Kapitel 3: Berechnungstheorie

Gliederung

Zielstellung

- Früher Optimismus …

> *Geben Sie einem Computer die richtige Software und er wird tun, was immer Sie wünschen. Die Maschine selbst mag ihre Grenzen haben, doch für die Möglichkeiten von Software gibt es keine Grenzen.*

„ein Computerspezialist“ zitiert im Time-Magazine, 1984

Zielstellung

- Früher Optimismus …
  (obwohl Theoretische-Informatik-Pioniere es schon lange besser wussten)

> *Geben Sie einem Computer die richtige Software und er wird tun, was immer Sie wünschen. Die Maschine selbst mag ihre Grenzen haben, doch für die Möglichkeiten von Software gibt es keine Grenzen.*

„ein Computerspezialist“ zitiert im Time-Magazine, 1984

Stimmt das?

Zielstellung

- Früher Optimismus …
  (obwohl Theoretische-Informatik-Pioniere es schon lange besser wussten)

> *Geben Sie einem Computer die richtige Software und er wird tun, was immer Sie wünschen. Die Maschine selbst mag ihre Grenzen haben, doch für die Möglichkeiten von Software gibt es keine Grenzen.*

„ein Computerspezialist“ zitiert im Time-Magazine, 1984

Stimmt das?

> *Leider nein. Es gibt unlösbare „algorithmische Probleme“.*

Kurt Gödel 1931
Alan M.Turing 1937

Zielstellung

- Gegenstand der Betrachtung

  - Was sind algorithmische Probleme?
  - Gibt es unlösbare algorithmische Probleme?
    Und wenn ja:
    - Warum?
    - Weiß man nur, dass es welche geben muss - aber keiner kennt sie - oder gibt es ganz konkrete, die man verstehen kann?
    - Gibt es nur einzelne solche „Ausnahmen“, oder gibt es viele?
    - Was nützt uns das praktisch?

Zielstellung

- Gleich vorweg: Was nützt uns das praktisch?

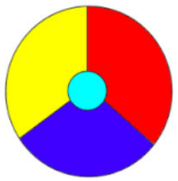
  - Auch wenn man manche Probleme gerne gelöst hätte, in Kenntnis ihrer Unlösbarkeit kann man sich …
    - Zeit,
    - Geld und
    - Mühe sparen,

    indem man die Finger davon lässt!

- Was sind algorithmische Probleme?
  Ideen dazu …

  - Es gibt einige wenige klassische mathematische Fragen
    - die immer noch unbeantwortet sind und
    - die man auch Nicht-Mathematikern gut erklären kann.

  - Es werden aber immer weniger.
    - Z.B. wurde das früher als Beispiel besonders beliebte Vierfarben-Problem in einer Publikation 1989 gelöst –
      - sogar mit massivem Computer-Einsatz,
      - aber erst nachdem manuell das Problem von unendlich vielen Fällen auf (zwar sehr, aber) endlich viele reduziert worden war.



Noch offene mathematische JA-NEIN-Fragen

- Gibt es unendlich viele Primzahl-Zwillinge?
  - Das sind Paare (p,q) von Primzahlen p und q mit q=p+2,
  - wie (3, 5), (11, 13), (17, 19), ….

- Das Collatz-Problem:
  Betrachte für jedes $n \in N$ eine Zahlenfolge $c_{(n)}$ wie folgt:
  - Beginne $c_{(n)}$ mit n, und dann wiederhole mit der letzten konstruierten Zahl x immer den folgenden Schritt :
  - Ist x gerade, so wird x/2 die nächste Zahl.
    Ist x ungerade, so wird 3x+1 die nächste Zahl.
  - Beispiel $c_{(3)}$: 3-10-5-16-8-4-2-1-4-…
- Frage: Endet jede Folge mit der ewigen Wiederholung 4-2-1-4-2-1-… ?
  (Andernfalls enthält die Folge **beliebig große** Zahlen, oder sie führt in eine **andere Schleife**.)

- Algorithmisch sind solche Fragen sehr leicht zu beantworten!

  - Worum geht es ganz präzise? – :

    Gibt es einen Algorithmus, der die Frage korrekt beantwortet?

  - Tatsächlich – egal ob Collatz-Problem oder Primzahlzwilling-Frage – jeweils einer der beiden folgenden Algorithmen beantwortet die JA-NEIN-Frage richtig!

    1. print(JA)

    2. print(NEIN)

    - … klingt wie ein Witz, ist aber keiner!
      Ähnlich kenne ich einen Algorithmus zur Beantwortung von:
      „Regnet es hier in einem Jahr um dieselbe Uhrzeit?“ !

  - Und ähnlich **trivial** ist es bei **endlich vielen** Fragen, die nach **endlich langen** Antworten fragen, denn ich kann jede mir für jede mögliche Antworten-Kombination einen Algorithmus konstruieren, der auf Anfragen genau diese Antworten ausgibt.

- Algorithmisch interessant sind also nur …
  - Fragen mit endlichen Antworten über **unendlich viele mögliche** Eingaben
    - *z.B. „Gib mir für die natürliche Zahl n die Ausgabe n^2 aus.*“
    - oder „*Ist das Wort w ein Palindrom?*“
  - oder evtl. noch Fragen mit unendlich langen Antworten? Aber …
    - Eine unendlich lange Antwort *Ant* liegt mir nie komplett vor,
    - und ich kann sie als Folge von endlichen Antworten auf abzählbar unendlich viele Fragen interpretieren:

      „Wie lauten die Zeichen 1-10 von Ant?“, „Wie 11-20“? usw.
    - Also genügt praktisch der obige, erste Fall.

- Ein Algorithmisches Problem Π besteht hier also aus ...

  - einer unendlichen Menge $X \subseteq \Sigma_{ein}^*$ zulässiger Eingaben;
  - einer Menge $Y \subseteq \Sigma_{aus}^*$ zulässiger Ausgaben;
    - Σ_{ein} und Σ_{aus} sind nicht-leere endliche Alphabete;
  - einer Funktion, welche jeder zulässigen Eingabe (aus X) eine Menge (Teilmenge von Y) korrekter Ausgaben (Antworten, Ergebnisse) zuordnet.
    - Es könnten ja mehrere denkbare Antworten korrekt sein.
      Beispiel: Wie komme ich mit dem Auto von *abc* nach *xyz*?

Π *kann man als Relation* $\Pi \subseteq X \times Y$ *auffassen.*

Beispiele

- Wortproblem für eine gegebene kontextsensitive Grammatik G über einem Alphabet Σ
  - $X = \Sigma^*$
  - $Y = \{ 0,1 \}$
  - $\Pi = \{ (x,1) \mid x \in L(G) \} \cup \{ (x,0) \mid x \notin L(G) \}$

- Collatz-Problem, aber nicht als eine Frage nach allen $c_{(n)}$, sondern als unendlich viele Fragen nach **jedem einzelnen** $c_{(n)}$
  - $X = N$
  - $Y = \{ \text{false,true} \}$
  - $\Pi = \{ (m,\text{true}) \mid \text{die Folge } c_{(m)} \text{ endet mit } 4,2,1,4,2,1,\ldots\} \cup$
    $\{ (m,\text{false}) \mid \text{die Folge } c_{(m)} \text{ endet nicht mit } 4,2,1,4,2,1,\ldots\}$

- lösbare algorithmische Probleme
  - Ein algorithmisches Problem Π ist lösbar, wenn es mindestens ein Computerprogramm gibt, das zu jeder zulässigen Eingabe $x \in X$ ein $y \in Y$ mit $(x,y) \in \Pi$ ausgibt.
- unlösbare algorithmische Probleme
  - Ein algorithmisches Problem Π ist unlösbar, wenn es kein Computerprogramm gibt, das zu jeder zulässigen Eingabe $x \in X$ ein $y \in Y$ mit $(x,y) \in \Pi$ berechnet.

Beispiele

- Wortproblem für eine gegebene kontextsensitive Grammatik G über einem Alphabet Σ

  Wir wissen, dass dieses algorithmische Problem lösbar ist.

- Collatz-Problem

  Es ist offen, ob dieses algorithmische Problem lösbar ist.

Mächtigkeitsvergleich

- Wie viele (mit Text und Formeln aufschreibbare) **Algorithmen** kann es geben?
- Egal in welchem Formalismus:
  Jeder Algorithmus besitzt eine endliche Beschreibung und ist daher eine Zeichenfolge über einem Alphabet Σ_{alg}^* .
- Σ_{alg}^* aber ist abzählbar,
- Also ist die Menge aller Algorithmen, als Teilmenge davon, höchstens abzählbar, und da wir i.a. unendlich viele Algorithmen aufschreiben können (z.B. „read(x); print(n)“ – mit n= 1, 2, 3, …), **abzählbar unendlich**.

Mächtigkeitsvergleich

- Wie viele **algorithmische Probleme** kann es geben?
- Die Eingabesprache X ist abzählbar, also eine Menge $X = \{ x_1, x_2, x_3, \ldots\}$;
- Die Ausgabesprache Y ist abzählbar.
- Selbst wenn wir nur die Teilmenge der algorithmischen Probleme mit einelementigen Mengen zulässiger Ausgaben betrachten, also Funktionen $f: X \rightarrow Y$, so gibt es dann so viele von ihnen wie Folgen
  - $f(x_1)$, $f(x_2)$, $f(x_3)$, …,
  - das sind aber mindestens so viele wie 0-1-Folgen (sofern Y nicht mit trivialen Algorithmen „bedient“ werden kann, was bei $|Y|<2$ der Fall ist: antworte nie, oder immer dasselbe.),
  - also **überabzählbar** viele.

- Mächtigkeitsvergleich

Fazit:

Es gibt „sehr viel mehr“ algorithmische Probleme als Algorithmen.

Für – nicht nur eines, sondern sogar – „die meisten“ algorithmischen Probleme gibt es also keinen Lösungsalgorithmus, der immer korrekte Antworten gibt.

Aber …

- Die Mächtigkeitsbetrachtung ist „nicht-konstruktiv“:

  Wir kennen damit noch kein konkretes Beispiel eines unlösbaren algorithmische Problems.

Plan

- Ein algorithmisches Problem Π ist unlösbar, wenn es kein Computerprogramm gibt, das zu jeder zulässigen Eingabe $x \in X$ ein $y \in Y$ mit $(x,y) \in \Pi$ berechnet.

- Eine solche Klassifizierung sollte unabhängig von der gewählten Programmiersprache und der zugrunde liegenden Hardware sein.

- Daher suchen wir ein
  - möglichst umfassendes und
  - leicht handhabbares
  - abstraktes

  Modell zur Beschreibung einer Maschinerie für alle Algorithmen, und damit indirekt der Menge der lösbaren algorithmischen Probleme.

  Solche Modelle nennt man Berechnungsmodelle.

Anforderungen an ein Berechnungsmodell

- Präzisierung einer Sprache zur Beschreibung von Computerprogrammen, wobei unter anderem die folgenden Fragen zu klären sind:
  - Welche elementare Operationen stehen zur Verfügung?
  - Wie viel Speicher steht zur Verfügung, und wie geht man damit um?
  - Wie funktioniert die Eingabe?
  - Wie funktioniert die Ausgabe?
- Festlegung der Semantik
  - Welche möglichen Ausgaben berechnet ein „Programm“ in dem gewählten Berechnungsmodell aus den Eingaben?

- Hier einige bekannte Berechnungsmodelle:
  - Turing-Maschinen
  - partiell-rekursive Funktionen
  - Markov-Algorithmen
  - Random-Access-Maschinen
  - Lambda-Kalkül
  - Petri-Netze mit Verbotskanten
  - „Kellerautomat“ mit 2 Kellern
  - Quantencomputer
  - DNA-Computer
- ... und die zentrale Beobachtung: ihre Gleichwertigkeit!

> Für jede Funktion f von N_0 nach N_0 gilt entweder (1) oder (2):
>
> (1) f ist in jedem dieser Berechnungsmodelle berechenbar.
> (2) f ist in keinem dieser Berechnungsmodelle berechenbar.

- Wie zeigt man die Gleichwertigkeit von Beschreibungsmitteln?

Braucht man dabei geniale mathematische Geistesblitze?

Nein – es geht durchweg um **handfeste Programmieraufgaben**:

- Durch welche „Kommandofolgen“ im Beschreibungsmittel 2 kann ich die „Kommandos“ der im Beschreibungsmittel 1 verfügbaren Typen „nachbauen“?

  … so wie man beispielsweise eine while-Schleife durch if-then und go-to nachbauen kann …

- Wie kann ich diese „Kommandofolgen“ im Beschreibungsmittel 2 analog zur „Kommandofolge“ im Beschreibungsmittel 1 so verknüpfen, dass ein dasselbe leistender gültiger Algorithmus im Beschreibungsmittel 2 herauskommt?

  … z.B. eine Endkonstellation einer Turing-Maschinen-Berechnung so „aufhübschen“, dass sie als Anfangskonstellation der nächsten Turing-Maschine taugt …

Zwischenbilanz und Ausblick

Konsequenzen

- Um nachzuweisen, dass es algorithmische Probleme gibt, die unlösbar sind, ist es de facto egal, welches „hinreichend vernünftige“ Berechnungsmodell zugrunde gelegt wird.
- Um zu zeigen, dass es algorithmische Probleme gibt, die unlösbar sind, muss man **ein** Berechnungsmodell definieren und „ausreichend gut“ verstehen.

Vorgehensweise

Wir ...

- schauen uns ein Berechnungsmodell genauer an,
- lernen Beispiele für unlösbare algorithmische Probleme kennen und
- stellen abschließend fest, dass ausgerechnet praktisch alle interessanten Fragen über Algorithmen algorithmisch unlösbar sind (Die Algorithmen werden als Wörter $A \in \Sigma_{alg}^*$ einer Sprache eingegeben, aber gefragt wird nach ihren „algorithmischen Leistungen“.)