Berechenbarkeit und Komplexität

Prof. Dr. Dietrich Kuske

FG Automaten und Logik, TU Ilmenau

Sommersemester 2023

1 Einführung

2 Berechenbarkeit

3 Entscheidbarkeit

4 Komplexitätstheorie

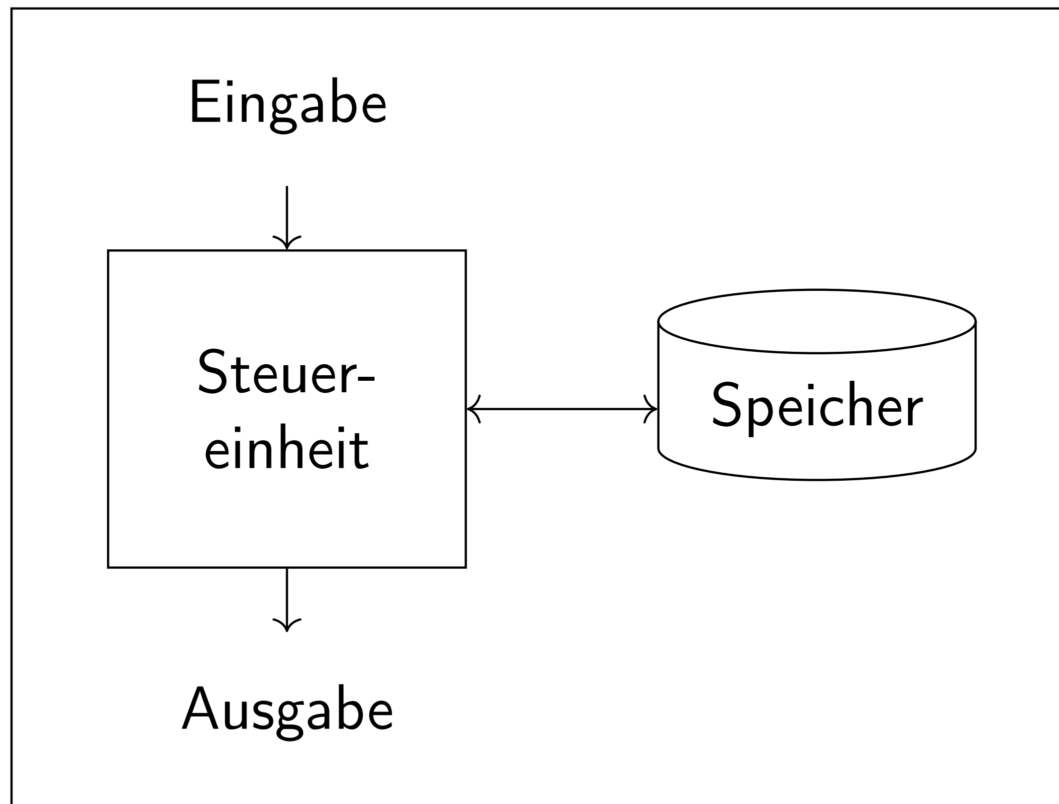
Grundthema von AFS und BuK

Grundfrage

Welche Probleme können von einem Algorithmus bzw. von einer Maschine mit beschränkten Ressourcen gelöst werden und welche können nicht gelöst werden?

- Was ist ein „Problem“?
- Was meinen wir mit „Maschine“?
- Welche „Ressourcen“ betrachten wir und wie werden diese „beschränkt“?

Maschinen



Steuereinheit:

- endlich viele Zustände
- ändern sich in Abhängigkeit von Eingabe und Speicherkonfiguration
- produziert Ausgaben
- gibt „Speicherbefehle“

Eingabe: Folge von „Buchstaben“

Ausgabe: gut/schlecht, Folge von „Buchstaben“ oder $\mathbb{N}$

Probleme

Wir betrachten Probleme als Abbildungen

$$\begin{array}{llcc} f\colon & \text{Menge der mgl. Eingaben} & \longrightarrow & \text{Menge der mgl. Ausgaben} \\ & E & \longrightarrow & A \end{array}$$

$E =$ alle „Wörter über Eingabealphabet“ oder Tupel natürlicher Zahlen
$A =$ gut/schlecht $(\hat{=}\{1,0\})$, alle „Wörter über Ausgabealphabet“, oder natürliche Zahlen

Spezialfall $A = \{0,1\}$
Solche Probleme f heißen Entscheidungsprobleme.
Sie sind gegeben durch die Menge $\left\{w \in E \mid f(w) = 1\right\} \subseteq E$. Solche Mengen nennen wir „Sprachen“.

beschränkte Ressourcen

- Art des Speicherzugriffs:
  verboten / Kellerspeicher / Register / Turing-Band
- Art der Steuereinheit: deterministisch / nichtdeterministisch
- Dauer der Berechnung: polynomiell / exponentiell / ...
- Größe des Speichers: fest / logarithmisch / polynomiell / ...

Ressourcen für BuK

- 5 Leistungspunkte ergeben 150 h Arbeitsaufwand
  (§4 PStO-AB Bachelor, Master und Diplom: „Ein Leistungspunkt entspricht ... einer Arbeitszeit von 30 Stunden.")
- das Semester hat 15 Wochen, also 10 h / Woche
- eine Vorlesung, eine halbe Übung: 2,25 h / Woche Präsenzveranstaltungen
- es bleiben also

$$7,75 \text{ h / Woche für das Selbststudium}$$

(bei unrealistischen 45 h expliziter Prüfungsvorbereitung bleiben noch 4,75 h / Woche Selbststudium)

Organisatorisches zur Vorlesung

Informationen, **aktuelle Version der Folien**, **Übungsblätter** und **Mitschnitte der Vorlesungen**: Website der Veranstaltung (nicht moodle)

Damit wir Informationen an Sie schicken können, melden Sie sich bitte trotzdem im moodle-Kurs an.

Literaturempfehlung: Uwe Schöning: Theoretische Informatik – kurz gefasst, Spektrum Akademischer Verlag

Die **Übungen** führt Herr M.Sc. Schwarz durch.

Prüfung: 90-minütige Klausur.
Bonuspunkte: Die Lösungen der Übungsaufgaben werden abgegeben, (teilweise) korrigiert und sie ergeben (anteilig) 10% der Klausurpunkte.

Abgabe der Übungsaufgaben: bis Donnerstag 13 Uhr in Briefkasten vor Z 1047 (alternativ: am Beginn der Vorlesung)

Arbeitsweise

1. Sie kommen natürlich zu jeder Vorlesung und hören aktiv zu.
2. Der anspruchsvolle Stoff kann nicht durch alleiniges Hören verstanden werden.
3. Daher werden Sie den Vorlesungsstoff semesterbegleitend nacharbeiten: Definitionen („Konzepte“) und Sätze („Sachverhalte“) herausschreiben und auswendig lernen, Beweise („Begründungen“) verstehen (= wiedergeben können), weitere Literatur zu Rate ziehen
4. Sie drucken die Übungsblätter frühzeitig aus, lesen sie genau, arbeiten die Lösung der Hausaufgaben aus und geben diese ab. Diese Lösungen und dabei auftretende Probleme werden in der Übung besprochen.
5. Auch Übungen werden semesterbegleitend nachgearbeitet.
6. Zu jeder Veranstaltung bringen Sie sämtliche Unterlagen zum Nachschlagen mit.
7. Bei Verständnisproblemen fragen Sie bitte frühzeitig!

1 Einführung

2 Berechenbarkeit

3 Entscheidbarkeit

4 Komplexitätstheorie

Die zentrale Frage der Berechenbarkeitstheorie

Welche Funktionen $\mathbb{N} \to \mathbb{N}$ können von einem Algorithmus berechnet werden?

Gilt das vielleicht für alle Funktionen?

vorweggenommene Antwort: nein

Für welche Sprachen L kann das Wortproblem („$w \in L$?“) von einem Algorithmus gelöst werden?

Gilt das vielleicht für alle Sprachen?

vorweggenommene Antwort: nein

Zunächst: Wie zeigt man diese negativen Resultate?
Was heißt überhaupt „die Funktion $f : \mathbb{N} \to \mathbb{N}$ kann von einem Algorithmus berechnet werden“ bzw. „das Wortproblem kann von einem Algorithmus gelöst werden“?

Intuitiver Berechenbarkeitsbegriff

Eine Funktion $f\colon \mathbb{N}^k \to \mathbb{N}$ ist intuitiv berechenbar, wenn es einen Algorithmus gibt, der f berechnet, d.h.

- das Verfahren erhält $(n_1, \ldots, n_k)$ als Eingabe,
- terminiert nach endlich vielen Schritten
- und gibt $f(n_1, \ldots, n_k)$ aus.

Wir nehmen zusätzlich an, daß es ein Alphabet Γ gibt, so daß jeder Algorithmus als Wort über Γ beschrieben werden kann.

Behauptung

Es gibt eine totale Funktion $f : \mathbb{N} \to \mathbb{N}$, die nicht intuitiv berechenbar ist.

Begründung:

Sei $\mathcal{P} \subseteq \Gamma^*$ die Menge der syntaktisch korrekten Algorithmenbeschreibungen. Wörter aus $\Gamma^* \setminus \mathcal{P}$ fassen wir als „Algorithmenbeschreibungen mit syntaktischen Fehlern“ auf.

Für $w \in \Gamma^*$ definieren wir eine Funktion $[\![w]\!] : \mathbb{N} \to \mathbb{N}$ wie folgt:

Gilt $w \in \mathcal{P}$, so ist $[\![w]\!] : \mathbb{N} \to \mathbb{N}$ die von der Algorithmenbeschreibung w berechnete Funktion.
Andernfalls setze $[\![w]\!](n) = 0$ für alle $n \in \mathbb{N}$.

Wir nehmen o.E. $\Gamma = \{0, 1, \ldots, b-1\}$ an.
Für $w = a_k a_{k-1} \ldots a_1 a_0 \in \Gamma^*$ sei $(w)_b = b^{k+1} + \sum_{k \geq i \geq 0} a_i b^i - 1$.
Dann ist $w \mapsto (w)_b$ eine Bijektion von Γ^* auf $\mathbb{N}$;
sei $w\colon \mathbb{N} \to \Gamma^*$ die Umkehrabbildung, d.h. $w(n)$ ist die „n-te Algorithmenbeschreibung (u.U. mit syntaktischen Fehlern)“.

Wir definieren die Funktion $f\colon \mathbb{N} \to \mathbb{N}$ gemäß

$$f(n) = [\![w(n)]\!](n) + 1\,.$$

Sei jetzt $w \in \Gamma^*$. Mit $n = (w)_b$ gilt $w = w(n)$. Dann haben wir

$$[\![w]\!](n) \neq [\![w]\!](n) + 1 = [\![w(n)]\!](n) + 1 = f(n)\,,$$

also $[\![w]\!] \neq f$. Die Funktion f ist also keine der Funktionen $[\![w]\!]$, insbesondere wird sie von keinem der Algorithmen mit Beschreibung in $\mathcal{P}$ berechnet. □

Veranschaulichung: trage in eine Tabelle die Funktionen $[\![w]\!]$ für $w \in \Gamma^*$ ein

Zum Beispiel:

n	$[\![w(0)]\!](n)$	$[\![w(1)]\!](n)$	$[\![w(2)]\!](n)$	$[\![w(3)]\!](n)$	$[\![w(4)]\!](n)$ $\cdots$
0	7	20	33	0	12
1	12	33	94	2	17
2	99	101	17	11	22
3	2	0	14	99	42
4	17	5	77	7	11
$\vdots$					

Veranschaulichung: trage in eine Tabelle die Funktionen $[\![w]\!]$ für $w \in \Gamma^*$ ein

Zum Beispiel: Alle Zahlen auf der Diagonale um eins erhöhen. Dadurch erhält man f.

n	$[\![w(0)]\!](n)$	$[\![w(1)]\!](n)$	$[\![w(2)]\!](n)$	$[\![w(3)]\!](n)$	$[\![w(4)]\!](n)$ $\cdots$
0	7 8	20	33	0	12
1	12	33 34	94	2	17
2	99	101	17 18	11	22
3	2	0	14	99 100	42
4	17	5	77	7	11 12
$\vdots$					

Veranschaulichung: trage in eine Tabelle die Funktionen $[\![w]\!]$ für $w \in \Gamma^*$ ein

Die Funktion f kann aber aufgrund dieser Konstruktion mit keiner der anderen Funktionen übereinstimmen.

n	$[\![w(0)]\!](n)$	$[\![w(1)]\!](n)$	$[\![w(2)]\!](n)$	$[\![w(3)]\!](n)$	$[\![w(4)]\!](n)$ $\cdots$
0	7 8	20	33	0	12
1	12	33 34	94	2	17
2	99	101	17 18	11	22
3	2	0	14	99 100	42
4	17	5	77	7	11 12
$\vdots$					

Dies ist wieder ein Diagonalisierungsbeweis (vgl. 6. Vorlesung Automaten und Formale Sprachen).

Zentrale Frage (vgl. Folien 1.11): Was kann man berechnen?

Welche Funktionen $\mathbb{N} \to \mathbb{N}$ können von einem Algorithmus berechnet werden?

Gilt das vielleicht für alle Funktionen?

Antwort: nein $\surd$

Für welche Sprachen L kann das Wortproblem („$w \in L$?“) von einem Algorithmus gelöst werden?

Gilt das vielleicht für alle Sprachen?

Antwort: nein (Beweis analog) $\surd$

Aber:

- Kann vielleicht jede „interessante“ bzw. „natürliche“ Funktion von einem Algorithmus berechnet werden?
- Wie kann man von einer konkreten Funktion zeigen, daß sie nicht von einem Algorithmus berechnet werden kann? Was heißt eigentlich „von einem Algorithmus berechnet“? Was ist ein Algorithmus genau?

Loop-Berechenbarkeit

Wir betrachten eine einfache Programmiersprache.

- Die Programme haben Variablen x_i, die mit natürlichen Zahlen belegt sind. Diesen Variablen dürfen arithmetische Ausdrücke (mit Konstanten, Variablen, Inkrementierungen und Dekrementierungen) zugewiesen werden.
- Außerdem enthalten die Programme ein Schleifenkonstrukt.

Syntaktische Komponenten für Loop-Programme

Variablen: $x_1, x_2, x_3, \ldots$ Konstanten: 0 und 1
Trennsymbole: ; und := Schlüsselwörter: `loop`, `do`, `end`
Operatorsymbole: $+$, $\dot{-}$

Definition

Ein Loop-Programm ist von der Form

- $x_i := c$, $x_i := x_j + c$, $x_i := x_j \dot{-} c$ mit $c \in \{0, 1\}$ und $i, j \geq 1$ (Wertzuweisung) oder
- $P_1; P_2$, wobei P_1 und P_2 Loop-Programme sind (sequentielle Komposition) oder
- `loop` x_i `do` P `end`, wobei P ein Loop-Programm ist und $i \geq 1$.

Informelle Beschreibung der Semantik

- Ein Loop-Programm, das eine k-stellige Funktion berechnen soll, startet mit den Parametern in den Variablen $x_1, \ldots, x_k$. Alle anderen Variablen haben den Startwert 0. Das Ergebnis liegt bei Terminierung in x_1.
- Interpretation der Wertzuweisungen:
  - $x_i := c$, $x_i := x_j + c$ – wie üblich
  - $x_i := x_j \dot{-} c$ – modifizierte Subtraktion: falls $c > x_j$, so ist das Resultat gleich 0, sonst $x_j - c$
- Sequentielle Komposition $P_1; P_2$: erst P_1, dann P_2 ausführen.
- `loop` x_i `do` P `end`: das Programm P wird so oft ausgeführt, wie die Variable x_i zu Beginn angibt.

Definition

Die modifizierte Subtraktion $\dot{-}$ ist definiert durch
$\dot{-}\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}\colon (m, n) \mapsto \max(0, m - n)$

Definition

Für jedes Loop-Programm P, in dem keine Variable x_i mit $i > k$ vorkommt, definieren wir zunächst eine Funktion $[\![P]\!]_k : \mathbb{N}^k \to \mathbb{N}^k$ durch Induktion über den Aufbau von P. Sei hierfür $(n_1, \ldots, n_k) \in \mathbb{N}^k$.

- $[\![x_i := c]\!]_k (n_1, \ldots, n_k) = (n_1, \ldots, n_{i-1}, c, n_{i+1}, \ldots, n_k)$
- $[\![x_i := x_j + c]\!]_k (n_1, \ldots, n_k) = (n_1, \ldots, n_{i-1}, n_j + c, n_{i+1}, \ldots, n_k)$
- $[\![x_i := x_j \dot{-} c]\!]_k (n_1, \ldots, n_k) = (n_1, \ldots, n_{i-1}, n_j \dot{-} c, n_{i+1}, \ldots, n_k))$
- $[\![P_1; P_2]\!]_k (n_1, \ldots, n_k) = [\![P_2]\!]_k ([\![P_1]\!]_k (n_1, \ldots, n_k))$
- $[\![\texttt{loop } x_i \texttt{ do } P \texttt{ end}]\!]_k (n_1, \ldots, n_k)$
  $$= [\![P]\!]_k^{n_i} (n_1, \ldots, n_k)$$
  $$= [\![P]\!]_k \left(\cdots \left([\![P]\!]_k ([\![P]\!]_k (n_1, \ldots, n_k)) \right) \cdots \right)$$

Sei im folgenden $\pi_i^\ell \colon \mathbb{N}^\ell \to \mathbb{N} \colon (n_1, \ldots, n_\ell) \mapsto n_i$ (Projektionsfunktion).

Definition

Eine Funktion $f \colon \mathbb{N}^k \to \mathbb{N}$ (mit $k \geq 0$) heißt loop-berechenbar, falls es ein $\ell \geq k$ und ein Loop-Programm P, in dem höchstens die Variablen $x_1, \ldots, x_\ell$ vorkommen, gibt mit:

$$\forall n_1, \ldots, n_k \in \mathbb{N} \colon f(n_1, \ldots, n_k) = \pi_1^\ell\big([\![P]\!]_\ell(n_1, \ldots, n_k, 0, \ldots, 0)\big).$$

Beispiel

Sei $c \in \mathbb{N}$. Dann ist die Funktion $\mathrm{const}_0 \colon \mathbb{N}^0 = \{()\} \to \mathbb{N} \colon () \mapsto c$ loop-berechenbar.

Beweis: Setze $k = 0$, $\ell = 1$ und betrachte das Loop-Programm

$$\underbrace{x_1 := x_1 + 1; x_1 := x_1 + 1; \ldots; x_1 := x_1 + 1}_{c \text{ mal}}.$$

□

Geschichtlicher Hintergrund

Diophant von Alexandria (um 250 n. Chr.) untersuchte ganzzahlige multivariate Polynome auf die Existenz von ganzzahligen Nullstellen.

Seitdem hat dieses Problem viele Mathematiker beschäftigt. Typische Ergebnisse lauten: Wenn das Polynom *p so und so aussieht*, dann kann man *so und so* feststellen, ob es eine ganzzahlige Nullstelle hat.

Im Jahr 1900 stellte David Hilbert (1862-1943) u.a. das folgende „10. Hilbertsche Problem“:

> *„Eine Diophantische Gleichung mit irgend welchen 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.“*

in heutigem Deutsch:

> *„Man gebe ein Verfahren an, das für ein beliebiges ganzzahliges multivariates Polynom entscheidet, ob es eine ganzzahlige Nullstelle hat.“*

Mit der Zeit kam man auf die Idee, daß es vielleicht ein solches Verfahren nicht gibt. Um diese Nicht-Existenz zu zeigen, muß zunächst genau gesagt werden, was denn ein „Verfahren“ ist, d.h., wann eine Funktion als berechenbar gelten soll.

Loop-Vermutung

Eine Funktion $\mathbb{N}^k \to \mathbb{N}$ mit $k \geq 0$ ist genau dann intuitiv berechenbar, wenn sie loop-berechenbar ist.

Dies ist keine mathematische, sondern eine erkenntnistheoretische Vermutung, denn der Begriff „intuitiv berechenbar“ ist nicht genau definiert. Natürlich ist jede loop-berechenbare Funktion auch intuitiv berechenbar.

Folgende Argumente können mich davon überzeugen, daß die Loop-Vermutung falsch ist, d.h. daß es intuitiv berechenbare Funktionen gibt, die nicht loop-berechenbar sind:

(K^-) Gib eine konkrete Funktion an, überzeuge mich, daß sie intuitiv berechenbar ist, und beweise, daß sie nicht loop-berechenbar ist.

(A^-) Gib Abschlußeigenschaften an, überzeuge mich, daß die Klasse der intuitiv berechenbaren Funktionen sie erfüllt, und beweise, daß die Klasse der loop-berechenbaren Funktionen sie nicht erfüllt.

Umgekehrt kann die Loop-Vermutung nur gestützt werden, wie z.B. physikalische Gesetze gestützt werden:

Versuche, die Loop-Vermutung zu widerlegen, und scheitere dabei.

Mit anderen Worten:

(K^+) Beweise von vielen Funktionen, daß sie loop-berechenbar sind.

(A^+) Beweise von vielen Abschlußeigenschaften, daß die Klasse der loop-berechenbaren Funktionen sie erfüllt.

ein weiteres mögliches Argument:

(U^+) Verschiedene Wissenschaftler schlagen unabhängig voneinander Algorithmenbegriffe vor, die sich alle als äquivalent herausstellen.

(K^+) Viele loop-berechenbare Fktn. (vgl. Folie 1.24 f.)

Loop-Programme können gewisse Programmkonstrukte simulieren, die in der Syntax nicht enthalten sind.

- Simulation von $\texttt{if}\ x_1 = 0\ \texttt{then}\ A\ \texttt{end}$ (x_n sei neue Variable)
  $x_n := 1;\ \texttt{loop}\ x_1\ \texttt{do}\ x_n := 0\ \texttt{end};\ \texttt{loop}\ x_n\ \texttt{do}\ A\ \texttt{end}$
- Simulation von $\texttt{if}\ x_1 \neq 0\ \texttt{then}\ A\ \texttt{end}$ (x_n sei neue Variable)
  $x_n := 0;\ \texttt{loop}\ x_1\ \texttt{do}\ x_n := 1\ \texttt{end};\ \texttt{loop}\ x_n\ \texttt{do}\ A\ \texttt{end}$
- Simulation von $x_i := x_j + x_k$ (x_n sei neue Variable)
  $x_n := x_j;\ \texttt{loop}\ x_k\ \texttt{do}\ x_n := x_n + 1\ \texttt{end}; x_i := x_n$
- Simulation von $x_i := x_j \dot{-} x_k$ (x_n sei neue Variable)
  $x_n := x_j;\ \texttt{loop}\ x_k\ \texttt{do}\ x_n := x_n \dot{-} 1\ \texttt{end}; x_i := x_n$
- Simulation von $x_i := x_j \cdot x_k$ (x_n sei neue Variable)
  $x_n := 0;\ \texttt{loop}\ x_k\ \texttt{do}\ x_n := x_n + x_j\ \texttt{end}; x_i := x_n$

- Simulation von `if` $x_i \le x_j$ `then` A `end` (x_n sei neue Variable)
  $x_n := x_i \dot{-} x_j$; `if` $x_n = 0$ `then` A `end`
- Simulation von $x_i := x_j$ `div` x_k (x_n ist neue Variable)
  $x_n := 0$;
  `loop` x_j `do if` $x_n \cdot x_k \le x_j$ `then` $x_n := x_n + 1$ `end end`;
  $x_1 := x_n \dot{-} 1$

Zusammenfassung 1. Vorlesung

in dieser Vorlesung neu

- zentrale Frage der Berechenbarkeitstheorie: Was ist intuitiv berechenbar?
- es gibt Funktion $f\colon \mathbb{N} \to \mathbb{N}$, die nicht intuitiv berechenbar ist
- Versuch der Beantwortung mit Hilfe der Loop-Berechenbarkeit
- die Loop-Vermutung als nicht mathematisch beweisbare Aussage
- viele Argumente der Form (K^+) für die Loop-Vermutung

kommende Vorlesung

- viele Argumente der Form (A^+) für die Loop-Vermutung
- Hilberts Vermutung: intuitiv berechenbar = primitiv-rekursiv
- Argument der Form (U^+) für die Loop-Vermutung: loop-berechenbar = primitiv-rekursiv