

Berechenbarkeit und Komplexität

Dozent: Mathias Weller (Skript adaptiert von Rolf Niedermeier)

Betreuer: Leon Kellerhals, Vincent Froese und Philipp Zschoche

Sekretariat: Christlinde Thielcke

Viele Fleißige Tutorinnen und Tutoren

Fachmentorin: Niloofar Nazemi

TU Berlin

Fakultät IV

Fachgebiet Algorithmik und Komplexitätstheorie

<https://www.akt.tu-berlin.de>

Informatikstudium an der TU Berlin

1. Semester 27 LP	Rechnerorganisation (6 LP)	Einführung in die Programmierung (6 LP)	Informatik Propädeutikum (3 LP)	Analysis I und Lineare Algebra für Ingenieurwissenschaften (12 LP)			
2. Semester 30 LP	Systemprogrammierung (6 LP)	Algorithmen und Datenstrukturen (6 LP)	Informationssysteme und Datenanalyse (6 LP)	Formale Sprachen und Automaten (6 LP)	Diskrete Strukturen (6 LP)		
3. Semester 30 LP	Rechnernetze und Verteilte Systeme (6 LP)	Softwaretechnik und Programmierparadigmen (6 LP)	Wissenschaftliches Rechnen (6 LP)	Berechenbarkeit und Komplexität (6 LP)	Logik (6 LP)		
	Wahlpflicht Technische Informatik (6 LP)	Wahlpflicht Programmierpraktikum (6–9 LP)		Wahlpflicht Theoretische Informatik (6 LP)	Stochastik für Informatik (9 LP)		
4.-6. Semester 93 LP	Wahlpflichtbereich Katalog Informatik (27–33 LP)			Informatik und Gesellschaft (6 LP)			
	Wahlbereich (15–18 LP)			Bachelorarbeit (12 LP)			

LP = Leistungspunkte nach dem ECTS-System (1 LP entspricht etwa 30 Zeitstunden)

■ Technische Grundlagen der Informatik ■ Methodisch-praktische Grundlagen der Informatik

■ Theoretische Informatik ■ Grundlagen des wiss. Arbeitens/Informatik in gesellschaftlicher Relevanz

■ Grundlagen der Mathematik ■ Wahlpflichtbereich ■ Wahlbereich □ Bachelorarbeit

Organisation

Vorlesungsbetrieb:

- ▶ Vorlesung: Screencasts und PDF-Folien verfügbar über ISIS
- ▶ "Freie Großübung" + "Modulkonferenz"

Tutorien:

- ▶ Tutorien: siehe ISIS und MOSES
- ▶ Tutor*innensprechstunde: TBA

Prüfungen: Portfolioprüfung

- ▶ Multiple-Choice-Test: 25 PP (ca. Mitte Dezember)
- ▶ Hausaufgabe in Dreiergruppen 25 PP (im Januar)
- ▶ Schriftlicher Test: 50 PP (Termin wird über ISIS bekanntgegeben)

Ergänzendes Material

Literatur:

- ▶ Uwe Schöning. *Theoretische Informatik–kurz gefasst*. Spektrum Akademischer Verlag 2008 (5. Auflage).
- ▶ Elaine Rich. *Automata, Computability, and Complexity*. Pearson 2008.
- ▶ Cristopher Moore, Stephan Mertens. *The Nature of Computation*. Oxford University Press 2011.

Weiteres Material:

YouTube-Kanal NLogSpace (https://www.youtube.com/channel/UCMWYg3eBFp5bbqjlllUku_w)

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem
7. Aufzählbarkeit & (Semi-)Entscheidbarkeit
8. Reduzierbarkeit
9. Satz von Rice
10. Das Postsche Korrespondenzproblem
11. Komplexität – Einführung
12. NP-Vollständigkeit
13. coNP
14. PSPACE

Das „hello, world“-Problem

Ziel: Entwicklung von Programm E mit folgender Spezifikation:

Input: Programm P

Output: „Top“, wenn P den string „hello, world“ ausgibt, „Flop“, sonst

Bemerkung: E hat „Typ höherer Ordnung“ (d.h. Eingabe ist (Text eines) Programms P).

Beispiel für Eingabe P

```
main(){  
    printf("hello, world");  
}
```

- ~ Existiert ein Programm E für diese spezielle Eingabe P ?
- ~ Existiert ein Programm E auch für **beliebige** Programme P ?

Wiederholung: Endliche Automaten

Definition (Endlicher Automat)

- ▶ Ein (**deterministischer**) **endlicher Automat** (kurz **DFA**) ist ein Quintupel $M = (Z, \Sigma, \delta, z_0, E)$ mit
 - ▶ Z ist eine nichtleere, endliche Menge von **Zuständen**,
 - ▶ Σ ist ein nichtleeres, endliches Alphabet von **Eingabezeichen** mit $Z \cap \Sigma = \emptyset$,
 - ▶ $\delta: Z \times \Sigma \rightarrow Z$ ist die **partielle Überführungsfunktion**,
 - ▶ $z_0 \in Z$ ist der **Startzustand** und
 - ▶ $E \subseteq Z$ ist die Menge der **Endzustände**.
- ▶ Zu M definieren wir die partielle Funktion $\hat{\delta}: Z \times \Sigma^* \rightarrow Z$ induktiv für alle $z \in Z$:

$$\hat{\delta}(z, \epsilon) := z$$

$$\forall_{x \in \Sigma^*} \quad \hat{\delta}(z, ax) := \hat{\delta}(\delta(z, a), x) \quad \text{falls } \delta(z, a) \neq \perp$$

- ▶ Ein DFA $M = (Z, \Sigma, \delta, z_0, E)$ **akzeptiert** ein Wort $w \in \Sigma^*$ falls $\hat{\delta}(z_0, w) \in E$
- ▶ Die von M **akzeptierte Sprache** ist $T(M) := \{x \in \Sigma^* \mid \hat{\delta}(z_0, x) \in E\}$.

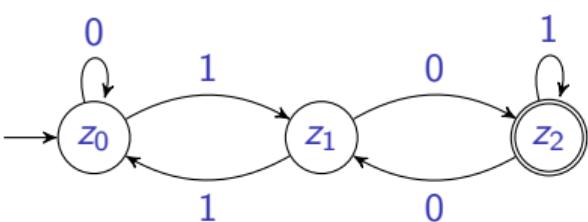
Wiederholung: Endliche Automaten II

Beispielautomat

$M = (\{z_0, z_1, z_2\}, \{0, 1\}, \delta, z_0, \{z_2\})$ mit

δ	z_0	z_1	z_2
0	z_0	z_2	z_1
1	z_1	z_0	z_2

Zustandsgraph



$z_i \rightsquigarrow$ das bisher gelesene Wort ist die Binärkodierung einer Zahl n mit Rest i modulo 3.

$$T(M) = \{w \in \{0, 1\}^* \mid w \text{ ist Binärdarstellung einer Zahl } n \text{ mit } n \bmod 3 = 2\}.$$

Frage: Sind die Binärdarstellungen der Zahlen n mit $n \bmod 4 = 1$ von einem DFA erkennbar?

Grenzen endlicher Automaten

Gibt es jeweils einen endlichen Automaten zur Erkennung folgender Sprachen?

- $\{w \in \{0,1\}^* \mid w \text{ ist Binärdarstellung einer geraden Zahl}\}$ ✓

Letztes Zeichen muss eine 0 sein.

- $\{a^n b^n \mid 0 \leq n \leq 1000\}$ ✓

Sprache enthält nur 1001 Wörter. Jede endliche Sprache ist regulär.

- $\{a^n b^n \mid n \geq 0\}$ ✗

Mit endlich vielen Zuständen können wir uns nicht „merken“ wieviele a 's wir schon gelesen haben.
Erkennung mit Kellerautomaten möglich.

- $\{(abc)^n \mid n \geq 0\}$ ✓

ähnlich zur „ $n \bmod 3 = 2$ “ Sprache.

- $\{a^n b^m c^k \mid n, m, k \geq 0\}$ ✓

Müssen uns nur „merken“ welche Buchstaben noch folgen dürfen.

- $\{a^n b^n c^n \mid n \geq 0\}$ ✗

Siehe $a^n b^n$. Erkennung mittels Turing-Maschinen.

- $\{a^i b^j c^i d^j \mid i, j \geq 0\}$ ✗

Siehe oben. Erkennung mittels Turing-Maschinen.

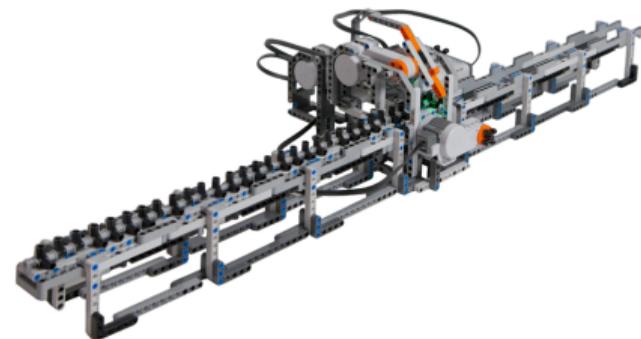
Die Turing Maschine



Alan Mathison Turing, 1912-1954.



Inspiration: „Menschliche Computer“ 1890.



LEGO Turing Maschine

Informell

endliche Kontrolle + unendliches Band

Quellen:

<http://therunnereclectic.files.wordpress.com/2014/11/alan-turing-running.jpg>
http://en.wikipedia.org/wiki/Harvard_Computers
http://cs.cmu.edu/~soonhok/images/20120718_LegoTM/legotm.png

Definition Turing-Maschinen

Definition ((deterministische) Turing-Machine)

Eine **Turing-Maschine** (kurz **DTM**) ist ein Septupel $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ mit

- ▶ Z , einer nicht-leeren, endlichen Menge von **Zuständen**,
- ▶ Σ , dem **Eingabealphabet**,
- ▶ $\Gamma \supseteq \Sigma$, dem **Arbeits-** oder **Bandalphabet** mit $\Gamma \cap Z = \emptyset$,
- ▶ $\delta: (Z \setminus E) \times \Gamma \rightarrow Z \times \Gamma \times \{L, R, N\}$, die **partielle Überführungsfunktion**
- ▶ $z_0 \in Z$, dem **Startzustand**,
- ▶ $\square \in \Gamma \setminus \Sigma$, dem **Blanksymbol** und
- ▶ $E \subseteq Z$, der Menge der **Endzustände**.

Interpretation: Wenn M im Zustand z das Zeichen a liest und $\delta(z, a) = (z', a', p)$, so

- ▶ geht M in Zustand z' über,
- ▶ **überschreibt** das a durch ein a'
- ▶ bewegt den Lese/Schreibkopf gemäß p (nach **Links**, **Rechts**, oder gar **Nicht**)

Konfigurationen

Definition (Konfiguration, Folgekonfiguration)

Sei $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ eine TM. Eine **Konfiguration** von M ist ein Wort azb mit $a, b \in \Gamma^*$ und $z \in Z$. (überflüssige \square -Symbole an den Rändern der Konfiguration weglassen)

Die **Startkonfiguration** zu einem Wort $x \in \Sigma^*$ ist z_0x .

Sei $k = a_1 \dots a_m z b_1 \dots b_n$ eine Konfiguration (falls $n = 0$, dann $b_1 := \square$). Dann

$$k \vdash_M^0 k$$

$$k \vdash_M^1 \quad a_1 \dots a_m z' c b_2 \dots b_n \quad \text{falls } \delta(z, b_1) = (z', c, N)$$

$$k \vdash_M^1 \quad a_1 \dots a_m c z' b_2 \dots b_n \quad \text{falls } \delta(z, b_1) = (z', c, R)$$

$$k \vdash_M^1 \quad a_1 \dots a_{m-1} z' a_m c b_2 \dots b_n \quad \text{falls } \delta(z, b_1) = (z', c, L) \text{ und } m > 0$$

$$k \vdash_M^1 \quad z' \square c b_2 \dots b_n \quad \text{falls } \delta(z, b_1) = (z', c, L) \text{ und } m = 0.$$

k ist **haltend** (d.h. k hat keine Folgekonfiguration) falls $\delta(z, b_1) = \perp$

k ist **akzeptierend** falls $z \in E$

Weiter sei $k \vdash_M^{i+1} k' \Leftrightarrow \exists q k \vdash_M^1 q \vdash_M^i k'$ für alle i und $k \vdash_M^* k' \Leftrightarrow \exists i \in \mathbb{N} k \vdash_M^i k'$

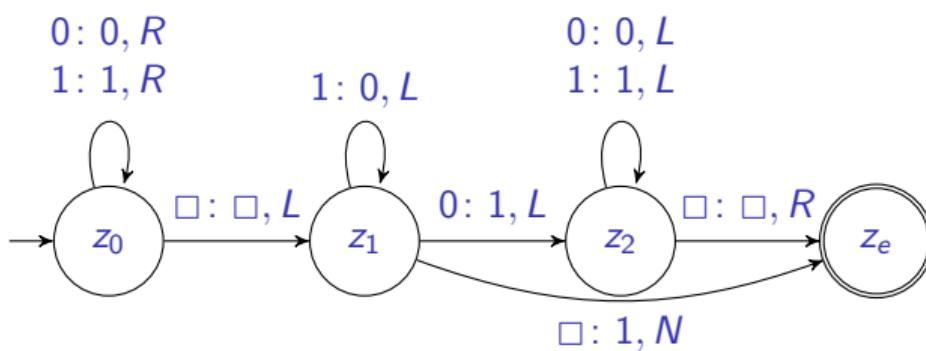
Frage: gibt es akzeptierende Konfigurationen, die nicht haltend sind?

Beispiel Turing-Maschine: Binärzahl inkrementieren

$$M = (\{z_0, z_1, z_2, z_e\}, \Sigma = \{0, 1\}, \Gamma = \{0, 1, \square\}, \delta, z_0, \square, \{z_e\})$$

δ	0	1	\square
z_0	$(z_0, 0, R)$	$(z_0, 1, R)$	(z_1, \square, L)
z_1	$(z_2, 1, L)$	$(z_1, 0, L)$	$(z_e, 1, N)$
z_2	$(z_2, 0, L)$	$(z_2, 1, L)$	(z_e, \square, R)

Zustandsgraph



Frage: Was macht M bei leerer Eingabe? Bei Eingabe 000?

Beispiel: Eingabe 101

$z_0 101$	\vdash_M^1
$1 z_0 01$	\vdash_M^1
$10 z_0 1$	\vdash_M^1
$101 z_0$	\vdash_M^1
$101 z_1 1$	\vdash_M^1
$1 z_1 00$	\vdash_M^1
$z_2 110$	\vdash_M^1
$z_2 \square 110$	\vdash_M^1
$z_e 110$	

$z_e 110$ haltend & akzeptierend

Akzeptieren und Halten einer TM

Definition (Akzeptieren, Halten)

Turing-Maschine $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$:

- ▶ **M hält** auf $w \in \Sigma^*$, falls eine **haltende** Konfiguration k' existiert mit $z_0 w \vdash_M^* k'$.
- ▶ **M akzeptiert** $w \in \Sigma^*$, falls eine **akzeptierende** Konfiguration k' existiert mit $z_0 w \vdash_M^* k'$.
- ▶ **M akzeptiert Sprache** $T(M)$ enthält genau die Wörter w die M akzeptiert. Formal,

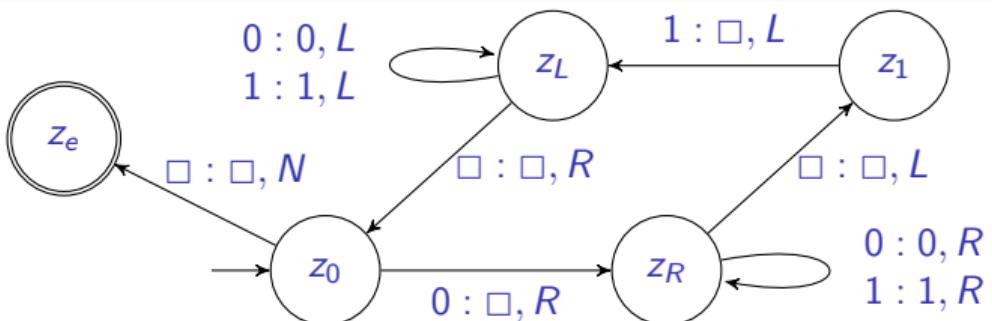
$$T(M) := \{w \in \Sigma^* \mid \exists_{\alpha, \beta \in \Gamma^*} \exists_{z \in E} : z_0 w \vdash_M^* \alpha z \beta\}.$$

Beispiel Turing-Maschine: akzeptiere $\{0^n1^n \mid n \in \mathbb{N}\}$

$M = (\{z_0, z_1, z_R, z_L, z_e\}, \Sigma = \{0, 1\}, \Gamma = \{0, 1, \square\}, \delta, z_0, \square, \{z_e\})$

δ	0	1	\square
z_0	(z_R, \square, R)	\perp	(z_e, \square, N)
z_1	\perp	(z_L, \square, L)	\perp
z_R	$(z_R, 0, R)$	$(z_R, 1, R)$	(z_1, \square, L)
z_L	$(z_L, 0, L)$	$(z_L, 1, L)$	(z_0, \square, R)

Zustandsgraph



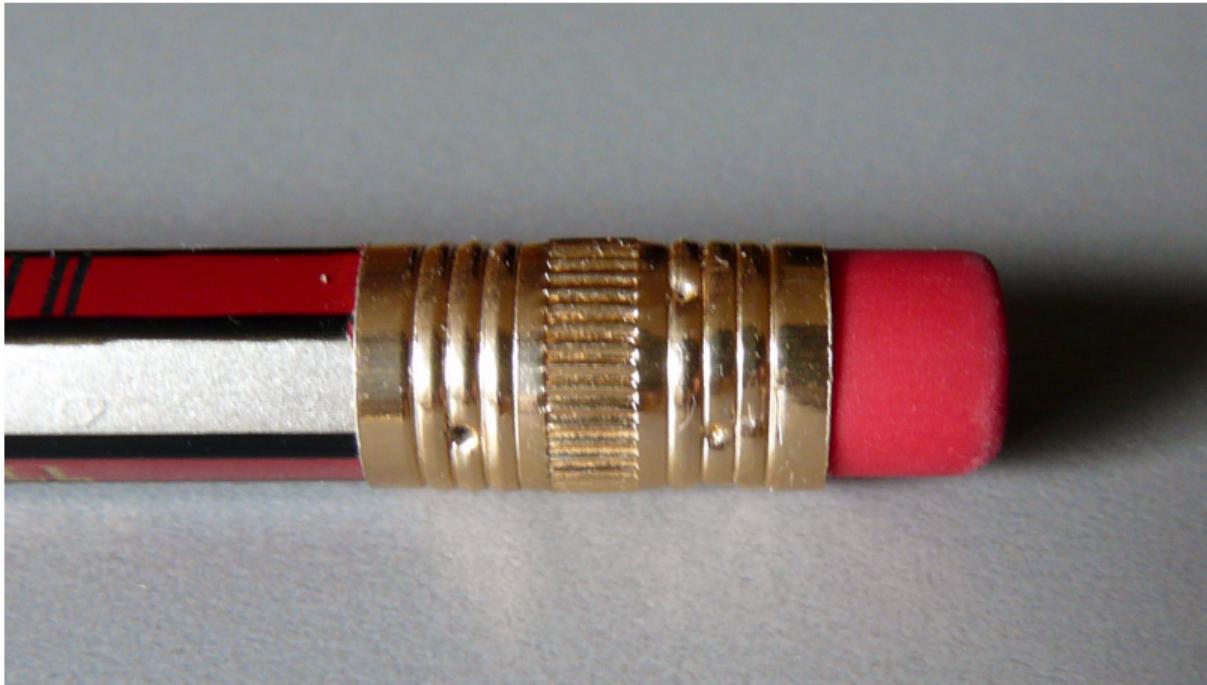
Beispiel: Eingabe 0011

$z_0 0011$	\vdash_M^1
$z_R 011$	\vdash_M^3
$011 z_R$	\vdash_M^1
$01 z_1 1$	\vdash_M^1
$0 z_L 1$	\vdash_M^2
$z_L \square 01$	\vdash_M^1
$z_0 01$	\vdash_M^1
$z_R 1$	\vdash_M^1
$1 z_R$	\vdash_M^1
$z_1 1$	\vdash_M^1
z_L	\vdash_M^1
z_0	\vdash_M^1
z_e	

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem
7. Aufzählbarkeit & (Semi-)Entscheidbarkeit
8. Reduzierbarkeit
9. Satz von Rice
10. Das Postsche Korrespondenzproblem
11. Komplexität – Einführung
12. NP-Vollständigkeit
13. coNP
14. PSPACE

Berechenbarkeitsbegriff

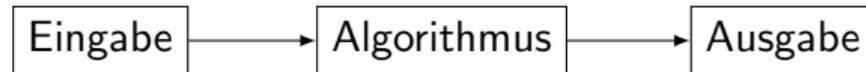


Ziel: Intuitiver Begriff \leadsto mathematische Formalisierung.

Quelle: de.wikipedia.org/wiki/Bleistift#/media/File:Bleistiftzwinge_fcm.jpg

Diskussion intuitiver Berechenbarkeitsbegriff

(Intuitive) Berechenbarkeit von Funktionen:



- ▶ DFA \rightsquigarrow endlicher Speicher nicht "berechnungsmächtig" genug
- ▶ Turing-Maschine \rightsquigarrow Speichergröße unbeschränkt
- ▶ LOOP-/WHILE-/GOTO- Programme \rightsquigarrow Variablengröße unbeschränkt
- ▶ ...

Bemerkung: leichte Diskrepanz zu modernen Computern

Frage: Sind Turing-Maschinen mit endlichem Band genauso "mächtig" wie DFAs?

Berechenbarkeitsbegriff

Definition

Eine (eventuell partielle) Funktion $f: \mathbb{N}^k \rightarrow \mathbb{N}$ heißt **berechenbar**, wenn es einen **endlichen Algorithmus** \mathcal{A} gibt, sodass für alle $n_1, \dots, n_k, m \in \mathbb{N}$ gilt

$$f(n_1, \dots, n_k) = m$$

\Leftrightarrow

bei Eingabe (n_1, \dots, n_k) hält \mathcal{A} nach endlicher Zeit mit Ausgabe m .

Bemerkung: existenzielle Aussage!

Beispiel 1 ($k=1$)

```
1 INPUT (n)
2 WHILE true DO {}
```

$\rightsquigarrow \Omega : \mathbb{N} \rightarrow \mathbb{N}$ mit $n \mapsto \perp$

Berechenbarkeitsbegriff - Beispiele

Beispiel 2

$$f(n) := \begin{cases} 1, & \text{falls } \exists_{i \in \mathbb{N}} \lfloor \pi \cdot 10^i \rfloor = n \\ 0, & \text{sonst} \end{cases}$$

Erläuterung:

$f(n) = 1$ genau dann, wenn n genau den "ersten Dezimalstellen" von π entspricht

Algorithmus

1. approximiere π "ausreichend genau"
2. vergleiche mit Eingabe

Beispiel 3

$$f(n) := \begin{cases} 1, & \text{falls } \exists_{i,j,p \in \mathbb{N}} \text{ sodass } 10^j > n \text{ und} \\ & \lfloor \pi \cdot 10^i - p \cdot 10^j \rfloor = n \\ 0, & \text{sonst} \end{cases}$$

Erläuterung:

$f(n) = 1$ genau dann, wenn n in der Dezimalbruchentwicklung von π vorkommt

?

Beispiel 4

$$f(n) := \begin{cases} 1, & \text{falls } \exists_{i,j,p \in \mathbb{N}} \text{ sodass } j > n \text{ und} \\ & \lfloor \pi \cdot 10^i - p \cdot 10^j \rfloor = \underbrace{11\dots1}_{\times n} \\ 0, & \text{sonst} \end{cases}$$

Erläuterung:

$f(n) = 1$ genau dann, wenn die Dezimalbruchentwicklung von π n konsekutive einsen enthält

✓

Berechenbarkeitsbegriff V

Problem: Berechenbarkeitsbegriff basiert auf Definition von "Algorithmus" . . .

Church'sche **These**

Intuitive Berechenbarkeit = Turing-Berechenbarkeit

Bemerkung:

noch kein echt "mächtigeres" Berechnungsmodell als Turing-Maschine entdeckt

Church'sche These \Rightarrow ein solches gibt es nicht

Turing-Berechenbarkeit I

Definition (Turing-Berechenbarkeit, Entscheidbarkeit)

- Eine (eventuell partielle) Funktion $f: \mathbb{N}^k \rightarrow \mathbb{N}$ heißt **Turing-berechenbar**, wenn es eine DTM $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ gibt, sodass für alle $n_1, \dots, n_k, m \in \mathbb{N}$ gilt
$$f(n_1, \dots, n_k) = m \Leftrightarrow \text{bei Eingabe } (n_1, \dots, n_k) \text{ hält } M \text{ nach endlicher Zeit mit Ausgabe } m.$$
$$\Leftrightarrow \exists_{z \in E} z_0 \text{ BIN}(n_1)\# \dots \# \text{BIN}(n_k) \vdash_M^* z \text{ BIN}(m)$$
wobei **BIN** Zahlen auf ihre Binärdarstellung abbildet.

- Eine (eventuell partielle) Funktion $f: \Sigma^* \rightarrow \Sigma^*$ heißt **Turing-berechenbar**, wenn es eine DTM $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ gibt, sodass für alle $x, y \in \Sigma^*$ gilt

$$f(x) = y \Leftrightarrow \exists_{z \in E} z_0 x \vdash_M^* z y$$

- Eine Sprache L heißt
entscheidbar wenn χ_L berechenbar ist und
semi-entscheidbar wenn χ'_L berechenbar ist

Frage: Wie hängen "Akzeptanz" und "(Semi-)Entscheidbarkeit" zusammen?

Charakteristische Funktion

$$\chi_L(x) = \begin{cases} 1, & \text{falls } x \in L \\ 0, & \text{falls } x \notin L \end{cases}$$

Halbe Charakteristische Fkt.

$$\chi'_L(x) = \begin{cases} 1, & \text{falls } x \in L \\ \perp, & \text{falls } x \notin L \end{cases}$$

Turing-Berechenbarkeit - Beispiele

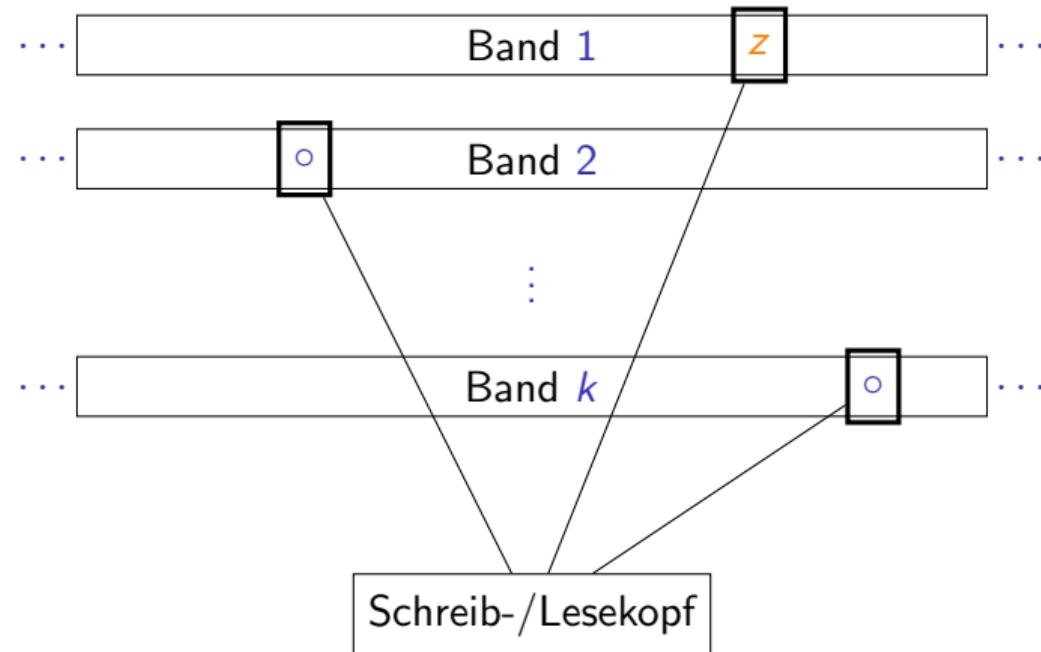
Turing-berechenbar?

1. Nachfolgerfunktion $\text{succ}: \mathbb{N} \rightarrow \mathbb{N}, n \mapsto n + 1$ ✓
2. nirgends definierte „Funktion“ Ω ✓
3. χ_L für L vom Typ 3? ✓
4. χ_L für $L = \{0^n 1^n \mid n \in \mathbb{N}\}$? ✓

Frage: Entspricht unsere TM für 4. genau der Berechenbarkeitsdefinition?

Mehrband-Turing-Maschinen I

...erlauben bequemeres Programmieren (um Berechenbarkeit zu zeigen)



(Deterministische) k -Band Turing-Maschine M

Überführungsfunktion $\delta: (Z \setminus E) \times \Gamma^k \rightarrow Z \times (\Gamma \times \{L, R, N\})^k$.

Konfiguration $\alpha_1 z \beta_1, \alpha_2 \circ \beta_2, \dots, \alpha_k \circ \beta_k$

Startkonfiguration $z_0 x_1, \circ x_2, \dots, \circ x_k$

Folgekonfiguration \vdash_M^1 entsprechend...

Berechnung von Funktionen

$z_0 x_1, \circ x_2, \dots, \circ x_k \vdash_M^* z_e \text{BIN}(f(x_1, \dots, x_k)), \alpha_2 \circ \beta_2, \dots, \alpha_k \circ \beta_k$

Akzeptanz von Sprachen

$z_0 x, \circ \square, \dots, \circ \square \vdash_M^* \alpha_1 z_e \beta_1, \alpha_2 \circ \beta_2, \dots, \alpha_k \circ \beta_k$

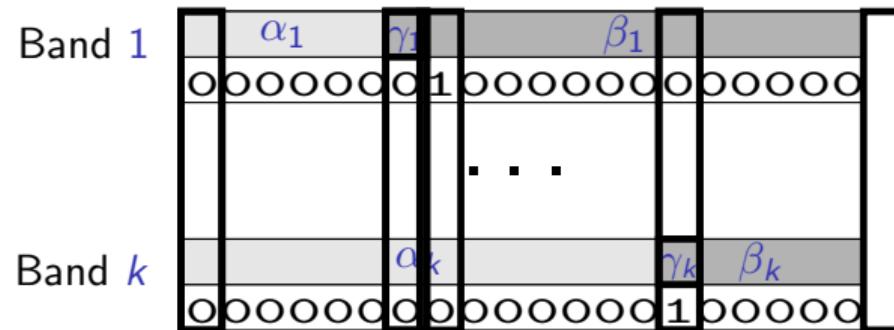
für $z \in Z$ und $z_e \in E$ und $\alpha_1, \dots, \alpha_k, \beta_1, \dots, \beta_k \in \Gamma^*$.

Mehrband-TM Äquivalenz

Theorem

Zu jeder k -Band-TM M gibt es eine Einband-TM Q mit $T(M) = T(Q)$ (bzw. $f_M = f_Q$).

Beweisidee: Q simuliert M mithilfe eines „fetten Bandes“ mit $2k$ „Spuren“:



„speichere“ das ersten Zeichen $\beta_i[0] \in \Gamma$ von β_i für alle $i \leq k$ im Zustand

„speichere“ neues Zeichen $\gamma_i \in \Gamma$ & Kopfrichtung $d_i \in \{L, R, N\}$ für alle $i \leq k$ im Zustand

Mehrband-Turing-Maschinen III

Die Idee besteht darin, eine Einband-Turing-Maschine Q mit einem erweiterten Bandalphabet zu verwenden. Die Elemente des Bandalphabets bestehen dabei jeweils aus $2k$ Zeichen des ursprünglichen Bandalphabets.

Für jedes Band i gibt es 2 „Spuren“, wovon eine den Inhalt von Band i enthält. In der zweiten Spur befindet sich genau eine 1, die die Position des Kopfes auf Band i angibt, und sonst nur 0'en. Somit erhalten wir eine Turing-Maschine Q mit einem „fetten“ Band, das alle Informationen über die k Bänder von M enthält. Diese können nun von einem „Dickkopf“ gelesen und der Überführungs-funktion von M entsprechend manipuliert werden.

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
- 3. LOOP-, WHILE-, und GOTO-Berechenbarkeit**
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem
7. Aufzählbarkeit & (Semi-)Entscheidbarkeit
8. Reduzierbarkeit
9. Satz von Rice
10. Das Postsche Korrespondenzproblem
11. Komplexität – Einführung
12. NP-Vollständigkeit
13. coNP
14. PSPACE

LOOP-, WHILE-, und GOTO-Berechenbarkeit



Quelle: commons.wikimedia.org/wiki/File:Nowitna_river.jpg

LOOP-Programme I

Programmiersprache LOOP zur Berechnung von Funktionen $\mathbb{N}^k \rightarrow \mathbb{N}$:

Variablen: x_0, x_1, \dots

Eingabe: x_1, \dots, x_k (alle anderen Variablen $x_i = 0$)

Konstanten: $0, 1, 2, \dots$

Berechnungsergebnis: Wert von x_0 am Programmende

Trennsymbole: $;$, $:=$

Operationen: $+, -$

Schlüsselwörter: **LOOP**, **DO**, **END**

Syntax	Semantik
$x_i := x_j + c$ (x_i, x_j Variablen, $c \in \mathbb{N}$)	Addition
$x_i := x_j - c$ (x_i, x_j Variablen, $c \in \mathbb{N}$)	Modifizierte Subtraktion $\max\{x_j - c, 0\}$
$P_1; P_2$ (P_1, P_2 LOOP Programme)	Sequenz erst P_1 , dann P_2
LOOP x_i DO P END (x_i Variable, P LOOP Programm)	Schleife #Durchläufe = Wert von x_i vor der Anweisung!

LOOP-Programme II

Simulation anderer Rechenoperationen durch LOOP-Programme:

$x_i := x_j$	$x_i := x_j + 0$
$x_i := c$	$x_i := x_j + c$ (für ein x_j mit $x_j = 0$)
IF $x_i = 0$ THEN P END	$x_j := 1;$ LOOP x_i DO $x_j := 0$ END ; LOOP x_j DO P END
$x_0 := x_1 + x_2$	$x_0 := x_1;$ LOOP x_2 DO $x_0 := x_0 + 1$ END

Analog: Multiplikation, ganzzahlige Division, Modulo.

Frage: Wie sehen LOOP-Programme für div & mod aus?

WHILE-Programme

WHILE-Programme \triangleq LOOP-Programme + WHILE-Schleifen

Syntax	Semantik
WHILE $x_i \neq 0$ DO P END (x_i Variable, P WHILE Programm)	While Schleife wiederhole P bis $x_i = 0$; x_i darf durch P modifiziert werden!

Berechnungsergebnis: Wert von x_0 am Programmende (falls Programm terminiert).

Definition (LOOP-/WHILE-Berechenbarkeit)

Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ heißt **LOOP-berechenbar** bzw. **WHILE-berechenbar** wenn es ein LOOP- bzw. WHILE-Programm P gibt, das f berechnet, d.h. für alle $n_1, \dots, n_k, m \in \mathbb{N}$ gilt
 $f(n_1, \dots, n_k) = m \Leftrightarrow P$ hält bei Eingabe n_1, \dots, n_k mit Berechnungsergebnis $x_0 = m$

Beobachtung: LOOP-Programme berechnen nur totale Funktionen.

Leitfrage: alle totalen intuitiv berechenbaren Funktionen über \mathbb{N} LOOP-berechenbar?

Zentraler Ansatz nachfolgend: **Simulation**.

Frage: Wie kann eine LOOP Schleife durch eine WHILE Schleife simuliert werden?

WHILE-Programme & Turing-Maschinen

Theorem

Jede WHILE-berechenbare Funktionen ist Turing-berechenbar.

Beweis (Skizze)

Bauen TM $M(P)$ durch Induktion über die „Termstruktur“ eines WHILE-Programms P :

Fall 1: $P = x_i := x_j \pm c$ \rightsquigarrow klar Turing-berechenbar (vgl. Binärzahl-Inkrementierer)

Fall 2: $P = P_1; P_2$ \rightsquigarrow Hintereinanderschaltung von $M(P_1)$ und $M(P_2)$

Identifizierte Endzustände von $M(P_1)$ mit Startzustand von $M(P_2)$.

Fall 3: $P = \text{WHILE } x_i \neq 0 \text{ DO } P_1 \text{ END}$ \rightsquigarrow Erweiterung von $M(P_1)$

1. neuer Startzustand z'_0 , der prüft ob $x_i \neq 0$

ja \rightsquigarrow Wechsel in Startzustand von M_1 .

nein \rightsquigarrow Stoppe in einem neuen Endzustand.

2. Identifizierte Endzustände von M_1 mit z'_0 .

GOTO-Programme



Quelle: pixabay.com/en/mountain-goats-jumping-leaping-1156056/

GOTO-Programme I

GOTO-Programme $\hat{=}$ Marken und Anweisungen:

$$P = M_1 : A_1;$$

$$M_2 : A_2;$$

⋮

$$M_k : A_k$$

Konvention: Nicht benutzte Marken weglassen!

Definition

GOTO-Berechenbarkeit analog zu WHILE-Berechenbarkeit.

Theorem

Jede WHILE-berechenbare Funktion ist GOTO-berechenbar.

Syntax: Mögliche Anweisungen A_i :

- ▶ $x_i := x_j \pm c$
- ▶ **GOTO** M_i
- ▶ **IF** $x_i = c$ **THEN GOTO** M_i

Semantik klar!

Beweis (simuliere WHILE $x_i \neq 0$ DO P END)

$M_1 : \text{IF } x_i = 0 \text{ THEN GOTO } M_2;$
 $P;$
GOTO $M_1;$
 $M_2 : \dots$

GOTO-Programme II

Theorem

Jede GOTO-berechenbare Funktion ist WHILE-berechenbar.

Beweis (Skizze)

$P = M_1 : A_1; \dots; M_k : A_k$ ein GOTO-Programm; ungenutzte Variable x_N

\rightsquigarrow WHILE-Programm P_W unter Benutzung des **IF-THEN**-Konstrukt:

```
 $x_N := 1;$ 
WHILE  $x_N \neq 0$  DO
  IF  $x_N = 1$  THEN  $A'_1$  END;
  IF  $x_N = 2$  THEN  $A'_2$  END;
  ...
  IF  $x_N = k$  THEN  $A'_k$  END;
  IF  $x_N = k + 1$  THEN  $x_N := 0$  END
END
```

GOTO-Anweisung A_i	\rightsquigarrow	WHILE-Anweisung A'_i
$x_j := x_i \pm c$	\rightsquigarrow	$x_j := x_i \pm c;$ $x_N := x_N + 1$
GOTO M_j	\rightsquigarrow	$x_N := j$
IF $x_j = c$ THEN GOTO M_n	\rightsquigarrow	$x_N := x_N + 1;$ IF $x_j = c$ THEN $x_N := n$

Bemerkung: Nur eine einzige WHILE-Schleife im Programm!

GOTO-Programme III

Theorem

Jede Turing-berechenbare Funktion ist GOTO-berechenbar.

Beweis (Skizze)

Sei $M = (Z, \Sigma, \Gamma, \delta, z_1, \square, E)$ mit $\Gamma = \{\square, 1, 2, \dots, m-1\}$, $Z = \{z_1, \dots, z_k\}$ eine DTM.

Idee: Darstellen der Konfiguration $\alpha z \beta$ als drei Zahlen (wobei $\square \triangleq 0$):

$x_1 = x = [\alpha]_m \rightsquigarrow$ linker Bandinhalt

$x_2 = y = [\text{rev}(\beta)]_m \rightsquigarrow$ rechter Bandinhalt

$x_3 = z = \ell \rightsquigarrow$ Zustandsnummer

Das GOTO-Programm hat nun folgende Gestalt:

Phase 1: „Erzeugung von x, y, z aus der Startkonfiguration“ (LOOP-berechenbar)

Phase 2: „Simulation von M “

Phase 3: „Rückübersetzung von y in Ausgabe x_0 “ (LOOP-berechenbar)

GOTO-Programme IV

Beweis (Fortsetzung)

Zu zeigen: GOTO-Programm für Phase 2 („Simulation von M “).

$M_2 : x_4 := y \text{ MOD } m;$

IF $z = 1$ **AND** $x_4 = 0$ **THEN GOTO** $M_{(1,0)}$;

IF $z = 1$ **AND** $x_4 = 1$ **THEN GOTO** $M_{(1,1)}$;

...

IF $z = k$ **AND** $x_4 = m - 1$ **THEN GOTO** $M_{(k,m-1)}$;

$M_{(1,0)}$: Simulation von $\delta(z_1, \square)$; **GOTO** M_2 ;

$M_{(1,1)}$: Simulation von $\delta(z_1, 1)$; **GOTO** M_2 ;

...

$M_{(k,m-1)}$: Simulation von $\delta(z_k, m - 1)$; **GOTO** M_2 ;

Simulation von δ

$\delta(z_\ell, i) = (z_j, k, L)$

$z := j;$

$y := y \text{ DIV } m;$

$y := y \cdot m + k;$

$y := y \cdot m + (x \text{ MOD } m);$

$x := x \text{ DIV } m$

$z_j \in E \rightsquigarrow \text{GOTO Phase 3}$

$\delta(z_\ell, i) = \perp$

\rightsquigarrow Endlosschleife

Fazit dieses Kapitels:

WHILE-Berechenbarkeit \equiv GOTO-Berechenbarkeit \equiv Turing-Berechenbarkeit.

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
- 4. Primitive und partielle Rekursion**
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem
7. Aufzählbarkeit & (Semi-)Entscheidbarkeit
8. Reduzierbarkeit
9. Satz von Rice
10. Das Postsche Korrespondenzproblem
11. Komplexität – Einführung
12. NP-Vollständigkeit
13. coNP
14. PSPACE

Primitive und partielle Rekursion



Quelle: commons.wikimedia.org/wiki/File:Barnsley_fern_mutated_-_Leptosporangiate_fern.png

Primitiv-rekursive Funktionen I

Definition

Die Klasse der **primitiv-rekursiven Funktionen** ist die kleinste Klasse von Funktionen die

(a) folgende **Grundfunktionen** enthält:

- i) alle konstanten Funktionen $f : \mathbb{N}^k \rightarrow \mathbb{N}$ mit $f(x_1, \dots, x_k) = c$
- ii) die Nachfolgerfunktion $\text{succ} : \mathbb{N} \rightarrow \mathbb{N}$ mit $\text{succ}(n) = n + 1$
- iii) die Projektionen $\pi_i^k : \mathbb{N}^k \rightarrow \mathbb{N}$ mit $\pi_i^k(x_1, \dots, x_k) = x_i$

(b) und abgeschlossen ist unter folgenden Operationen:

i) **Komposition** von $g_1, \dots, g_m : \mathbb{N}^k \rightarrow \mathbb{N}$ und $h : \mathbb{N}^m \rightarrow \mathbb{N}$:

$$f : \mathbb{N}^k \rightarrow \mathbb{N} \quad \text{mit} \quad f = h \circ (g_1, \dots, g_m) \text{ d.h.}$$

$$f(x_1, \dots, x_k) = h(g_1(x_1, \dots, x_k), \dots, g_m(x_1, \dots, x_k))$$

ii) **primitive Rekursion** mit $g : \mathbb{N}^k \rightarrow \mathbb{N}$ und $h : \mathbb{N}^{k+2} \rightarrow \mathbb{N}$:

$$f : \mathbb{N}^{k+1} \rightarrow \mathbb{N} \quad \text{mit} \quad f = \text{pr}(h, g) \text{ d.h.}$$

$$f(0, x_1, \dots, x_k) = g(x_1, \dots, x_k)$$

$$f(n+1, x_1, \dots, x_k) = h(n, f(n, x_1, \dots, x_k), x_1, \dots, x_k).$$

Primitiv-rekursive Funktionen II

Erinnerung: primitive Rekursion

$$f : \mathbb{N}^{k+1} \rightarrow \mathbb{N} \quad \text{mit} \quad f = \text{pr}(h, g) \text{ d.h.}$$

$$f(0, x_1, \dots, x_k) = g(x_1, \dots, x_k)$$

$$f(n+1, x_1, \dots, x_k) = h(n, f(n, x_1, \dots, x_k), x_1, \dots, x_k)$$

Bemerkung: Alle primitiv-rekursiven Funktionen sind total.

Theorem (ohne Beweis)

primitiv-rekursiv \equiv LOOP-berechenbar

Beispiel 1 – add : $\mathbb{N}^2 \rightarrow \mathbb{N}$

$$\text{add} := \text{pr}(\text{succ} \circ \pi_2^3, \pi_1^1) \text{ d.h.}$$

$$\text{add}(0, x) = \pi_1^1(x) = x$$

$$\begin{aligned}\text{add}(n+1, x) &= (\text{succ} \circ \pi_2^3)(n, \text{add}(n, x), x) \\ &= \text{succ}(\text{add}(n, x))\end{aligned}$$

Beispiel 2 – mul : $\mathbb{N}^2 \rightarrow \mathbb{N}$

$$\text{mul} := \text{pr}(\text{add} \circ (\pi_2^3, \pi_3^3), 0_1) \text{ d.h.}$$

$$\text{mul}(0, x) = 0_1(x) = 0$$

$$\begin{aligned}\text{mul}(n+1, x) &= (\text{add} \circ (\pi_2^3, \pi_3^3))(n, \text{mul}(n, x), x) \\ &= \text{add}(\text{mul}(n, x), x)\end{aligned}$$

Frage: Konstruieren Sie mittels primitiver Rekursion: (1) modifizierte Vorgängerfunktion $f(x) = \max\{x - 1, 0\}$, (2) modifizierte Subtraktion $f(x, y) = \max\{x - y, 0\}$, (3) Fakultätsfunktion $f(x) = x!$

μ -rekursive Funktionen I

Definition (μ - bzw. partielle Rekursion)

Die Klasse der μ -rekursiven Funktionen ist die kleinste Klasse von Funktionen die

- a) die Grundfunktionen enthält und
- b) abgeschlossen ist unter folgenden Operationen:

- i) Komposition,
- ii) primitive Rekursion und
- iii) **μ -Operator** von $g : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$:

$$f : \mathbb{N}^k \rightarrow \mathbb{N} \quad \text{mit} \quad f = \mu(g) \text{ d.h.}$$

$$f(x_1, \dots, x_k) := \min \{n \mid g(n, x_1, \dots, x_k) = 0 \wedge \forall_{0 \leq i < n} g(i, x_1, \dots, x_k) \neq \perp\}$$

$\leadsto \mu(g)$ womöglich nicht total!

Frage: Welche uns bekannte Funktion verbirgt sich hinter $\mu(1_2)$?

Theorem (ohne Beweis)

μ -rekursiv \equiv

Turing/WHILE/GOTO-berechenbar

Theorem (Kleene'sche Normalform, ohne Beweis)

Zu jeder μ -rekursiven Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ gibt es 2 primitiv-rekursive Funktionen g und h sodass

$$f(x_1, \dots, x_k) = g(x_1, \dots, x_k, \mu(h)(x_1, \dots, x_k)).$$

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
- 5. Grenzen der LOOP-Berechenbarkeit**
6. (Un-)Entscheidbarkeit, Halteproblem
7. Aufzählbarkeit & (Semi-)Entscheidbarkeit
8. Reduzierbarkeit
9. Satz von Rice
10. Das Postsche Korrespondenzproblem
11. Komplexität – Einführung
12. NP-Vollständigkeit
13. coNP
14. PSPACE

Grenzen der LOOP-Berechenbarkeit

Wissen: alle LOOP-berechenbaren Funktionen sind total

Frage: gibt es totale Funktionen die nicht LOOP-berechenbar sind? Ja! (Diagonalisierung)

Theorem

Sei L eine Liste aller 1-stelligen, LOOP-berechenbaren Funktionen. Dann ist $g : \mathbb{N} \rightarrow \mathbb{N}$ mit
$$g(n) := L_n(n) + 1$$

total und nicht LOOP-berechenbar.

Beweis

Wäre g LOOP-berechenbar, so gäbe es ein $k \in \mathbb{N}$ mit $L_k = g$.

$$\leadsto g(k) = L_k(k) + 1 = g(k) + 1$$

Aber: Ist g (Turing-)berechenbar?

Ackermannfunktion I

Die Ackermannfunktion (Variante Rósza Péter):

$$\text{ack}(0, y) := y + 1,$$

$$\text{ack}(x, 0) := \text{ack}(x - 1, 1),$$

$$\text{ack}(x, y) := \text{ack}(x - 1, \text{ack}(x, y - 1))$$

$$= \underbrace{\text{ack}(x - 1, \text{ack}(x - 1, \text{ack}(x - 1, \dots, \text{ack}(x - 1, 1)))) \dots}_{(y+1) \text{ mal}}$$

Die Ackermannfunktion wächst extrem schnell (z.B. gilt $\text{ack}(4, 2) \approx 2 \cdot 10^{19728}$).

Eine „modernisierte“ Variante:

$$a(0, y) := 1,$$

$$a(1, y) := 3y + 1,$$

$$a(x, y) := \underbrace{a(x - 1, a(x - 1, \dots, a(x - 1, y) \dots))}_{y \text{ mal}}$$

Beobachtung: a ist total und in beiden Argumenten monoton wachsend

Frage: können Sie zeigen, dass $a(x, y) \leq \text{ack}(x, 3y)$? (*)

Ackermannfunktion II

Theorem

Die Ackermannfunktion a ist nicht LOOP-berechenbar.

Strategie: zeigen, dass a schneller wächst als jede LOOP-berechenbare Funktion.

↪ definieren zunächst $f_P(n)$ als die maximale Summe aller Variablenendwerte, die das Programm P erzeugen kann, wenn die initiale Belegung höchstens Summe n hat.

Definition

Sei P ein LOOP-Programm, welches die Variablen x_0, x_1, \dots, x_k verwendet.

Die i te Speicherüberführungsfunktion $F_i^P : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ an der Stelle (n_0, n_1, \dots, n_k) ist der Wert von x_i am Ende des Programmes P falls P mit $x_j = n_j$ für alle $0 \leq j \leq k$ gestartet wird.
Außerdem sei die Funktion $f_P : \mathbb{N} \rightarrow \mathbb{N}$ definiert als

$$f_P(n) := \max \left\{ \sum_{i=0}^k F_i^P(n_0, \dots, n_k) \mid \sum_{i=0}^k n_i \leq n \right\}.$$

Ackermannfunktion III

Lemma

Zu jedem LOOP-Programm P existiert ein $\ell \in \mathbb{N}$ sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$.

Beweis (Induktion über Termstruktur von P)

Fall 1: $P = „x_i := x_j \pm c“$

$\rightsquigarrow f_P(n) \leq 2n + c \leq 3n < 3n + 1 = a(1, n)$. \rightsquigarrow Wähle $\ell := \max\{c, 1\}$.

Fall 2: $P = „P_1; P_2“$

Induktionsvoraussetzung \rightsquigarrow es gibt $\ell_1, \ell_2 \in \mathbb{N}$, sodass für alle $n \geq \max\{\ell_1, \ell_2\} =: \ell_3$ gilt:
 $f_{P_1}(n) < a(\ell_1, n) \leq a(\ell_3, n)$ und $f_{P_2}(n) < a(\ell_2, n) \leq a(\ell_3, n)$.

Da $f_P(n) \leq f_{P_2}(f_{P_1}(n))$ folgt (falls $\ell_3 \geq 2$):

$f_P(n) < a(\ell_3, f_{P_1}(n)) \leq a(\ell_3, a(\ell_3, n)) \leq \underbrace{a(\ell_3, a(\ell_3, \dots, a(\ell_3, n) \dots))}_{n-\text{mal}} = a(\ell_3 + 1, n)$.

\rightsquigarrow Wähle $\ell := \max\{\ell_3 + 1, 2\}$.

Ackermannfunktion III

Lemma

Zu jedem LOOP-Programm P existiert ein $\ell \in \mathbb{N}$ sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$.

Beweis (Induktion über Termstruktur von P)

Fall 3: $P = \text{„LOOP } x_i \text{ DO } P' \text{ END“}$

Induktionsvoraussetzung \rightsquigarrow es gibt $\ell' \in \mathbb{N}$, sodass für alle $n \geq \ell'$ gilt: $f_{P'}(n) < a(\ell', n)$.

Dann gilt $f_P(n) \leq (\underbrace{f_{P'} \circ \dots \circ f_{P'}}_{n \text{ mal}})(n) < \underbrace{a(\ell', a(\ell', \dots, a(\ell', n) \dots)}_{n \text{ mal}} = a(\ell' + 1, n)$.

\rightsquigarrow Wähle $\ell := \ell' + 1$.

Ackermannfunktion IV

Lemma

Zu jedem LOOP-Programm P existiert ein $\ell \in \mathbb{N}$ sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$.

Theorem

Die Ackermannfunktion a ist nicht LOOP-berechenbar.

Beweis

Annahme: a LOOP-berechenbar.

$\rightsquigarrow g(n) := a(n, n)$ LOOP-berechenbar vermöge LOOP-Programm P .

\rightsquigarrow es gibt ein $\ell \in \mathbb{N}$, sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$.

$\rightsquigarrow g(\ell) \leq f_P(\ell) < a(\ell, \ell) = g(\ell)$.

Bemerkung: Funktion g im Beweis wächst schneller als jede LOOP-berechenbare Funktion

Ein WHILE-Programm für die Ackermannfunktion

Theorem

a ist WHILE-berechenbar.

$$a(0, y) := 1,$$

$$a(1, y) := 3y + 1,$$

$$a(x, y) := \underbrace{a(x - 1, a(x - 1, \dots, a(x - 1, y) \dots))}_{y \text{ mal}}$$

Beweis

Idee: Rekursion in WHILE-Schleife pressen

Schwierigkeit: Unbeschränkte Rekursionstiefe mit beschränkter Anzahl Variablen!

↪ speichern mehrere Zahlen in einer Variable.

↪ injektive, LOOP-berechenbare „Pairing-Funktion“, z.B. $c(x, y) := 2^{x+y} + x$

(Umkehrfunktionen $\text{first}(c(x, y)) := x$ und $\text{second}(c(x, y)) := y$ LOOP-berechenbar)

↪ Kellerinhalt n_1, n_2, \dots, n_k in Zahl $n := c(n_1, c(n_2, \dots, c(n_k, 0) \dots))$ gespeichert

INIT ↪ $n := 0$

PUSH(x) ↪ $n := c(x, n)$

POP ↪ $x := \text{first}(n); n := \text{second}(n); \text{return } x$

Ein WHILE-Programm für die Ackermannfunktion

Theorem

a ist WHILE-berechenbar.

Beweis

```
1 INIT; PUSH(x); PUSH(y);
2 WHILE STACK SIZE > 1 DO // second(n) ≠ 0
3   y ← POP;
4   x ← POP;
5   IF x = 0 THEN
6     | PUSH(1)
7   ELSEIF x = 1 THEN
8     | PUSH(3 · y + 1)
9   ELSE
10    | LOOP y DO PUSH(x - 1) END;
11    | PUSH(y)
12 END
13 x0 ← POP;
```

$$a(0, y) := 1,$$
$$a(1, y) := 3y + 1,$$
$$a(x, y) := \underbrace{a(x - 1, a(x - 1, \dots, a(x - 1, y) \dots))}_{y \text{ mal}}$$

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
- 6. (Un-)Entscheidbarkeit, Halteproblem**
7. Aufzählbarkeit & (Semi-)Entscheidbarkeit
8. Reduzierbarkeit
9. Satz von Rice
10. Das Postsche Korrespondenzproblem
11. Komplexität – Einführung
12. NP-Vollständigkeit
13. coNP
14. PSPACE

Kodierung von Turing-Maschinen

Kodierung von Turing-Maschinen als Wort über $\{0, 1, \#\}$:

Sei $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, \{z_e\})$ mit

$$Z = \{z_0, z_1, \dots, z_n = z_e\} \quad \Sigma = \{0, 1\} \quad \Gamma = \{a_0 = \square, a_1, \dots, a_k\}$$

beschreibe jede Transition $\delta(z_i, a_j) = (z_{i'}, a_{j'}, y)$ als Wort über $\{0, 1, \#\}$:

$$w_{i,j,i',j',y} := \# \# \text{BIN}(i) \# \text{BIN}(j) \# \text{BIN}(i') \# \text{BIN}(j') \# \text{BIN}(m) \text{ mit } m := \begin{cases} 0, & y = L \\ 1, & y = R \\ 2, & y = N. \end{cases}$$

↪ beschreibe M als beliebige Konkatenation aller ihrer “Übergangswörter” $w_{i,j,i',j',y}$.

Kodierung von $\{0, 1, \#\}$ mit $\{0, 1\}$ (zum Beispiel durch $0 \rightarrow 00$, $1 \rightarrow 01$, $\# \rightarrow 11$).

↪ Kodierung von M ist $\langle M \rangle \in \{0, 1\}^*$.

↪ Kodierung umkehrbar aber nicht alle Wörter über $\{0, 1\}^*$ kodieren eine Turing-Maschine.

$$M_w := \begin{cases} M & \text{falls } w = \langle M \rangle \\ M_\Omega & \text{sonst} \end{cases}$$

$w \in \{0, 1\}^*$ keine valide Kodierung
↪ feste Maschine M_Ω , die die nigends definierte Funktion berechnet

Spezielles Halteproblem

Definition

Das **spezielle Halteproblem** ist die Sprache

$$K := \{w \in \{0,1\}^* \mid M_w \text{ hält auf Eingabe } w\},$$

Theorem

Das spezielle Halteproblem $K = \{w \in \{0,1\}^* \mid M_w \text{ hält auf Eingabe } w\}$ ist unentscheidbar.

Beweis (durch Widerspruch)

Annahme: K entscheidbar \rightsquigarrow charakteristische Funktion χ_K berechenbar durch TM M .

Erweitere M zu M' , sodass M' genau dann hält, wenn M eine 0 ausgibt.

Sei $w' := \langle M' \rangle$, d.h. $M' = M_{w'}$.

$\rightsquigarrow M'$ hält bei Eingabe w'

$\Leftrightarrow M$ gibt bei Eingabe w' eine 0 aus

$\Leftrightarrow \chi_K(w') = 0$

$\Leftrightarrow w' \notin K$

$\Leftrightarrow M'$ hält nicht bei Eingabe $\langle M' \rangle = w'$. ↯

Fleißige Bieber

TM-Wettbewerb: Maschinen mit n Zuständen konkurrieren, wer läuft am längsten (Maschinen die nicht halten sind disqualifiziert)
~ "fleißige Bieber"

Definition (Fleißige Bieber, [Radó '62])

Für jedes Codewort w einer Turing-Maschine M , sei

$s(w)$ = Anzahl Schritte die M (bei leerer Eingabe) macht (0 falls M nicht hält) und

$e(w)$ = Anzahl (nicht- \square) Symbole auf dem Band wenn M (bei leerer Eingabe) hält (sonst 0).

Für jedes $n \in \mathbb{N}$ sei $\mathcal{M}_1(n)$ die Menge aller Turing-Maschinen $M = (Z, \Sigma, \Gamma, \delta, \square, z_0, E)$ mit

$$|Z| = n \quad \Sigma = \{1\} \quad \Gamma = \Sigma \cup \{\square\}.$$

Wir definieren die Funktionen

$$S_1(n) := \max_{M \in \mathcal{M}_1(n)} s(\langle M \rangle) \quad E_1(n) := \max_{M \in \mathcal{M}_1(n)} e(\langle M \rangle)$$

Eine Maschine $M \in \mathcal{M}_1(n)$ heißt **unärer fleißiger Bieber (busy beaver)** falls M auf leerer Eingabe $E_1(n)$ nicht- \square Symbole auf das Band schreibt und dann hält.

Fleißige Bieber sind Unberechenbar I

Theorem

Die Funktion S ist nicht Turing-berechenbar.

Beweis

Annahme: S ist Turing-berechenbar

~ $\sim S$ wird von einer Turing-Maschine berechnet.

~ \sim es gibt $n \in \mathbb{N}$ sodass S von einer Turing-Maschine $M_{BB} \in \mathcal{M}(n)$ berechnet wird.

Sei M' die Maschine, die wie folgt agiert:

1. verdopple die Zahl auf dem Band
2. simuliere M_{BB}
3. wandle das binäre Ausgabewort in unär

Sei M'' die Maschine, die wie folgt agiert:

1. erzeuge die Zahl n' auf dem Band
 2. simuliere M'
- sodass M'' genau $2n'$ Zustände hat.

Sei n' die Anzahl Zustände von M' .

~ $\sim M''$ hat $2n'$ Zustände aber M'' macht bei leerer Eingabe **echt mehr** als $S(2n')$ Schritte ↴

Frage: Überlegen Sie sich den **analogen** Beweis dafür, dass E nicht Turing-berechenbar ist.

Fleißige Bieber sind Unberechenbar II

Theorem

Die Funktion S wächst (asymptotisch) schneller als jede berechenbare Funktion!

Beweis (Skizze)

Annahme: es gibt berechenbare Funktion f und $n_0 \in \mathbb{N}$, sodass $f(n) > S(n)$ für alle $n > n_0$.

↪ bauen Turing-Maschine M , die entscheidet ob $w \in H_0$ für gegebene Codierung w einer TM mit $\Sigma = \{0, 1\}$ und $\Gamma = \{0, 1, \square\}$ und $n := |Z|$.

1. $n \leq n_0$, dann gib fest verdrahtete Antwort aus (endlich viele)
2. berechne $f(n)$
3. simuliere M_w auf leerem Band höchstens $f(n)$ Schritte
4. gib aus, ob M_w nach höchstens $f(n)$ Schritten hält

↪ die von M berechnete Funktion ist total und es gilt:

$w \in H_0 \Leftrightarrow M_w$ hält auf leerem Band

$\Leftrightarrow M_w$ hält auf leerem Band nach höchstens $f(n)$ Schritten $\Leftrightarrow M$ gibt 1 aus

↪ M berechnet χ_{H_0} ↳

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem
- 7. Aufzählbarkeit & (Semi-)Entscheidbarkeit**
8. Reduzierbarkeit
9. Satz von Rice
10. Das Postsche Korrespondenzproblem
11. Komplexität – Einführung
12. NP-Vollständigkeit
13. coNP
14. PSPACE

(Semi-)Entscheidbarkeit und Aufzählbarkeit

Definition (Erinnerung)

- Eine Sprache $A \subseteq \Sigma^*$ heißt **entscheidbar**, falls die charakteristische Funktion $\chi_A : \Sigma^* \rightarrow \{0, 1\}$ berechenbar ist.
- Eine Sprache $A \subseteq \Sigma^*$ heißt **semi-entscheidbar**, falls die halbe charakteristische Funktion $\chi'_A : \Sigma^* \rightarrow \{0, 1\}$ berechenbar ist.

$$\chi_A(x) := \begin{cases} 1, & \text{falls } x \in A, \\ 0, & \text{falls } x \notin A. \end{cases}$$

$$\chi'_A(x) := \begin{cases} 1, & \text{falls } x \in A, \\ \perp, & \text{falls } x \notin A. \end{cases}$$

Definition

Eine Sprache $A \subseteq \Sigma^*$ heißt **(rekursiv) aufzählbar**, falls $A = \emptyset$ gilt oder falls es eine **totale, berechenbare** Funktion $f : \mathbb{N} \rightarrow \Sigma^*$ derart gibt, dass $A = \{f(0), f(1), f(2), \dots\} = f(\mathbb{N})$.

Das heißt, f zählt A auf.

Beachte: f muss weder injektiv noch monoton sein!

Frage: Können Sie ein f angeben, das die Sprache $\{w \in \{0, 1\}^* \mid w \text{ ist Binärkodierung einer Primzahl}\}$ aufzählt?

Entscheidbare und Semi-Entscheidbare Sprachen

Theorem

$A \subseteq \Sigma^*$ ist genau dann entscheidbar, wenn sowohl A als auch $\Sigma^* \setminus A$ semi-entscheidbar ist.

Beweis

„ \Rightarrow “: A entscheidbar \Rightarrow 1. A semi-entscheidbar.
2. $\Sigma^* \setminus A$ entscheidbar $\Rightarrow \Sigma^* \setminus A$ semi-entscheidbar.

„ \Leftarrow “: x'_A und $x'_{\Sigma^* \setminus A}$ berechenbar durch
WHILE-Programme mit einer WHILE-Schleife (& disjunkten Variablennamen):

$x_i := 1$; WHILE $x_i \neq 0$ DO P_A END; $x_0 := 1$

sowie

$x_j := 1$; WHILE $x_j \neq 0$ DO $P_{\bar{A}}$ END; $x_0 := 1$

Dann entscheidet folgendes Programm A :

1 $x_i := 1$; $x_j := 1$;
2 WHILE $x_i \neq 0$ und $x_j \neq 0$ DO
3 | P_A ; $P_{\bar{A}}$;
4 END
5 IF $x_i = 0$ THEN $x_0 := 1$ ELSE $x_0 := 0$;

(Rekursiv) Aufzählbare Sprachen

Theorem

Eine Sprache A ist aufzählbar
gdw.

A is semi-entscheidbar.

Beachte: Wir nehmen an, dass $\chi'_A : \mathbb{N} \rightarrow \{0, 1\}$
(Bijektion zwischen \mathbb{N} & Σ^* berechenbar)

Beweis

„ \Rightarrow “: $f(\mathbb{N}) = A$ total & berechenbar

$\sim \chi'_A$ berechnet durch

1 $x_2 := 0;$

2 **WHILE** $x_0 \neq 1$ **DO**

3 | **IF** $f(x_2) = x_1$ **THEN** $x_0 := 1;$

4 | $x_2 := x_2 + 1;$

5 **END**

„ \Leftarrow “ (Skizze):

Konstruiere Algorithmus der eine totale Funktion f berechnet die A aufzählt:

Versuch 3: In Schritt i des Algorithmus für f , simuliere Algorithmus für $\chi'_A(j)$ für jedes $j \leq i$ genau einen Schritt, bis $n+1$ „Erfolge“ ($\chi'_A(j) = 1$) beobachtet wurden und gebe das letzte erfolgreiche w_j aus.

Zusammenfassung (Semi-) Entscheidbarkeit

Für beliebige Sprachen $A \subseteq \Sigma^*$ gelten folgende Äquivalenzen:

A ist semi-entscheidbar

$\Leftrightarrow \chi'_A$ ist berechenbar

$\Leftrightarrow A$ ist aufzählbar

$\Leftrightarrow A = T(M)$ wird von einer Turing-Maschine M akzeptiert

$\Leftrightarrow A$ ist Definitionsbereich einer (partiellen) berechenbaren Funktion $f : \Sigma^* \rightarrow \Pi^*$
(A lässt sich schreiben als $A = f^{-1}(\Pi^*)$)

$\Leftrightarrow A$ ist Wertebereich einer (partiellen) berechenbaren Funktion $g : \Pi^* \rightarrow \Sigma^*$
(A lässt sich schreiben als $A = g(\Pi^*)$)

$\Leftrightarrow A$ ist Typ 0-Sprache (Chomsky-Hierarchie)

A ist entscheidbar

$\Leftrightarrow \chi_A$ ist berechenbar

$\Leftrightarrow A$ endlich oder aufzählbar durch totale, berechenbare, **streng monotone** Funktion

$\Leftrightarrow A = T(M)$ wird von einer Turing-Maschine M akzeptiert die **auf allen Eingaben hält**

$\Leftrightarrow A$ ist Urbild eines Bildwertes einer **totalen**, berechenbaren Funktion $f : \Sigma^* \rightarrow \Pi^*$
(A lässt sich schreiben als $A = f^{-1}(1)$, mit $1 \in \Pi^*$)

$\Leftrightarrow A$ ist Wertebereich einer **totalen**, berechenbaren, **streng monotonen** Funktion $g : \Pi^* \rightarrow \Sigma^*$
(A lässt sich schreiben als $A = g(\Pi^*)$)

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem
7. Aufzählbarkeit & (Semi-)Entscheidbarkeit
- 8. Reduzierbarkeit**
9. Satz von Rice
10. Das Postsche Korrespondenzproblem
11. Komplexität – Einführung
12. NP-Vollständigkeit
13. coNP
14. PSPACE

Reduzierbarkeit I

Definition

Das **allgemeine Halteproblem** ist die Menge $H := \{w\#x \mid M_w \text{ hält auf Eingabe } x\}$

Erinnerung: spezielles Halteproblem $K := \{w \in \{0,1\}^* \mid M_w \text{ hält auf } w\}$ unentscheidbar

Informell: keine TM kann feststellen, ob die eingegebene TM auf ihrem Codewort hält oder nicht

Klar: H ist Generalisierung von K

Informell: H ist sicher nicht leichter zu entscheiden als $K \rightsquigarrow H$ ist unentscheidbar!

Formell:

zentrales Konzept der Reduktion!

Reduzierbarkeit II

Definition

Das **allgemeine Halteproblem** ist die Menge $H := \{w\#x \mid M_w \text{ hält auf Eingabe } x\}$.

Definition

Eine Sprache $A \subseteq \Sigma^*$ heißt **reduzierbar auf** eine Sprache $B \subseteq \Pi^*$ (**in Zeichen** $A \leq B$), wenn es eine totale, berechenbare Funktion $f: \Sigma^* \rightarrow \Pi^*$ gibt, sodass für alle $x \in \Sigma^*$ gilt

$$x \in A \Leftrightarrow f(x) \in B.$$

Wir nennen f eine **Reduktion** von A auf B (**Beachte:** f muss weder surjektiv noch injektiv sein).

$A \leq B$ formalisiert die Intuition “ A ist **leichter** als B ” d.h.

“wenn wir B entscheiden könnten, dann könnten wir auch A entscheiden”

Beispiel

$K \leq H$ wird vermittelt durch die Reduktion $f: \{0, 1\}^* \rightarrow \{0, 1, \#\}^*$ mit $f(w) = w\#w$.

Frage: Ist eine Sprache L entscheidbar, so ist χ_L eine Reduktion von L auf welche Sprache?

Reduzierbarkeit III

Lemma

$A \leq B$ genau dann, wenn $\overline{A} \leq \overline{B}$ (wobei $\overline{A} = \text{Co-}A$).

Lemma

Gilt $A \leq B$ und ist B (semi-)entscheidbar, so ist auch A (semi-)entscheidbar.

Beweis

Sei f eine Reduktion von A auf B (d.h. f total, berechenbar mit $x \in A \Leftrightarrow f(x) \in B$).

Dann gilt $\chi'_A = \chi'_B \circ f$, denn

$$x \in A \Rightarrow (\chi'_B \circ f)(x) = \chi'_B(f(x)) = 1$$

$$x \notin A \Rightarrow (\chi'_B \circ f)(x) = \chi'_B(f(x)) = \perp$$

Ist also χ_B (bzw. χ'_B) berechenbar, so auch χ_A (bzw. χ'_A).

Wortproblem für Turing-Maschinen

Lemma

Für die Sprache $U := \{w\#x \mid x \in T(M_w)\}$ gilt: $U \leq H$ und $H \leq U$.

Beweis

Konstruktion einer Reduktion f

$H \leq U$: bei Eingabe w berechnet f das Codewort einer Maschine M' , die wie M_w arbeitet, aber in einen Endzustand übergeht, sobald M_w hält (egal ob akzeptierend oder ablehnend).

$U \leq H$: bei Eingabe w berechnet f das Codewort einer Maschine M' , die wie M_w arbeitet, aber in eine Endlosschleife geht, wenn M_w in einem Nicht-Endzustand hält.

Fazit: H und U im Berechenbarkeitssinne „äquivalent“ (U unentscheidbar da $K \leq H \leq U$)

Halteproblem auf leerem Band

Definition

Das Halteproblem auf leerem Band ist $H_0 := \{w \mid w\# \in H\}$.

Theorem

H_0 ist unentscheidbar.

Beweis

Wir zeigen $H \leq H_0$ ("H leichter als H_0 ") durch Konstruktion einer Reduktion f von H auf H_0 .

Erinnerung: $H = \{w\#x \mid M_w \text{ hält auf Eingabe } x\}$.

Bei Eingabe $w\#x$ berechnet f das Codewort einer Maschine M' , die zunächst das Wort x auf dem Band erzeugt und dann wie M_w arbeitet ($\sim M_w \text{ hält auf } x \Leftrightarrow M' \text{ hält auf } \epsilon$).

(bei allen anderen Eingaben über $\{0, 1, \#\}$ gibt f eine ungültige Kodierung aus, z.B. 0)

Es gilt für alle Wörter $q \in \{0, 1, \#\}^*$:

Falls $q = w\#x$ für $w, x \in \{0, 1\}^*$, dann

$$w\#x \in H \Leftrightarrow M_w \text{ hält auf } x$$

$$\Leftrightarrow M' \text{ hält auf } \epsilon \Leftrightarrow f(w\#x) \in H_0$$

Sonst: $q \notin H$ und $f(q) \notin H_0$. Fazit: $H \leq H_0$.

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem
7. Aufzählbarkeit & (Semi-)Entscheidbarkeit
8. Reduzierbarkeit
- 9. Satz von Rice**
10. Das Postsche Korrespondenzproblem
11. Komplexität – Einführung
12. NP-Vollständigkeit
13. coNP
14. PSPACE

Satz von Rice I

Informell: Fragen bzgl. Leistungsfähigkeit/Verhalten einer TM **generell unentscheidbar**

Theorem (Satz von Rice)

Sei \mathcal{R} die Menge aller Turing-berechenbaren Funktionen.

Sei $\mathcal{S} \subseteq \mathcal{R}$ eine **nicht-triviale** Teilmenge von \mathcal{R} (d.h. $\mathcal{S} \neq \emptyset$ und $\mathcal{S} \neq \mathcal{R}$).

Dann ist $\mathcal{C}(\mathcal{S}) := \{w \mid \text{die von } M_w \text{ berechnete Funktion liegt in } \mathcal{S}\}$ unentscheidbar.

Beweis

Fall 1: Die überall undefinierte Funktion Ω ist nicht in \mathcal{S} . Wir zeigen $H_0 \leq \mathcal{C}(\mathcal{S})$.

Da $\mathcal{S} \neq \emptyset$, existiert eine Turingmaschine Q , deren berechnete Funktion q in \mathcal{S} liegt.

Wir konstruieren Reduktion $f : \{0,1\}^* \rightarrow \{0,1\}^*$ die bei Eingabe eines Codewortes w das Codewort w' der Maschine M' ausgibt die bei Eingabe x

1. erst M_w auf leerem Band simuliert, und $w \in H_0 \Leftrightarrow M_w \text{ hält auf leerem Band}$
2. nachdem M_w hält, Q auf Eingabe x simuliert. $\Leftrightarrow M' \text{ berechnet } q \in \mathcal{S}$

$$\leadsto M' \text{ berechnet } \begin{cases} q & \text{falls } w \in H_0 \\ \Omega & \text{sonst} \end{cases} \quad \Leftrightarrow \langle M' \rangle \in \mathcal{C}(\mathcal{S})$$

Frage: Überlegen Sie sich den **analogen** Fall 2 des Beweises (Reduktion von $\overline{H_0}$ auf $\mathcal{C}(\mathcal{S})$)

Satz von Rice II

Korollar

Folgende Fragestellungen bzgl. der Leistung von Turing-Maschinen sind unentscheidbar:

(a) Die berechnete Funktion ist

- ▶ konstant,
- ▶ total,
- ▶ primitiv-rekursiv.

(b) Die akzeptierte Sprache ist

- ▶ leer,
- ▶ endlich,
- ▶ Σ^* ,
- ▶ regulär,
- ▶ kontextfrei,
- ▶ kontextsensitiv.

Abschließende Bemerkungen / Mitteilungen:

1. "ist die akzeptierte Sprache vom Typ-0?" ist trivial (immer "ja")
2. "zweiter Satz von Rice" charakterisiert semi-entscheidbare Teilmengen $\mathcal{S} \subseteq \mathcal{R}$:
zB. $\mathcal{C}(\{q \mid q \neq \Omega\})$ semi-entscheidbar, aber $\mathcal{C}(\{\Omega\})$ nicht
3. Es gibt nachweislich schwierigere Probleme als das allgemeine Halteproblem:
zB. das "Äquivalenzproblem für Turing-Maschinen" $Eq := \{w \# w' \mid T(M_w) = T(M_{w'})\}$
 $H \leq Eq$ aber **nicht** $Eq \leq H$

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem
7. Aufzählbarkeit & (Semi-)Entscheidbarkeit
8. Reduzierbarkeit
9. Satz von Rice
- 10. Das Postsche Korrespondenzproblem**
11. Komplexität – Einführung
12. NP-Vollständigkeit
13. coNP
14. PSPACE

Das Postsche Korrespondenzproblem I

Definition

Für ein endliches Alphabet Σ ist das Postsche Korrespondenzproblem die Menge

$$\text{PCP} := \{ \langle (x_1, y_1), (x_2, y_2), \dots, (x_k, y_k) \rangle \in (\Sigma^* \times \Sigma^*)^k \mid \exists_{n \geq 1} \exists_{i_1, i_2, \dots, i_n \in \{1, 2, \dots, k\}} : \\ x_{i_1} \cdot x_{i_2} \cdot \dots \cdot x_{i_n} = y_{i_1} \cdot y_{i_2} \cdot \dots \cdot y_{i_n} \}$$

Beispiel 1

$$\left(\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 101 \end{pmatrix}, \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} 10 \\ 00 \end{pmatrix}, \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} = \begin{pmatrix} 011 \\ 11 \end{pmatrix} \right) \in \text{PCP}$$

Wähle $i_1 = 1, i_2 = 3, i_3 = 2, i_4 = 3$.

$$x_1 \cdot x_3 \cdot x_2 \cdot x_3 = 1 \cdot 011 \cdot 10 \cdot 011 = 101110011$$

$$y_1 \cdot y_3 \cdot y_2 \cdot y_3 = 101 \cdot 11 \cdot 00 \cdot 11 = 101110011$$

Beispiel 2

$$\left(\begin{pmatrix} 1 \\ 10 \end{pmatrix}, \begin{pmatrix} 10 \\ 1 \end{pmatrix}, \begin{pmatrix} 01 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 001 \end{pmatrix} \right) \notin \text{PCP}$$

„Suffixfreiheit“: jedes x_i endet mit einem anderen Zeichen als y_i

Das Postsche Korrespondenzproblem II

Hinweise:

1. PCP oft in Unentscheidbarkeitsbeweisen (Reduktionen) benutzt
2. PCP semi-entscheidbar für beliebiges Σ (vermöge Brute-Force-Algorithmus).
3. „unäres PCP“ ($|\Sigma| = 1$) entscheidbar:

Beweisidee: Es kommt nur darauf an, dass

$$\sum_{j \leq n} |x_{ij}| = \sum_{j \leq n} |y_{ij}| \quad \text{also} \quad \sum_{j \leq n} (|x_{ij}| - |y_{ij}|) = 0.$$

↪ unäres PCP äquivalent zur Frage:

“lassen sich k gegebene ganze Zahlen $(|x_i| - |y_i|) \in \mathbb{Z}$ nichttrivial linear zu 0 kombinieren?”

↪ genau dann unmöglich wenn alle Zahlen positiv oder alle negativ

4. PCP entscheidbar für $k = 2$ Eingabepaare, aber unentscheidbar für $k \geq 4$ Eingabepaare.
5. PCP entscheidbar falls nur nach einer Lösung mit Länge $\leq n$ gesucht
6. Für beliebiges Σ lässt sich PCP auf PCP mit $\Sigma' = \{0, 1\}$ zurückführen

Das Modifizierte PCP

Folgende Variante MPCP (M wie „Modified“) sehr nützlich

$$\text{MPCP} := \left\{ \left\langle \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, \begin{pmatrix} x_2 \\ y_2 \end{pmatrix}, \dots, \begin{pmatrix} x_k \\ y_k \end{pmatrix} \right\rangle \in \text{PCP} \mid \exists_{n \geq 1} \exists_{i_2, \dots, i_n \in \{1, 2, \dots, k\}} : x_1 \cdot x_{i_2} \cdots x_{i_n} = y_1 \cdot y_{i_2} \cdots y_{i_n} \right\}$$

Lemma

$$\text{MPCP} \leq \text{PCP}.$$

Verwende neue Symbole $\$, \# \notin \Sigma$

Definiere für $a \in \Sigma$, $w \in \Sigma^*$:

$$(aw)^{\text{li}} = \#aw^{\text{li}} \quad (aw)^{\text{re}} = a\#w^{\text{re}} \quad \varepsilon^{\text{li}} = \varepsilon^{\text{re}} = \varepsilon$$

Noch zu zeigen: f ist Reduktion, also $P \in \text{MPCP} \Leftrightarrow f(P) \in \text{PCP}$.

“ \Rightarrow ”: $(1, i_2, \dots, i_n)$ Lösung für $P \Rightarrow (k+1, i_2, \dots, i_n, k+2)$ Lösung für $f(P)$

“ \Leftarrow ”: $(k+1, i_2, \dots, i_n, k+2)$ Lösung für $f(P)$

oBdA. $i_m \neq k+2$, sonst $(k+1, i_2, \dots, i_m)$ auch Lösung $\rightsquigarrow (1, i_2, \dots, i_n)$ Lösung für P .

Reduktion f

$$\begin{pmatrix} x_j \\ y_j \end{pmatrix} \mapsto \begin{pmatrix} x_j^{\text{re}} \\ y_j^{\text{li}} \end{pmatrix}$$

Sowie neue Paare

$$\begin{pmatrix} \#x_1^{\text{re}} \\ y_1^{\text{li}} \end{pmatrix} \begin{pmatrix} \$ \\ \#\$ \end{pmatrix}$$

H_0 reduzierbar auf MPCP

Beweis (Skizze)

Reduktion f erhält das Codewortes von $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, \{z_e\})$ und erzeugt MPCP-Instanz.

oBdA: Eingabemaschine M hält $\Leftrightarrow M$ hält in akzeptierendem Zustand

Zeigen $\langle M \rangle \in H_0 \Leftrightarrow f(\langle M \rangle) \in \text{MPCP}$

“ \Rightarrow ”: M hält auf leerem Band

\rightsquigarrow es gibt Konfigurationenfolge

$$\square z_0 \square \vdash_M^* \alpha z_e \beta \text{ mit } \alpha, \beta \in \Gamma^*.$$

Simulation durch MPCP wie im Beispiel,

am Ende β entfernt durch Löschregel,

Abschluss durch Abschlussregel

“ \Leftarrow ”: haben MPCP Lösung für $f(\langle M \rangle)$

\rightsquigarrow Lösung beginnt mit initialer Konfiguraion

Überführungsregeln erzwingen valide Übergänge

\rightsquigarrow Lösung hört mit Abschluss auf

$\rightsquigarrow z_e$ wird erreicht

$(\#_{\square z_0 \square \#})$ - **initiale Konfiguration**

(^a_a) für alle $a \in \Gamma$ – **Kopierregeln**

$(\#_{\#}, (\#_{\# \square}), (\square \#))$ – **Randregeln**

$(^{z_e a}_{z_e})$ für alle $a \in \Gamma$ – **Löschregeln**

$(^{z_e \# \#}_{\#})$ – **Abschlussregel**

Überführungsregeln: $\forall z_i, z_j \in Z \text{ & } \forall a, b, c \in \Gamma$

$(^{z_i a}_{z_j b})$ falls $\delta(z_i, a) = (z_j, b, N)$

$(^{z_i a}_{b z_j})$ falls $\delta(z_i, a) = (z_j, b, R)$

$(^{a z_i b}_{z_j a c})$ falls $\delta(z_i, b) = (z_j, c, L)$

Unentscheidbarkeit von PCP

Korollar

- ▶ $H_0 \leq \text{MPCP}$.
- ▶ PCP (und MPCP) sind unentscheidbar.
- ▶ H_0 ist semi-entscheidbar (und damit H und K)
- ▶ es gibt “universelle Turing-Maschine”

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem
7. Aufzählbarkeit & (Semi-)Entscheidbarkeit
8. Reduzierbarkeit
9. Satz von Rice
10. Das Postsche Korrespondenzproblem
- 11. Komplexität – Einführung**
12. NP-Vollständigkeit
13. coNP
14. PSPACE

Nichtdeterministische Turing-Maschinen I

Definition (Nichtdeterministische Turing-Machine)

Eine **Nichtdeterministische Turing-Maschine** (kurz **NTM**) ist ein Septupel
 $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ mit

- ▶ ...
- ▶ $\delta \subseteq (Z \setminus E) \times \Gamma \times Z \times \Gamma \times \{L, R, N\}$
- ▶ ...

Die "Folgekonfiguration"-Relation \vdash_M^1 von M spannt einen **Berechnungsbaum** auf

- ▶ $z_0 w \vdash_M^* k$ bedeutet: k kann von Startkonfiguration erreicht werden (**Berechnungspfad**)
- ▶ haltende/akzeptierende Konfig., halten auf/akzeptieren von Wörtern analog zu DTM
- ▶ **Zertifikat** für w in $T(M)$ ist endlicher Pfad von $z_0 w$ in akzeptierende Konfiguration
- ▶ akzeptierte Sprache analog zu DTM: $T(M) := \{w \in \Sigma^* \mid \exists_{\alpha, \beta \in \Gamma^*} \exists_{z \in E} : z_0 w \vdash_M^* \alpha z \beta\}$
- ▶ die von M berechnete Funktion ist $f : \mathbb{N} \rightarrow \mathbb{N}$ sodass, für alle $x \in \mathbb{N}$ und $y \in \mathbb{N}$,
$$f(x) = y \quad \Leftrightarrow \quad \{y' \in \Gamma^* \mid \exists_{z \in E} z_0 \text{BIN}(x) \vdash_M^* z y'\} = \{\text{BIN}(y)\}$$

Nichtdeterministische Turing-Maschinen II

Bemerkung: DTM sind spezielle NTM (ohne Gebrauch des Nichtdeterminismus)

Theorem

Für jede NTM N gibt es eine DTM M mit $T(M) = T(N)$.

Beweis (Idee)

Zeigen: $T(N)$ ist Wertebereich einer berechenbaren Funktion ($\leadsto T(N)$ semi-entscheidbar)

$$f(x, z) = \begin{cases} x & \text{falls } z \text{ ein Zertifikat für } x \text{ in } T(N) \text{ ist} \\ \perp & \text{sonst} \end{cases}$$

f kann von DTM berechnet werden indem sie dem Pfad im Berechnungsbaum von N folgt.

Einführung Komplexitätstheorie - TSP



Quelle: http://de.wikipedia.org/wiki/Datei:TSP_Deutschland_3.png

Algorithmische Komplexität

Bisher: qualitativ: berechenbar/entscheidbar oder nicht?

Jetzt: quantitativ: wie schnell/effizient kann ein entscheidbares Problem entschieden werden?
... es gibt viele Algorithmen zur Lösung berechenbarer Probleme wie z.B.

- ▶ Sortieren
- ▶ Potenzieren einer natürlichen Zahl
- ▶ ...

Einige davon sind

- ▶ schneller (weniger Elementaroperationen) oder
- ▶ platzsparender (weniger Speicher) als Andere.

Zentrale Frage

Wann ist ein Algorithmus **effizient** bzw. ein Berechnungsproblem **effizient lösbar**?

(Praktisch meist von Anwendung abhängig)

O -Notation zur Laufzeitanalyse

Problem: wie misst man Laufzeit von Algorithmen?

Beobachtung: Laufzeit muss (mindestens) von der Eingabegröße n abhängen

Ziel: "Effizienz" von Algorithmen unabhängig von Rechentechnik & Programmiersprache

↪ "Landau-Symbole" / O -Notation

Definition

Seien $f, g : \mathbb{N} \rightarrow \mathbb{N}$. Dann,

- ▶ $f \in O(g)$ falls $\exists_{c \in \mathbb{N}^+} \exists_{n_0 \in \mathbb{N}} \forall_{n \geq n_0} f(n) \leq c \cdot g(n)$,
- ▶ $f \in \Theta(g)$ falls $f \in O(g)$ und $g \in O(f)$,

Beispiele

- | | | |
|------------------------------|---------------------------|---|
| ▶ $10\sqrt{n} \in O(n)$ | ▶ $10^6 \in \Theta(1)$ | ▶ $n^{10} \in O(2^n)$ |
| ▶ $2n \in \Theta(n)$ | ▶ $n \log_2 n \in O(n^2)$ | ▶ $3n^4 + 5n^3 + 7 \log_2 n \in O(n^4)$ |
| ▶ $\log_2 n \in O(\sqrt{n})$ | ▶ $n\sqrt{n} \in O(n^2)$ | |

Deterministische Zeitklassen

Definition (time_M , $\text{DTIME}(f(n))$)

Für jede (Mehrband-) DTM M sei $\text{time}_M(n)$ die maximale Anzahl Konfigurationsübergänge von M auf Eingaben x der Länge n (Schritte bevor M auf x hält).

Für eine monoton wachsende Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ ist $\text{DTIME}(f(n))$ die Klasse aller Sprachen $L \subseteq \Sigma^*$, die von einer deterministischen Mehrband-TM M akzeptiert werden, welche für jedes $x \in \Sigma^*$ maximal $O(f(|x|))$ Schritte ausführt, das heißt,

$$\text{DTIME}(f(n)) := \{L \subseteq \Sigma^* \mid \exists_{\text{DTM } M} L = T(M) \wedge \text{time}_M(n) \in O(f(n))\}$$

Definition (P)

$$P := \bigcup_{k \geq 1} \text{DTIME}(n^k).$$

“deterministisch, in Polynomzeit”

Nichtdeterministische Zeitklassen

Definition (time_N , $\text{NTIME}(f(n))$)

Für jede (Mehrband-) **NTM** N sei $\text{time}_N(n)$ die maximale Länge eines Berechnungspfades von N auf Eingaben x der Länge n .

Für eine monoton wachsende Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ ist $\text{NTIME}(f(n))$ die Klasse aller Sprachen $L \subseteq \Sigma^*$, die von einer **nichtdeterministischen** Mehrband-TM N akzeptiert werden, deren Berechnungspfade für jede Eingabe $x \in \Sigma^*$ maximal Länge $O(f(|x|))$ haben, das heißt,

$$\text{NTIME}(f(n)) := \{L \subseteq \Sigma^* \mid \exists_{\text{NTM } N} L = T(N) \wedge \text{time}_N(n) \in O(f(n))\}$$

Definition (NP)

$$\text{NP} := \bigcup_{k \geq 1} \text{NTIME}(n^k).$$

“**nicht**deterministisch, in Polynomzeit”

Bemerkung: $P \subseteq \text{NP}$ klar, da jede DTM eine NTM ist.

Alternative Definition von NP

Theorem (Alternative Definition für NP („Guess and Check“))

Eine Sprache $L \subseteq \Sigma^*$ ist in NP, gdw. ein Polynom $p : \mathbb{N} \rightarrow \mathbb{N}$ und eine polynomiell zeitbeschränkte DTM M (d.h. $\text{time}_M(n) \in O(n^c)$) existieren, sodass für jedes $x \in \Sigma^*$ gilt

$$x \in L \Leftrightarrow \exists_{u \in \Sigma^{p(|x|)}} \langle x, u \rangle \in T(M).$$

Beweis (Skizze)

“ \Rightarrow ”: Sei $L \in \text{NP}$, d.h. es gibt eine polynomiell zeitbeschränkte NTM N mit $T(N) = L$.

Wir wählen u als Kodierung eines akzeptierenden Berechnungspfads (“Zertifikat”) für x in $T(N)$.

Das Zertifikat ist polynomiell lang, da N polynomiell zeitbeschränkt ist.

$\leadsto x \in L$ gdw. es ein solches Zertifikat $u \in \Sigma^{p(|x|)}$ für x in $T(N)$ gibt.

“ \Leftarrow ”: Sei M eine DTM wie im Theorem, zeitbeschränkt durch Polynom q .

Wir konstruieren eine NTM N die:

1. das Zertifikat u der Länge $p(|x|)$ nichtdeterministisch erzeugt (“räte”) und
2. sich danach wie M auf $\langle x, u \rangle$ verhält.

$\leadsto N$ terminiert in $p(|x|) + q(|x| + |u|)$ Schritten (also polynomieller Zeit) und

$x \in L \Leftrightarrow \exists_{u \in \Sigma^{p(|x|)}} \langle x, u \rangle \in T(M) \Leftrightarrow x \in T(N)$. Also $L \in \text{NTIME}(p(n) + q(n + p(n)))$, also $L \in \text{NP}$.

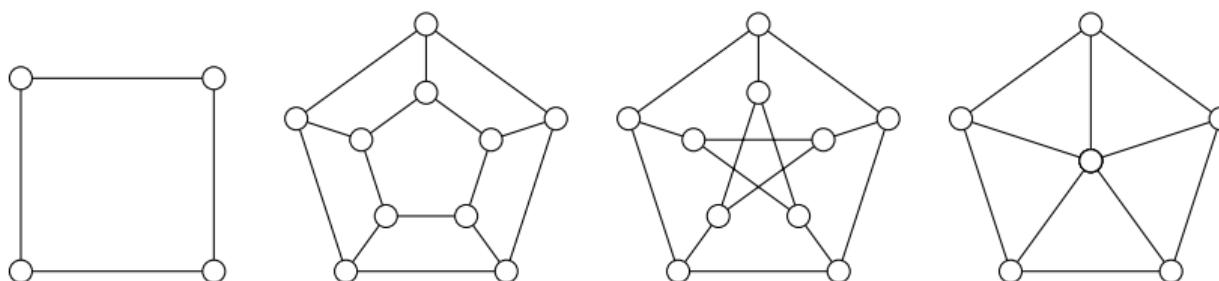
3-COLORING versus 2-COLORING

3-Coloring (2-Coloring)

Eingabe: ungerichteter Graph $G = (V, E)$

Frage: Lassen sich die Knoten von G mit **drei** (**zwei**) Farben so färben, dass keine zwei mit einer Kante verbundenen Knoten die gleiche Farbe haben?

Beispiele: Dreifärbbar? Zweifärbbar?



Mitteilung: Beide Probleme liegen in NP und 2-Coloring sogar in P

Frage: geben Sie einen deterministischen Polynomzeitalgorithmus für 2-Coloring an

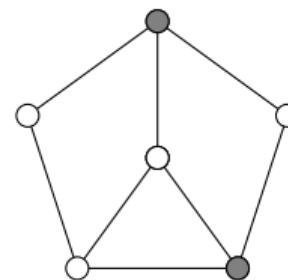
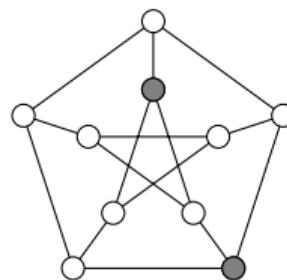
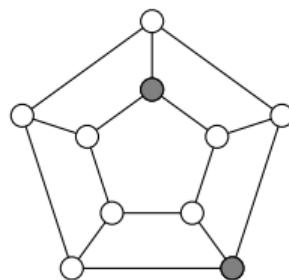
Longest Path versus Shortest Path

Shortest Path (Longest Path)

Eingabe: ungerichteter Graph $G = (V, E)$, zwei Knoten s, t und eine natürliche Zahl $k \leq |V|$

Frage: Existiert ein „einfacher“ Pfad zwischen s und t der Länge **höchstens** (mind.) k ?

Beispiel: Pfad der Länge ≤ 2 ? Pfad der Länge ≥ 9 (oder ≥ 5)?



Mitteilung: Beide Probleme liegen in NP und Shortest Path liegt sogar in P (Breitensuche)!

3-SAT versus 2-SAT

3-SAT (2-SAT)

Eingabe: aussagenlogische Formel F in „konjunktiver Normalform“ mit ≤ 3 (bzw. ≤ 2) Literalen pro Klausel.

Frage: Ist F erfüllbar, d.h. gibt es eine $\{0, 1\}$ -wertige Belegung der in F verwendeten Booleschen Variablen derart, dass F zu wahr (d.h. 1) ausgewertet wird?

Beispiele

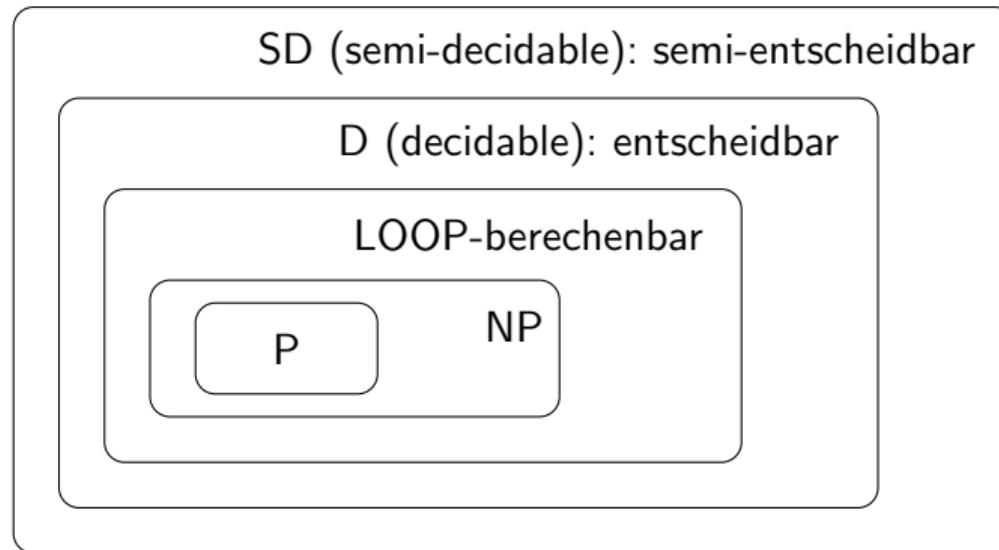
- ▶ $(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_3 \vee x_4) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_2 \vee x_3 \vee x_4)$
ist erfüllbar z.B. mit $x_1 = 0$, $x_2 = 0$, $x_3 = 1$ (und x_4 beliebig).
- ▶ $(x_1 \vee \overline{x_2}) \wedge (x_1 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (x_2 \vee x_3)$ nicht erfüllbar

Mitteilung: Beide Probleme liegen in NP und 2-SAT liegt sogar in P

P versus NP

Die bekannteste offene Frage der (Theoretischen) Informatik ist: $P \stackrel{?}{=} NP$.

Zur Einordnung von P versus NP: „Gegläubtes Schaubild“ (unter $P \subsetneq NP$):



Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem
7. Aufzählbarkeit & (Semi-)Entscheidbarkeit
8. Reduzierbarkeit
9. Satz von Rice
10. Das Postsche Korrespondenzproblem
11. Komplexität – Einführung
- 12. NP-Vollständigkeit**
13. coNP
14. PSPACE

Polynomzeitreduktion I

Vielleicht das wichtigste Konzept der Komplexitätstheorie!

Definition

Eine Sprache $A \subseteq \Sigma^*$ heißt **polynomiell reduzierbar auf** eine Sprache $B \subseteq \Pi^*$ (**in Zeichen** $A \leq_m^p B$), wenn es eine totale, **in Polynomzeit** berechenbare Funktion $f: \Sigma^* \rightarrow \Pi^*$ gibt, sodass für alle $x \in \Sigma^*$ gilt

$$x \in A \Leftrightarrow f(x) \in B.$$

Wir nennen f eine **Polynomzeit-Reduktion** von A auf B
(Beachte: f muss weder surjektiv noch injektiv sein).

Bemerkung: „ m “ in \leq_m^p steht für „many-one-Reduktion“.

Mitteilungen:

- (a) $A \leq_m^p B \Rightarrow A \leq B$
- (b) \leq_m^p ist transitiv, d.h. wenn $A \leq_m^p B$ und $B \leq_m^p C$, dann auch $A \leq_m^p C$
(Konkatenation der Reduktionen ist Polynomzeitreduktion von A auf C)

Transitivität der Polynomzeitreduktion

Beweis (für (b))

Sei f die Reduktionsfunktion für $A \leq_m^p B$, die in polynomieller Zeit $p(n)$ berechnet werden kann, und sei g die Reduktionsfunktion für $B \leq_m^p C$, die in polynomieller Zeit $q(n)$ berechnet werden kann.

Dann ist $g \circ f$ eine Reduktionsfunktion von A auf C , denn es gilt:

$$\forall x \in \Sigma^* : x \in A \Leftrightarrow f(x) \in B \Leftrightarrow g(f(x)) \in C.$$

Die Berechnung von $f(x)$ kann in $p(|x|)$ Schritten durchgeführt werden, also gilt auch $|f(x)| \leq p(|x|)$. Daher kann $g(f(x))$ mit höchstens $q(p(|x|))$ Schritten berechnet werden. Somit ist $g \circ f$ also polynomzeitberechenbar. ■

Polynomzeitreduktion II

Lemma

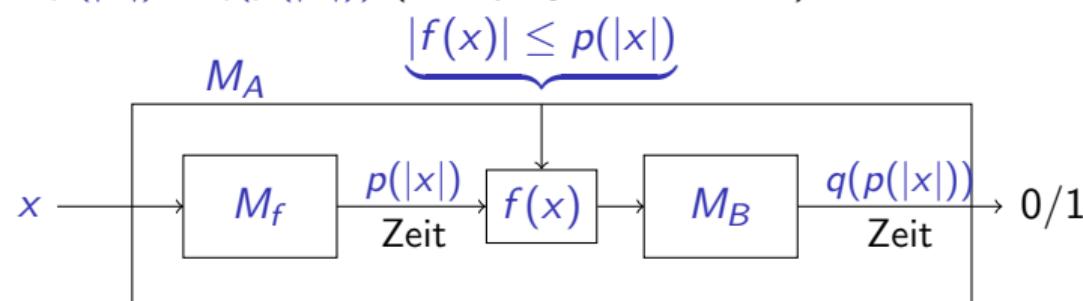
Gilt $A \leq_m^p B$ und ist $B \in P$ (bzw. $B \in NP$), so ist auch $A \in P$ (bzw. $A \in NP$).

Beweis

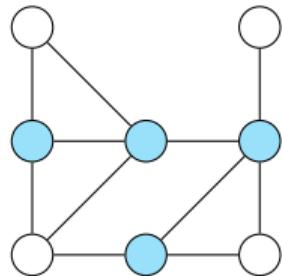
1. $A \leq_m^p B \rightsquigarrow$ „Reduktionsfunktion“ f in $p(n)$ Schritten berechenbar durch TM M_f
2. $B \in P$ (bzw. $B \in NP$) $\rightsquigarrow B$ in $q(n)$ Schritten entscheidbar durch TM M_B
(wobei p und q Polynome)

Wie zuvor gilt $\chi_A = \chi_B \circ f$

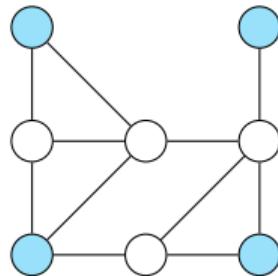
$\rightsquigarrow \chi_A$ berechnet in $p(|x|) + q(p(|x|))$ (also polynomiell viele) Schritten.



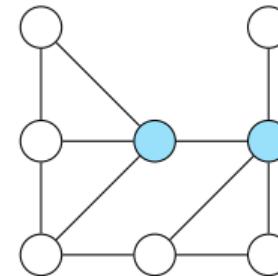
INDEPENDENT SET, VERTEX COVER und DOMINATING SET



VERTEX COVER



INDEPENDENT SET



DOMINATING SET

Eingabe: ungerichteter Graph G , Zahl $k > 0$

Frage: gibt es k Knoten in G , sodass ...

Vertex Cover: ... jede Kante in G mindestens einen dieser k Knoten als Endpunkt hat?

Independent Set: ... keine 2 dieser k Knoten mit einer Kante verbunden sind?

Dominating Set: ... jeder andere Knoten eine Kante zu mindestens einem dieser Knoten hat?

VERTEX COVER und INDEPENDENT SET

Theorem

VERTEX COVER \leq_m^p INDEPENDENT SET.

Beweis

Definiere Reduktionsfunktion f vermöge $f(\langle G, k \rangle) := \langle G, |V(G)| - k \rangle$.
(offensichtlich ist f in polynomieller Zeit berechenbar)

Dann gilt:

$\langle G, k \rangle \in \text{VERTEX COVER} \Leftrightarrow G$ hat eine Knotenmenge $X \subseteq V(G)$ mit $|X| \leq k$, so dass
jede Kante mindestens einen Endpunkt in X hat
 $\Leftrightarrow G$ hat eine Knotenmenge $X \subseteq V(G)$ mit $|X| \leq k$, so dass
keine Kante beide Endpunkte in $V(G) \setminus X$ hat
 $\Leftrightarrow \langle G, |V(G)| - k \rangle \in \text{INDEPENDENT SET}$.

NP-Vollständigkeit

Definition

Eine Sprache $A \subseteq \Sigma^*$ heißt...

- ... **NP-schwer**, falls $\forall_{L \in \text{NP}} L \leq_m^p A$.
- ... **NP-vollständig**, wenn A NP-schwer ist und $A \in \text{NP}$ gilt.

Anschaulich: (mit „polynomieller Unschärfe“)

- NP-schwere Sprachen sind „mindestens so schwer“ zu entscheiden wie jede Sprache in NP
- NP-vollständige Sprachen sind „genau so schwer“ wie jede NP-vollständige Sprache

Lemma

Ist A NP-schwer und $A \leq_m^p B$, so ist auch B NP-schwer

Beweis

Für jede Sprache $L \in \text{NP}$ gilt $L \leq_m^p A \leq_m^p B$.

Somit gilt wegen Transitivität auch $L \leq_m^p B$. Also ist B auch NP-schwer.

NP-Vollständigkeit II

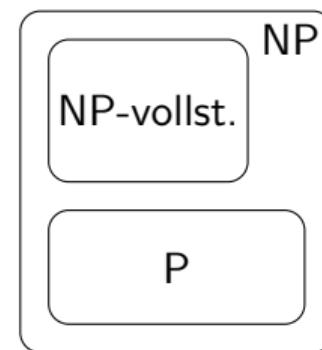
Theorem

Für jede NP-vollständige Sprache A gilt: $A \in P \Leftrightarrow P = NP$.

Beweis

„ \Rightarrow “: $(\forall L \in NP \ L \leq_m^P A) \wedge (A \in P) \Rightarrow \forall L \in NP \ L \in P \Rightarrow NP = P$
„ \Leftarrow “: $(A \in NP) \wedge (P = NP) \Rightarrow A \in P$

„**Geglaubte**“ (d.h. Annahme $P \neq NP$) **Situation:**



Erfüllbarkeitsproblem

SAT

Eingabe: aussagenlogische Formel F

Frage: Ist F erfüllbar, d.h. gibt es eine $\{0, 1\}$ -wertige Belegung der in F verwendeten Booleschen Variablen derart, dass F zu wahr (d.h. 1) ausgewertet wird?

Beispiele

0, 1,

$x_1, x_2, \overline{x_3}$,

$(x_1 \wedge \overline{x_2})$,

$((\overline{x_1 \wedge \overline{x_2}}) \vee x_2 \vee \overline{x_3})$

Theorem (Satz von Cook und Levin)

SAT ist NP-vollständig.

Beweis (Idee, Details später)

Teil 1: „SAT \in NP“: rate erfüllende Belegung (Zertifikat) und verifiziere sie.

Teil 2: „SAT ist NP-schwer“: mit $L \in \text{NP}$ beliebig, transformiere NTM N mit $T(N) = L$ in Formel $\varphi(x)$ sodass $x \in L \Leftrightarrow \varphi(x) \in \text{SAT}$.

CNF-SAT ist NP-vollständig

CNF-SAT

Eingabe: aussagenlogische Formel F in „konjunktiver Normalform“

Frage: Ist F erfüllbar, d.h. gibt es eine $\{0, 1\}$ -wertige Belegung der in F verwendeten Booleschen Variablen derart, dass F zu wahr (d.h. 1) ausgewertet wird?

Theorem

SAT \leq_m^p CNF-SAT (\rightsquigarrow CNF-SAT NP-vollständig)

Beweis (Skizze)

Reduktion: $\varphi \rightsquigarrow$ erfüllbarkeits-äquivalente Formel ψ :

(1) neue Variable y_i für jeden Knoten im “Formelbaum”

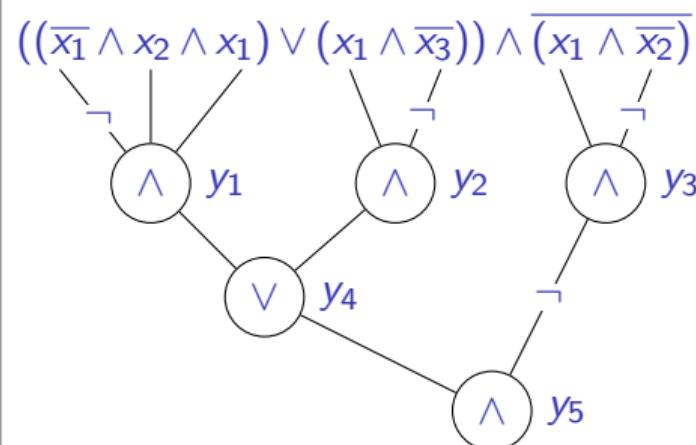
mit “äquivalentem Wahrheitswert”

(2) neue Klausel für die Wurzel

ψ erfüllbar $\Leftrightarrow \varphi$ erfüllbar ✓

poly-time computable ✓ (ab jetzt implizit)

Beispiel



3-SAT ist NP-vollständig

Theorem

CNF-SAT \leq_m^p 3-SAT (also ist 3-SAT NP-vollständig).

Beweis (Skizze)

Reduktion: CNF-Formel $\varphi \sim$ erfüllbarkeits-äquivalente 3CNF-Formel ψ

Für jede Klausel $c_j = (\ell_1 \vee \ell_2 \vee \dots \vee \ell_r) \in \varphi$,

- ▶ falls $r \leq 3$, dann füge c_j zu ψ hinzu;
- ▶ sonst füge c'_j hinzu mit

$$c'_j := (\ell_1 \vee \ell_2 \vee y_1) \wedge (\overline{y_1} \vee \ell_3 \vee y_2) \wedge (\overline{y_2} \vee \ell_4 \vee y_3) \dots (\overline{y_{r-3}} \vee \ell_{r-1} \vee \ell_r)$$

wobei y_1, \dots, y_{r-3} neue Variablen sind.

\sim Belegung β erfüllt $c_j \Leftrightarrow$ Erweiterung von β erfüllt c'_j

ψ erfüllbar $\Leftrightarrow \varphi$ erfüllbar ✓

Bemerkung: $|\psi| \leq 2|\varphi|$

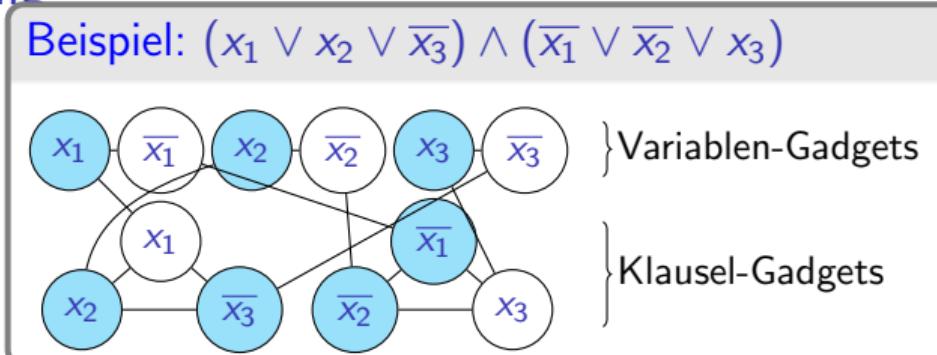
VERTEX COVER ist NP-vollständig

Theorem

3-SAT \leq_m^P VERTEX COVER.

Beweis (Skizze)

Formel $\varphi \rightsquigarrow (G, k = \#\text{Var} + 2\#\text{Klauseln})$



1. Variablen-Gadget: Variable $x_i \rightsquigarrow$ 2 benachbarte Knoten mit Beschriftungen x_i und \bar{x}_i
2. Klausel-Gadget: Klausel $(\ell_1 \vee \ell_2 \vee \ell_3) \rightsquigarrow$ Dreieck mit Beschriftungen ℓ_1, ℓ_2, ℓ_3
3. Verbinde Knoten mit gleicher Beschriftung

„ \Rightarrow “: aus Variablen-Gadget, wähle entsprechend der Belegung

\rightsquigarrow alle anderen Kanten mit 2 Knoten aus jedem Klausel-Gadget überdeckt

„ \Leftarrow “:

(a) = 1 Knoten von jedem Variablen-Gadget in jeder VC-Lösung

(b) = 2 Knoten von jedem Klausel-Gadget in jeder VC-Lösung.

\rightsquigarrow jedes Klausel-Gadget benachbart zu einem Knoten in VC-Lösung

\rightsquigarrow entsprechende Belegung erfüllt die Formel!

DOMINATING SET ist NP-vollständig

Theorem

VERTEX COVER \leq_m^p DOMINATING SET.

Beweis (Skizze)

$$(G, k) \sim (G', k)$$

1. setze initial $G' = G$
2. für jede Kante $e = \{u, v\}$ in G :
erzeuge einen neuen (grauen) Knoten in G' und verbinde ihn mit u und v

Korrektheit: „ \Rightarrow “: VC-Lösung in G ist auch DS-Lösung in G'

„ \Leftarrow “: Sei $X \subseteq V(G')$ eine DS-Lösung für G' mit $|X| \leq k$

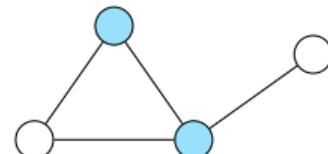
(a) neuer (grauer) Knoten \in DS-Lösung \rightsquigarrow mit weißem Nachbarn tauschen

\rightsquigarrow Lösung ohne graue Knoten

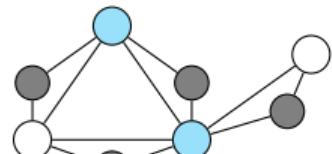
(b) graue Knoten dominieren \rightsquigarrow jede Kante in G hat Endpunkt in X

$\rightsquigarrow X$ ist vertex cover in G

Beispiel



VERTEX COVER



DOMINATING SET

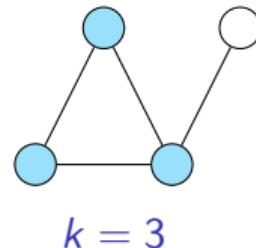
CLIQUE ist NP-vollständig

Clique

Eingabe: ungerichteter Graph G und Zahl $k \in \mathbb{N}$

Frage: Hat G einen vollständigen Teilgraph G' mit $\geq k$ Knoten?

Beispiel



Theorem

INDEPENDENT SET \leq_m^p CLIQUE.

Beweis (Skizze)

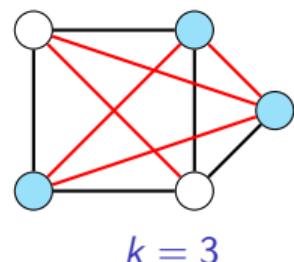
$$(G = (V, E), k) \rightsquigarrow (\overline{G} = (V, \binom{V}{2} \setminus E), k)$$

Korrektheit:

Jede unabhängige Knotenmenge in G
bildet eine Clique in \overline{G} und umgekehrt, also:

$$(G, k) \in \text{INDEPENDENT SET} \Leftrightarrow (\overline{G}, k) \in \text{CLIQUE}$$

Beispiel

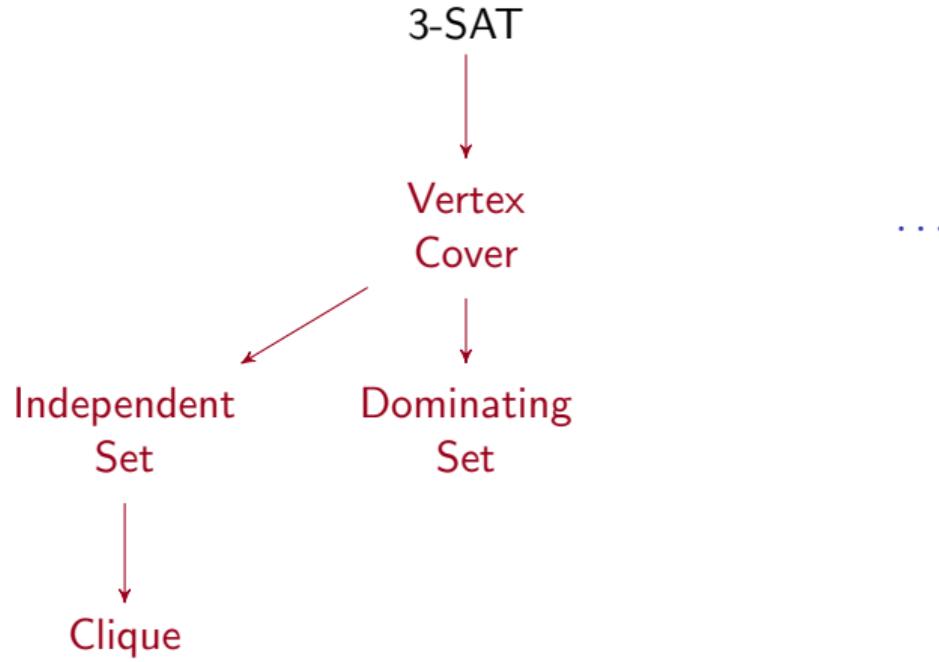


INDEPENDENT SET
CLIQUE

Wenn ein Dominostein fiele...



Netzwerk polynomieller Reduktionen I



HITTING SET und SET COVER

Eingabe:

- (1) Grundmenge („Universum“) $U := \{x_1, x_2, \dots, x_n\}$,
- (2) eine Teilmengenfamilie $\mathcal{F} := \{S_1, S_2, \dots, S_m\}$ mit $S_i \subseteq U$ für $1 \leq i \leq n$ und
- (3) ein $k \in \mathbb{N}$

Hitting Set

Frage: Existiert eine Teilmenge $X \subseteq U$ mit $|X| \leq k$ und $X \cap S_i \neq \emptyset$ für jedes S_i ?

Set Cover

Frage: Existiert ein $\mathcal{Z} \subseteq \mathcal{F}$ mit $|\mathcal{Z}| \leq k$ und $\bigcup_{S \in \mathcal{Z}} S = U$?

Beispiel

- (1) $U = \{1, 2, 3, 4, 5, 6\}$,
 - (2) $S_1 = \{1, 3\}$, $S_2 = \{3, 4\}$, $S_3 = \{1, 5\}$, $S_4 = \{2, 4, 6\}$, $S_5 = \{1, 3, 5\}$
 - (3) $k = 2$
- $\leadsto X = \{1, 4\}$, $\mathcal{Z} = \{S_4, S_5\}$.

HITTING SET ist NP-vollständig

Theorem

VERTEX COVER \leq_m^p HITTING SET.

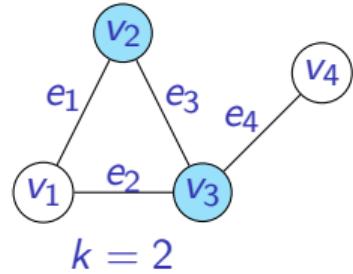
Beweis (Skizze)

$(G = (V, E), k) \rightsquigarrow (U = V, \mathcal{F} = E, k)$

Korrektheit: klar

In der Tat ist VERTEX COVER auch bekannt als „2-Hitting Set“.

Beispiel



$U = \{v_1, v_2, v_3, v_4\}$
 $\mathcal{F} = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}, \{v_3, v_4\}\}$

SET COVER ist NP-vollständig

Theorem

HITTING SET \leq_m^p SET COVER.

Beweis (Skizze)

$(U, \mathcal{F}, k) \sim (U_{SC} = \mathcal{F}, \mathcal{F}_{SC} = \{F_x \mid x \in U\}, k)$
mit $F_x := \{S_i \in \mathcal{F} \mid x \in S_i\}$

Korrektheit:

$X \subseteq U$ ist ein Hitting Set für \mathcal{F}

$\Leftrightarrow \forall S_i \in \mathcal{F} \exists x \in X \ x \in S_i$

$\Leftrightarrow \bigcup_{x \in X} F_x = \mathcal{F}$

$\Leftrightarrow Z := \{F_x \mid x \in X\}$ ist ein Set Cover für $\mathcal{F} = U_{SC}$

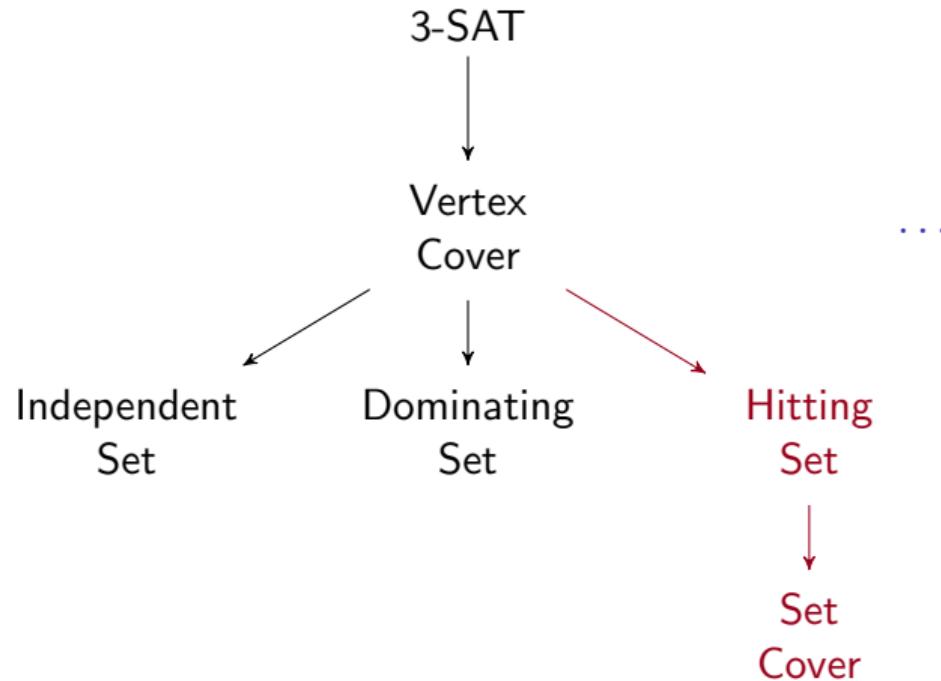
HS/SC Dualität

$\mathcal{F} \setminus U$	1	2	3	4
{1, 2}				
{1, 4}				
{2, 3, 4}				



$\mathcal{F}_{SC} \setminus \mathcal{F}$	{1, 2}	{1, 4}	{2, 3, 4}
F_1 1			
F_2 2			
F_3 3			
F_4 4			

Netzwerk polynomieller Reduktionen II



SUBSET SUM

Ein Problem u.a. aus dem Bereich „Scheduling“ (Ablaufsteuerung).

Subset Sum

Eingabe: Multi-Menge $U := \{u_1, u_2, \dots, u_n\}$ von natürlichen Zahlen und eine Zahl $B \in \mathbb{N}$

Frage: Existiert eine Teilmenge $X \subseteq U$, die sich zu B summiert, d.h. $\sum_{u \in X} u = B$?

Beispiel

$U = \{4, 4, 11, 16, 21\}$ und $B = 29$.

$\rightsquigarrow X = \{4, 4, 21\}$.

SUBSET SUM ist NP-vollständig

Theorem

3-SAT \leq_m^p SUBSET SUM.

Beweis (Skizze)

Konstruktion: Variablen x_1, \dots, x_n , Klauseln c_1, \dots, c_m

1. Für jedes x_i bilde zwei Dezimalzahlen $y_i, z_i \in \{0, 1\}^{n+m}$ mit:
Vordere n Ziffern: i -te Stelle von y_i und z_i ist 1, alle anderen sind 0.
Hintere m Ziffern: j -te Stelle von y_i ist 1 falls $x_i \in c_j$, und sonst 0.
 j -te Stelle von z_i ist 1 falls $\bar{x}_i \in c_j$, und sonst 0.
2. Für jede Klausel c_j , bilde zwei **dezimale** „Füllzahlen“ g_j, h_j
3. Setze Dezimalzahl $B := \underbrace{1 \dots 1}_{n} \underbrace{3 \dots 3}_{m}$.

Korrektheit „ \Rightarrow “: Sei β eine erfüllende Belegung.

\leadsto Lösung = $\{y_i \mid \beta(x_i) = 1\} \cup \{z_i \mid \beta(x_i) = 0\} + \text{geeignete } g_i \& h_i$

„ \Leftarrow “: Sei X eine Menge von Zahlen mit $\sum_{u \in X} u = B$.

erste n Ziffern $\leadsto y_i \in X \Leftrightarrow z_i \notin X$

Die Belegung β mit $\beta(x_i) = 1$ falls $y_i \in X$, und $\beta(x_i) = 0$ sonst, ist erfüllend.

Beispiel

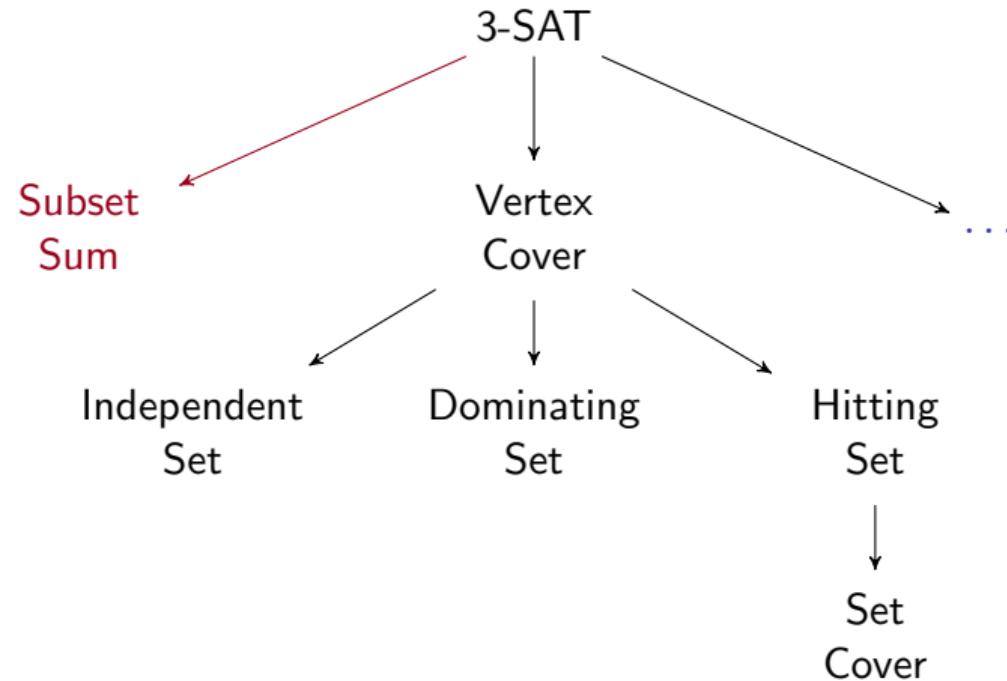
$$c_1: x_1 \vee x_2 \vee \bar{x}_3$$

$$c_2: \bar{x}_1 \vee x_2 \vee x_3$$

$$c_3: \bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3$$

	x_1	x_2	x_3	c_1	c_2	c_3
y_1 :	1	0	0	1	0	0
z_1 :	1	0	0	0	1	1
y_2 :	0	1	0	1	1	0
z_2 :	0	1	0	0	0	1
y_3 :	0	0	1	0	1	0
z_3 :	0	0	1	1	0	1
g_1 :	0	0	0	1	0	0
h_1 :	0	0	0	1	0	0
g_2 :	0	0	0	0	1	0
h_2 :	0	0	0	0	0	1
g_3 :	0	0	0	0	0	1
h_3 :	0	0	0	0	0	1
$B :$	1	1	1	3	3	3

Netzwerk polynomieller Reduktionen III



Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem
7. Aufzählbarkeit & (Semi-)Entscheidbarkeit
8. Reduzierbarkeit
9. Satz von Rice
10. Das Postsche Korrespondenzproblem
11. Komplexität – Einführung
12. NP-Vollständigkeit
- 13. coNP**
14. PSPACE

Co-Nichtdeterministische Turing-Maschinen

Definition (Co-Nichtdeterministische Turing-Maschine)

- ▶ $\delta \subseteq (Z \setminus E) \times \Gamma \times Z \times \Gamma \times \{L, R, N\}$
- ▶ “Folgekonfiguration”-Relation \vdash_M^1 von M spannt Berechnungsbaum auf
- ▶ coNTM akzeptiert \Leftrightarrow alle Berechnungspfade erreichen akzeptierende Konfiguration time_{coN} und $\text{coNTIME}(f(n))$ analog zu time_N und $\text{NTIME}(f(n))$

Definition (coNP)

$$\text{coNP} := \bigcup_{k \geq 1} \text{coNTIME}(n^k).$$

“**co-nicht**deterministisch, in Polynomzeit”

Theorem (Alternative Definition für coNP („Guess and Check“))

Eine Sprache $L \subseteq \Sigma^*$ ist in coNP, gdw. ein Polynom $p : \mathbb{N} \rightarrow \mathbb{N}$ und eine polynomiell zeitbeschränkte DTM M (d.h. $\text{time}_M(n) \in O(n^c)$) existieren, sodass für jedes $x \in \Sigma^*$ gilt

$$x \in L \Leftrightarrow \forall_{u \in \Sigma^{p(|x|)}} \langle x, u \rangle \in T(M)$$

beziehungsweise

$$x \notin L \Leftrightarrow \exists_{u \in \Sigma^{p(|x|)}} \langle x, u \rangle \notin T(M).$$

Beachte: zentraler Unterschied zu NP: „ \forall “ statt „ \exists “

Die Komplexitätsklasse coNP I

Erinnerung: Sei $L \subseteq \Sigma^*$, dann ist $\bar{L} := \Sigma^* \setminus L$ das Komplement von L .

Theorem

$$\text{coNP} = \{L \subseteq \Sigma^* \mid \bar{L} \in \text{NP}\}.$$

Beweis

Sei $L \subseteq \Sigma^*$. "Guess and Check" $\rightsquigarrow \bar{L} \in \text{NP}$ genau dann wenn es eine polynomiell zeitbeschränkte DTM M gibt mit

$$x \in \bar{L} \Leftrightarrow \exists_{u \in \Sigma^{p(|x|)}} \langle x, u \rangle \in T(M).$$

Eine solche DTM M gibt es genau dann, wenn es auch eine polynomiell zeitbeschränkte DTM M' gibt, die genau dann ablehnt, wenn M akzeptiert. Also gilt

$$\begin{aligned} x \in L &\Leftrightarrow x \notin \bar{L} \Leftrightarrow \neg(\exists_{u \in \Sigma^{p(|x|)}} \langle x, u \rangle \in T(M)) \\ &\Leftrightarrow \forall_{u \in \Sigma^{p(|x|)}} \langle x, u \rangle \notin T(M) \\ &\Leftrightarrow \forall_{u \in \Sigma^{p(|x|)}} \langle x, u \rangle \in T(M') \end{aligned}$$

Die Komplexitätsklasse coNP II

Erinnerung: Sei $L \subseteq \Sigma^*$, dann ist $\bar{L} := \Sigma^* \setminus L$ das Komplement von L .

Theorem

$$\text{coNP} = \{L \subseteq \Sigma^* \mid \bar{L} \in \text{NP}\}.$$

Bemerkungen:

- ▶ coNP ist **nicht** das Komplement von NP (z.B. $H \notin \text{NP}$ und $H \notin \text{coNP}$)
- ▶ $P \subseteq \text{NP} \cap \text{coNP}$ (da $L \in P \Leftrightarrow \bar{L} \in P$)
- ▶ coNP-Vollständigkeit analog zu NP-Vollständigkeit:
 $A \subseteq \Sigma^*$ ist coNP-vollständig $\Leftrightarrow \forall_{L \in \text{coNP}} L \leq_m^P A$ und $A \in \text{coNP}$
- ▶ $\overline{\text{SAT}} := \{\varphi \mid \varphi \text{ ist unerfüllbar}\} \in \text{coNP}$ (sogar coNP-vollständig)
- ▶ $(P = \text{NP}) \Rightarrow (\text{NP} = \text{coNP} = P)$
für alle $L \in \text{coNP}$ gilt: $\bar{L} \in \text{NP} \Rightarrow \bar{L} \in P \Rightarrow L \in P$ und somit $L \in \text{NP}$
- ▶ **Offen:** $(\text{NP} = \text{coNP}) \Rightarrow (P = \text{NP})?$

Ein coNP-vollständiges Problem

TAUT

Eingabe: aussagenlogische Formel F

Frage: Ist F eine **Tautologie**, d.h. wird F für alle $\{0,1\}$ -wertigen Belegungen der in F verwendeten Booleschen Variablen zu wahr (d.h. 1) ausgewertet?

Theorem

TAUT ist coNP-vollständig.

Beweis

1. TAUT \in coNP via “Guess and Check” (nicht-erfüllende Belegung zertifiziert $F \notin$ TAUT)

2. TAUT ist coNP-schwer (d.h. $\forall L \in$ coNP $L \leq_m^p$ TAUT):

Da $\bar{L} \in$ NP, gilt $\bar{L} \leq_m^p$ SAT vermöge einer Polynomzeitreduktion $f: x \mapsto F_x$. Also
 $x \in L \Leftrightarrow x \notin \bar{L} \Leftrightarrow F_x \notin$ SAT $\Leftrightarrow \neg F_x \in$ TAUT.

Also ist $g: x \mapsto \neg F_x$ eine Polynomzeitreduktion von L auf TAUT.

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem
7. Aufzählbarkeit & (Semi-)Entscheidbarkeit
8. Reduzierbarkeit
9. Satz von Rice
10. Das Postsche Korrespondenzproblem
11. Komplexität – Einführung
12. NP-Vollständigkeit
13. coNP
- 14. PSPACE**