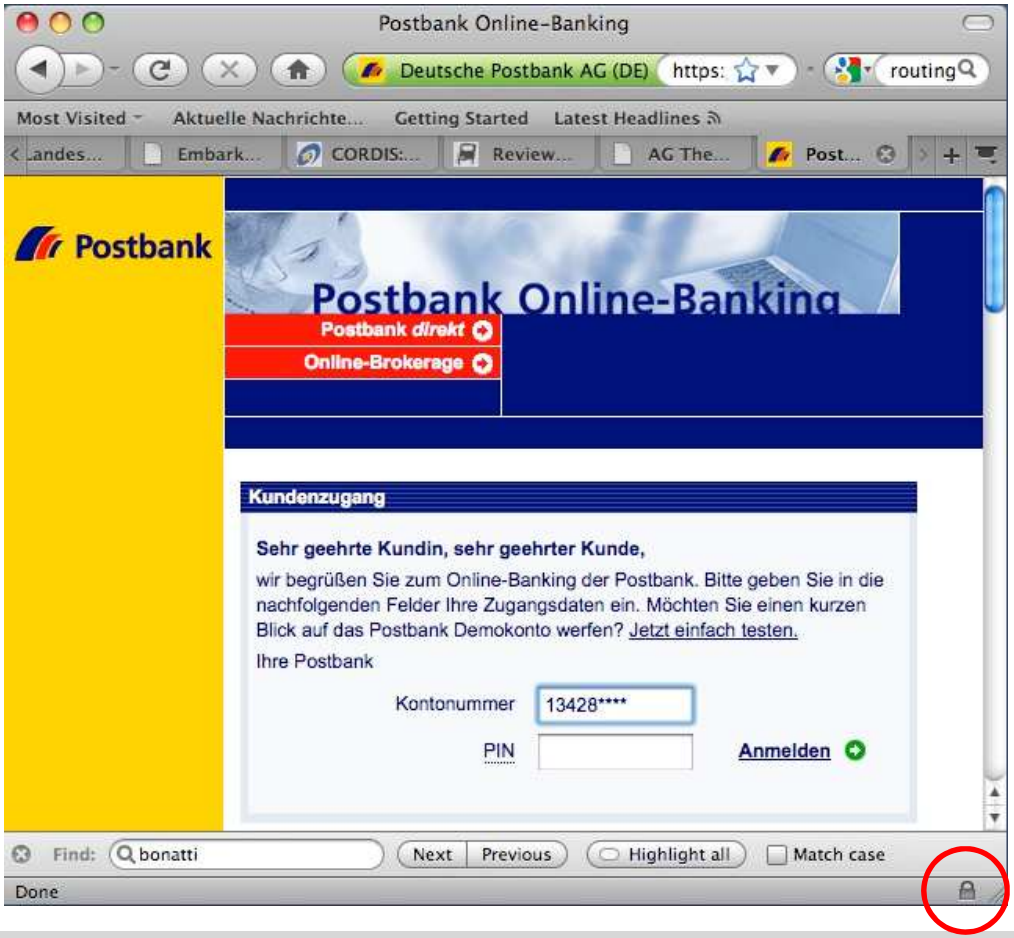
Problem: i.d.R. zu viele mögliche Eingaben, um alle zu testen

In kritischen Anwendungen viel besser: Verifikation

- erlaubt einen formalen Beweis der Korrektheit
  (automatische Analyse des Programmes, kein Testen)
- basiert auf mathematischen Methoden, insb. Logik
- Teilgebiet der theoretischen Informatik

Theoretische Informatik – Beispiel 3

Aufgabe: Verwende Online-Banking, ohne bestohlen zu werden



Was genau bedeutet das?

Theoretische Informatik – Beispiel 3

Das Schloss bedeutet natürlich verschlüsselte Übertragung

Aber es bleiben berechtigte Fragen:

- Kann jemand den Schlüssel abfangen?

  Sehr leicht sogar, aber das macht nichts

- Kann man ganz sicher sein, dass niemand einen Trick gefunden hat, die Verschlüsselung zu brechen?

  Nein, kann man nicht!

- Kann ich der Verschlüsselung vertrauen?

  Ja, durchaus! (für den Hausgebrauch)

Diese Fragen werden in der Kryptographie studiert.

Theoretische Informatik

Besteht aus vielen Teilgebieten:

- Automaten und formale Sprachen - TheoInf I
- Komplexitätstheorie und Theorie der Berechenbarkeit - TheoInf II
- Verifikation und mathematische Logik
- Kryptographie
- Algorithmentheorie
- Datenbanktheorie
- etc.

Zur Rolle der Mathematik

Mathematik

Ab jetzt keine explodierenden Raketen und Comics, sondern: Mathematik

Exakte + formale Beschreibungen essentiell in der Theoretischen Informatik

Lasst Euch nicht abschrecken:

- Mit etwas gutem Willen erkennt man leicht, warum die mathematischen Definitionen und Resultate wichtig für die Informatik sind
- Mathe ist nicht gleich Mathe:
  - Wir benötigen oft andere mathematische Methoden, als Ihr aus der Schule kennt.
  - Wir wollen nicht rechnen (=langweilig), sondern Dinge beweisen (=verstehen)

Mathematik

Die notwendige Mathematik lernt Ihr:

- hier!
- in Mathe I
- durch viel Übung, z.B. in den Übungsgruppen

Zu Eurer Unterstützung:
Verzeichnis der Notation, Symbole und Begriffe findet sich hinten im Skript!

Automaten und formale Sprachen - Einführung

Grundlegende Definitionen

Alphabet:

- endliche Menge von Symbolen
- wir verwenden meist Σ für Alphabete, a, b, c, etc für Symbole
- Beispiele:
  - $\Sigma = \{a, b, \ldots, z\}$ $\quad \Sigma = \{0, 1\}$ $\quad \Sigma = \{0, \ldots, 9\} \cup \{,\}$
  - Σ ist das standardisierte Alphabet ISO-8859-1
  - $\Sigma =$ { program, const, var, label, procedure, function, type, begin, end, if, then, else, case, of, repeat, until, while, do, for, to } $\cup$ { VAR, VALUE, FUNCTION }

Grundlegende Definitionen

Wort:

- endliche Folge von Symbolen
- $w = a_1 \cdots a_n$ mit $a_i \in \Sigma$ heisst Wort über dem Alphabet Σ
- Beispiele:
  - $w = abc$ $\quad w = 1000110$ $\quad w =, , , 10, 0221, 4292, ,$
  - Jedes Pascalprogramm kann als Wort über

    $\Sigma = \{$ program, const, var,label, procedure, function, type, begin,
    end, if, then, else, case, of, repeat, until, while, do, for, to $\}$
    $\cup \{$ VAR, VALUE, FUNCTION $\}$

    betrachtet werden (Abstraktion der Variablen, Werte, Funktionen nötig)
  - Die leere Folge von Buchstaben ist auch ein Wort, das leere Wort ε

Grundlegende Definitionen

Formale Sprache:

- endliche oder unendliche Menge von Wörtern
- Σ^* ist die Menge aller Wörter über Σ, $\quad \Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$
- $L \subseteq \Sigma^*$ heisst Sprache über dem Alphabet Σ
- Beispiele:
  - $L = \emptyset \quad L = \{abc\}, \quad L = \{a, b, c, ab, ac, bc\}$
  - $L = \{w \in \{a, \ldots, z\}^* \mid w$ ist ein Wort der deutschen Sprache $\}$
  - L als Menge aller Worte über
    $\Sigma = \{$ program, const, var,label, procedure, function, type, begin,
    end, if, then, else, case, of, repeat, until, while, do, for, to $\}$
    $\cup \{$ VAR, VALUE, FUNCTION $\}$
    die wohlgeformten Pascal-Programme beschreiben

Formale Sprachen in der Informatik

Formale Sprachen als Abstraktionsmechanismus in der Informatik wichtig:

- Menge aller wohlgeformten Pascal/Java/C++ - Programme
- Menge aller wohlgeformten Eingaben für Programm / Webseite
  z.B. Menge aller Kontonummern / Menge aller Geburtsdaten
- Jeder Suchausdruck definiert Sprache
  z.B. Linux “grep”-Befehl: finde alle Dokumente, die die Wörter “USA” und “Terrorangriff” enthalten
- Kommunikationsprotokolle
  z.B. Menge aller wohlgeformten TCP-Pakete
- “Erlaubte Verhalten” von Soft- und Hardwaresystemen
  z.B. die erlaubten Verhaltensweisen eines Moduls zur Raketensteuerung

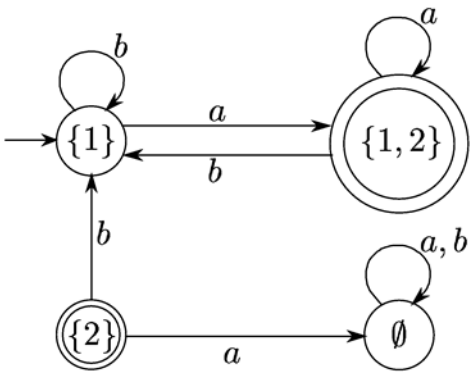
Wichtige Fragestellungen

(a) Charakterisierung:

Interessante Sprachen sind meist unendlich.

Wie beschreibt man unendliche Sprachen mit endlichen Mitteln?

– Automaten: abstrakte Maschinen, die ein Wort als Eingabe erhalten und dieses genau dann akzeptieren, wenn es zur Sprache gehört (endliche Automaten, Kellerautomaten, Turing-Maschinen).



Wichtige Fragestellungen

– Grammatiken, die genau die Worte der Sprache generieren (z.B. kontextfreie Grammatiken für die Syntax von Programmiersprachen).

$$
\begin{aligned}
&G = (N, \Sigma, P, S) \text{ mit} \\
&N = \{S, B\}, \\
&\Sigma = \{a, b, c\} \\
&P = \{S \longrightarrow aSBc, \\
&\qquad S \longrightarrow abc, \\
&\qquad cB \longrightarrow Bc, \\
&\qquad bB \longrightarrow bb\}
\end{aligned}
$$

Wichtige Fragestellungen

- Ausdrücke, die beschreiben, wie man die Sprache aus Basissprachen mittels geeigneter Operationen (z.B. Vereinigung) erzeugen kann.

$$(a+b)^*ab(a+b)^*$$

Chomsky-Hierarchy

Wir werden verschiedene Typen von Sprachen unterscheiden

Die verschiedenen Sprachtypen haben recht unterschiedliche Eigenschaften

Klasse	Automatentyp	Grammatiktyp
Typ 0	Turingmaschine (TM)	allgemeine Chomsky-Grammatik
Typ 1	TM mit linearer Bandbeschränkung	kontextsensitive Grammatik
Typ 2	Kellerautomat	kontextfreie Grammatik
Typ 3	endlicher Automat	einseitig lineare Grammatik

Die aus praktischer Sicht wichtigsten Typen sind Typ 2 und Typ 3:

Typ 3: auch Beschreibung durch reguläre Ausdrücke,
die z.B. als Suchpattern bei der Textsuche Verwendung finden.

Typ 2: können weitgehend die Syntax von Programmiersprachen beschreiben.

Wichtige Fragestellungen

(b) Algorithmische Probleme: was sind die relevanten Berechnungsprobleme für formale Sprachen? Wie löst man sie algorithmisch für verschiedene Sprachklassen?

– Wortproblem: gegeben eine Beschreibung der Sprache L (als Automat, Grammatik,..) und ein Wort w. Gilt $w \in L$?

Anwendungsbeispiele:

* Programmiersprache, deren Syntax durch eine Grammatik beschrieben ist. Entscheide, ob ein gegebenes Programm P syntaktisch korrekt ist.

* Suchpattern als regulärer Ausdruck. Suche die Dateien (= Wörter), welche das Suchpattern enthalten (= zu der durch den Ausdruck beschriebenen Sprache gehören).

Wichtige Fragestellungen

(b) Algorithmische Probleme: was sind die relevanten Berechnungsprobleme für formale Sprachen? Wie löst man sie algorithmisch für verschiedene Sprachklassen?

- Leerheitsproblem: gegeben eine Beschreibung der Sprache L. Ist $L \neq \emptyset$?

  Anwendungsbeispiel:
  Wenn ein Suchpattern die leere Sprache beschreibt, so muß man die Dateien gar nicht durchsuchen.

- Äquivalenzproblem: beschreiben zwei verschiedene Beschreibungen dieselbe Sprache?

  Anwendungsbeispiel:
  Ein komplexes Suchpattern soll durch ein einfaches ersetzt werden. Sind die beiden Pattern äquivalent?

Wichtige Fragestellungen

(c) Abschlußeigenschaften: unter welchen Operationen auf Sprachen (wie Schnitt $L_1 \cap L_2$, Vereinigung $L_1 \cup L_2$, Komplement $\overline{L}$) ist die Sprachklasse abgeschlossen?

Mathematisch von sehr grossem Interesse

Anwendungsbeispiele:

- Suchpattern: suche nach Dateien, die das Pattern nicht enthalten (Komplement)
- suche nach Dateien, die zwei Pattern gleichzeitig enthalten (Schnitt).

Teil I: Endliche Automaten und reguläre Sprachen

0. Grundbegriffe
1. Endliche Automaten
2. Nachweis der Nichterkennbarkeit
3. Abschlußeigenschaften und Entscheidungsprobleme
4. Reguläre Ausdrücke und Sprachen
5. Minimale DEAs und die Nerode-Rechtskongruenz

Teil II: Grammatiken, kontextfreie Sprachen und Kellerautomaten

6. Die Chomsky Hierarchie
7. Rechtslineare Grammatiken und reguläre Sprachen
8. Normalformen und Entscheidungsprobleme
9. Abschlußeigenschaften und Pumping Lemma
10. Kellerautomaten