

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)
4.-6. Semester 93 LP	Wahlpflicht Technische Informatik (6 LP)	Wahlpflicht Programmierpraktikum (6-9 LP)	Wahlpflicht Theoretische Informatik (6 LP)	Stochastik für Informatik (9 LP)	Informatik und Gesellschaft (6 LP)
		Wahlpflichtbereich Katalog Informatik (27-33 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”

Organisation

Vorlesungsbetrieb:

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

Tutorien:

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

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/UCMWYg3eBFp5bbqj1lllUku_w)

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Die Ackermannfunktion ~~*~~
6. (Un-)Entscheidbarkeit, Halteproblem und Reduzierbarkeit
7. Das Postsche Korrespondenzproblem
8. Komplexität – Einführung
9. NP-Vollständigkeit
10. PSPACE

es gibt unentscheidbare Probleme!

was ist praktisch berechenbar
Effizient!

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

Das „hello, world“-Problem

Ziel: Entwicklung von Programm E mit folgender Spezifikation:

Input: Programm $P \rightsquigarrow$ z.B.: Compiler / Interpreter

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).

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 ?

→ Top

~ Existiert ein Programm E auch für **beliebige** Programme P ?

→ nicht möglich ↴

Wiederholung: Endliche Automaten

Definition (Endlicher Automat)

- ▶ Ein (**deterministischer**) **endlicher Automat** (kurz **DFA**) ist ein Quintupel $M = (\underline{Z}, \underline{\Sigma}, \delta, z_0, E)$ mit
 - ▶ Z ist eine nichtleere, endliche Menge von **Zuständen**,
 - ▶ Σ ist ein nichtleeres, endliches Alphabet von **Eingabezeichen** mit $\underline{Z} \cap \underline{\Sigma} = \emptyset$,

Wiederholung: Endliche Automaten

Definition (Endlicher Automat)

- ▶ Ein (**deterministischer**) **endlicher Automat** (kurz **DFA**) ist ein Quintupel $M = (\underline{Z}, \underline{\Sigma}, \underline{\delta}, z_0, E)$ mit
 - ▶ Z ist eine nichtleere, endliche Menge von **Zuständen**,
 - ▶ Σ ist ein nichtleeres, endliches Alphabet von **Eingabezeichen** mit $\underline{Z} \cap \Sigma = \emptyset$,
 - ▶ $\delta: \underline{Z} \times \underline{\Sigma} \rightarrow \underline{Z}$ ist die **partielle Überführungsfunktion**,

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**.

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 $z \in Z$:

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

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

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 $z \in Z$:
$$\hat{\delta}(z, \underline{\epsilon}) := z$$
$$\forall_{x \in \Sigma^*} \quad \hat{\delta}(z, \underline{ax}) := \hat{\delta}(\hat{\delta}(z, a), \underline{x}) \quad \text{falls } \underline{\delta(z, a)} \neq \perp$$
- ▶ Ein DFA $M = (Z, \Sigma, \delta, z_0, E)$ **akzeptiert** ein Wort $w \in \Sigma^*$ falls $\hat{\delta}(z_0, w) \in E$

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	$\boxed{z_0}$	z_2

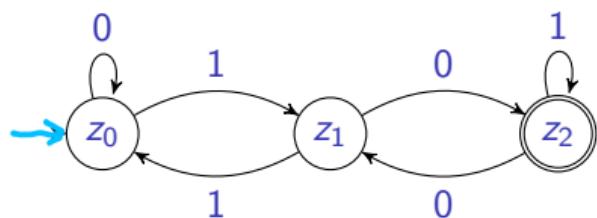
Wiederholung: Endliche Automaten II

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



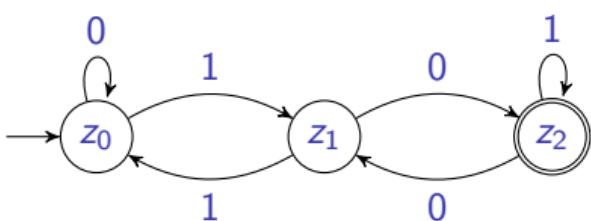
Wiederholung: Endliche Automaten II

Beispielautomat

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

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

Zustandsgraph



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

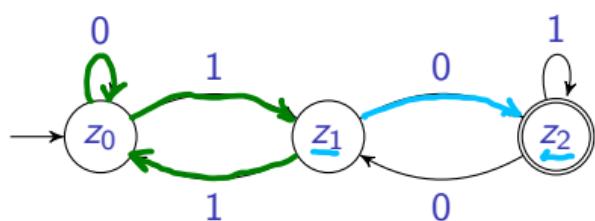
Wiederholung: Endliche Automaten II

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.

$$\underline{110} = 6$$

$$n = 3x + 1$$

$$n_0 = 2(3x + 1) + 0 = 6x + 2$$

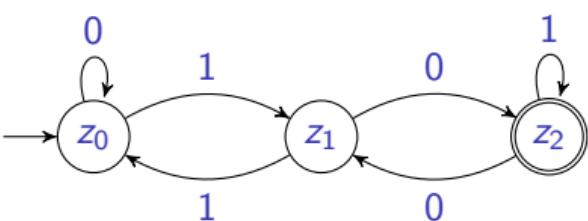
$$T(M) = \{w \in \{0, 1\}^* \mid w \text{ ist Binärdarstellung einer Zahl } n \text{ mit } n \bmod 3 = 2\}.$$

Wiederholung: Endliche Automaten II

Beispielautomat

$$M = (\{z_0, z_1, z_2\}, \{0, 1\}, \delta, z_0, \{z_2\}) \text{ mit } \begin{array}{c|ccc} \delta & z_0 & z_1 & z_2 \\ \hline 0 & z_0 & z_2 & z_1 \\ 1 & z_1 & z_0 & z_2 \end{array}$$

Zustandsgraph



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

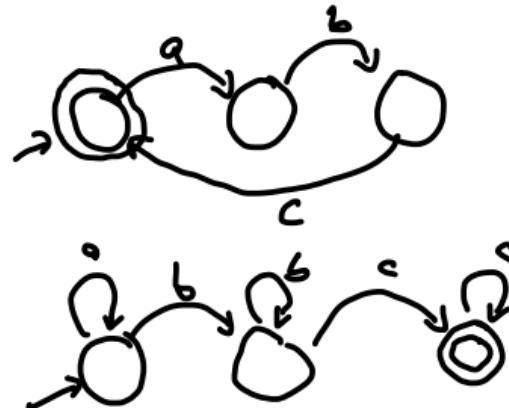
$$T(M) = \{w \in \{0, 1\}^* \mid w \text{ ist Binärdarstellung einer Zahl } n \text{ mit } n \bmod 3 = 2\}.$$

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?

- $\{w \in \{0,1\}^* \mid w \text{ ist Binärdarstellung einer geraden Zahl}\}$ ✓
- $\{a^n b^n \mid 0 \leq n \leq 1000\}$ ✓
- $\{\underline{a^n b^n} \mid n \geq 0\}$ ✗
- $\{(abc)^n \mid n \geq 0\}$ ✓
- $\{a^n b^m c^k \mid n, m, k \geq 1\}$ ✓
- $\{\underline{a^n b^n c^n} \mid n \geq 0\}$ ✗
- $\{\underline{a^i b^j c^i d^j} \mid i, j \geq 0\}$ ✗



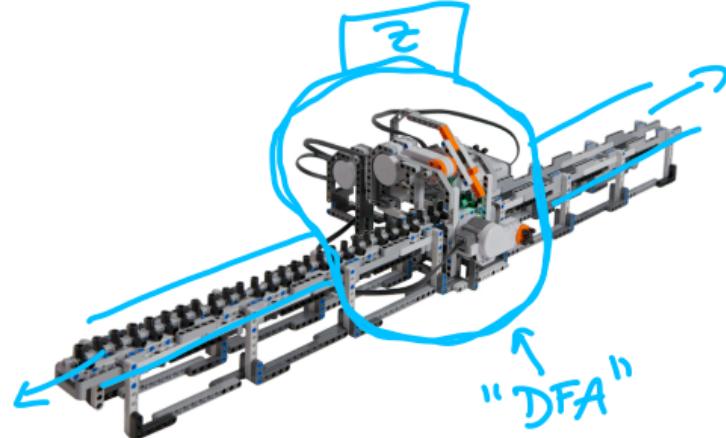
Die Turing Maschine



Alan Mathison Turing, 1912-1954.



Inspiration: „Menschliche Computer“ 1890.



LEGO Turing Maschine

Informell

endliche Kontrolle + unendliches Band

Quellen:

<http://therunnerelectic.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

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**,

- ▶ δ die partielle **Überführungsfunktion**
- ▶ $z_0 \in Z$, dem **Startzustand**,

- ▶ $E \subseteq Z$, der Menge der **Endzustände**.

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 $\underline{\Gamma \cap Z} = \emptyset$,
- ▶ δ die partielle **Überführungsfunktion**
- ▶ $z_0 \in Z$, dem **Startzustand**,
- ▶ $\underline{\square \in \Gamma \setminus \Sigma}$, dem **Blanksymbol** und
- ▶ $E \subseteq Z$, der Menge der **Endzustände**.

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**.

Definition Turing-Maschinen

Definition ((deterministische) Turing-Machine)

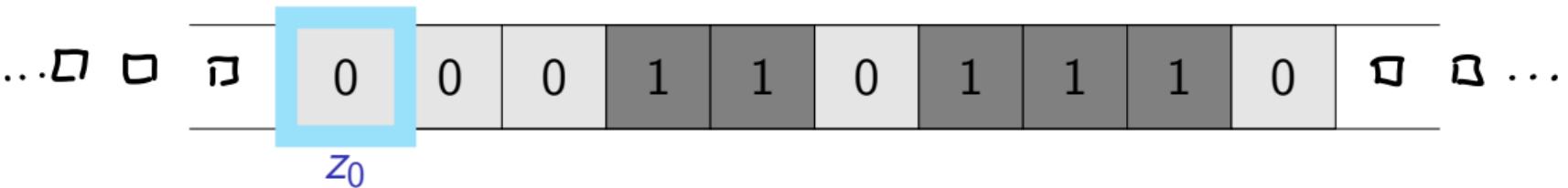
Eine **Turing-Maschine** (kurz **DTM**) ist ein Septupel $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ mit

- ▶ 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**.

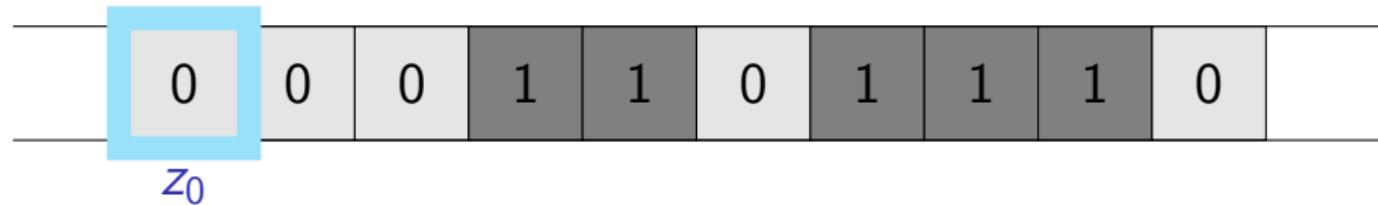
Interpretation: Wenn M im Zustand z das Zeichen a liest und $\underline{\delta(z, a)} = (\underline{z'}, \underline{a'}, \underline{p})$, so

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

Arbeitsweise einer Turing-Maschine

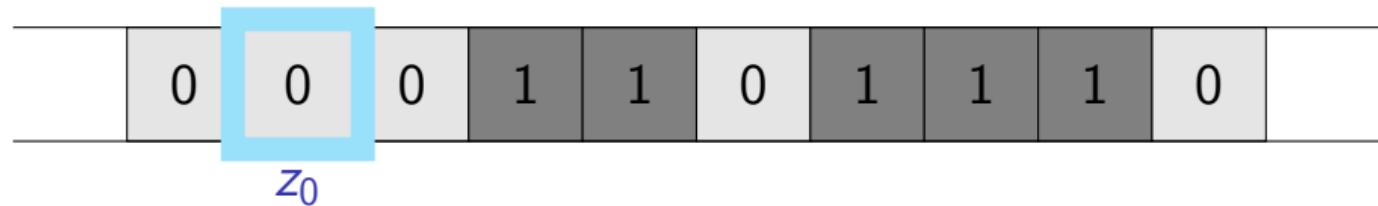


Arbeitsweise einer Turing-Maschine



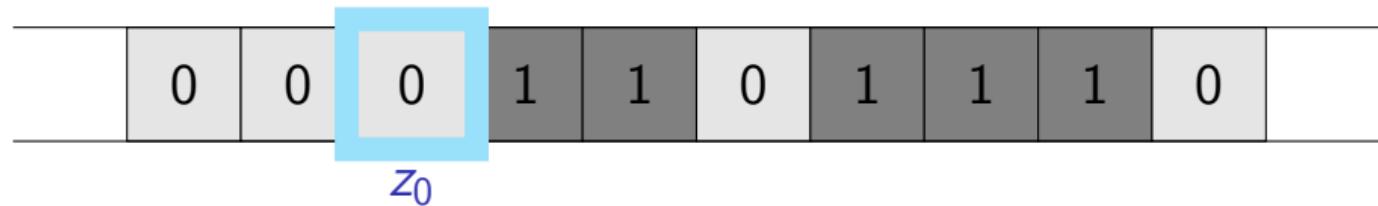
$$\delta(z_0, 0) = (z_0, 0, R)$$

Arbeitsweise einer Turing-Maschine



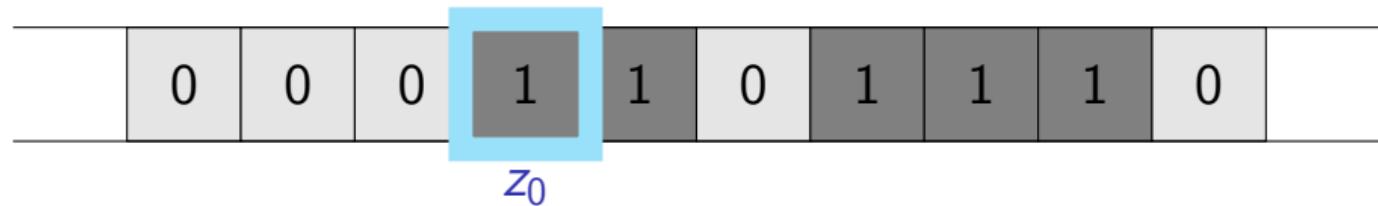
$$\delta(z_0, 0) = (z_0, 0, R)$$

Arbeitsweise einer Turing-Maschine



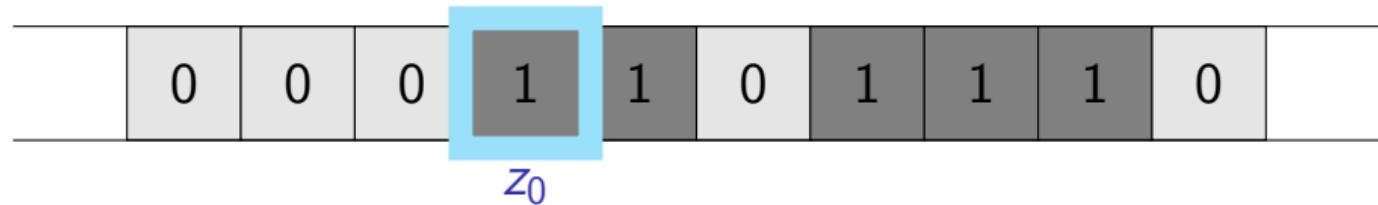
$$\delta(z_0, 0) = (z_0, 0, R)$$

Arbeitsweise einer Turing-Maschine



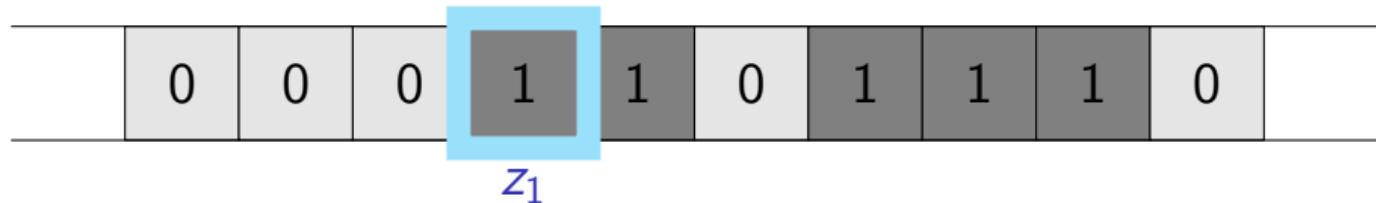
$$\delta(z_0, 0) = (z_0, 0, R)$$

Arbeitsweise einer Turing-Maschine



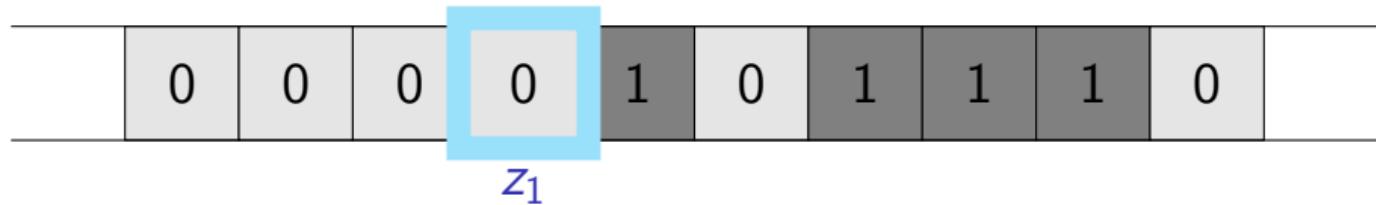
$$\begin{aligned}\delta(z_0, 0) &= (z_0, 0, R) \\ \delta(z_0, 1) &= (z_1, 0, L)\end{aligned}$$

Arbeitsweise einer Turing-Maschine



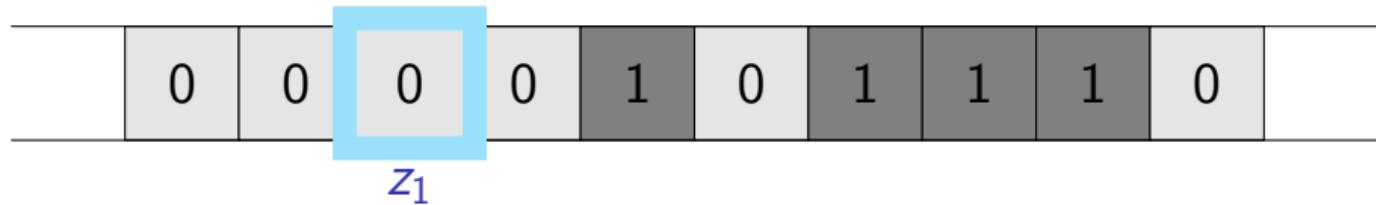
$$\begin{aligned}\delta(z_0, 0) &= (z_0, 0, R) \\ \delta(z_0, 1) &= (z_1, 0, L)\end{aligned}$$

Arbeitsweise einer Turing-Maschine



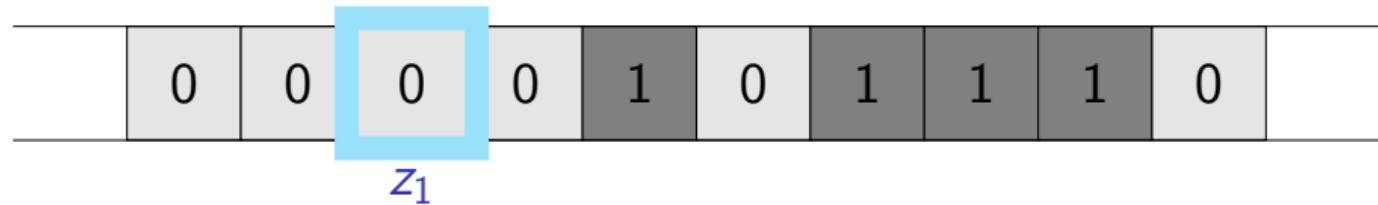
$$\begin{aligned}\delta(z_0, 0) &= (z_0, 0, R) \\ \delta(z_0, 1) &= (z_1, 0, L)\end{aligned}$$

Arbeitsweise einer Turing-Maschine



$$\begin{aligned}\delta(z_0, 0) &= (z_0, 0, R) \\ \delta(z_0, 1) &= (z_1, 0, L)\end{aligned}$$

Arbeitsweise einer Turing-Maschine

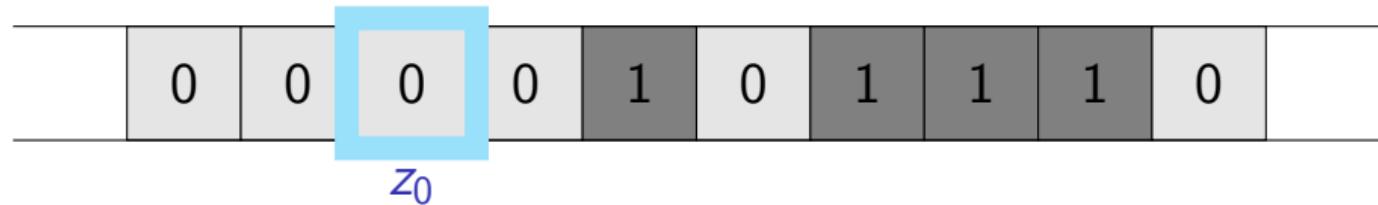


$$\delta(z_0, 0) = (z_0, 0, R)$$

$$\delta(z_0, 1) = (z_1, 0, L)$$

$$\delta(z_1, 0) = (z_0, 1, R)$$

Arbeitsweise einer Turing-Maschine

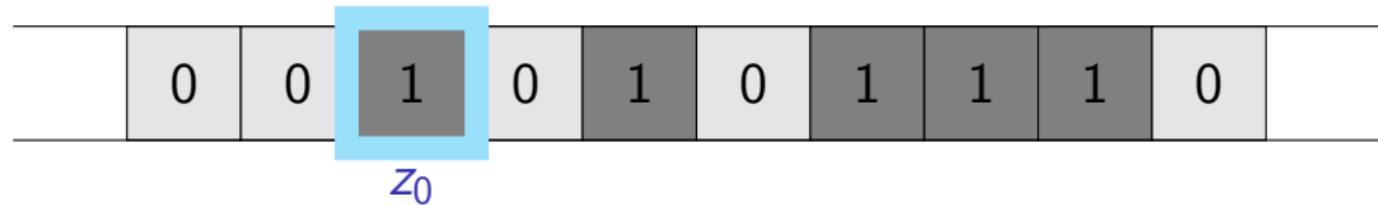


$$\delta(z_0, 0) = (z_0, 0, R)$$

$$\delta(z_0, 1) = (z_1, 0, L)$$

$$\delta(z_1, 0) = (z_0, 1, R)$$

Arbeitsweise einer Turing-Maschine

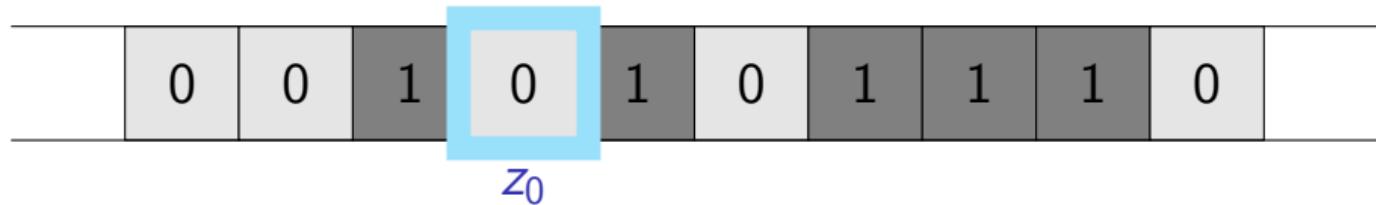


$$\delta(z_0, 0) = (z_0, 0, R)$$

$$\delta(z_0, 1) = (z_1, 0, L)$$

$$\delta(z_1, 0) = (z_0, 1, R)$$

Arbeitsweise einer Turing-Maschine

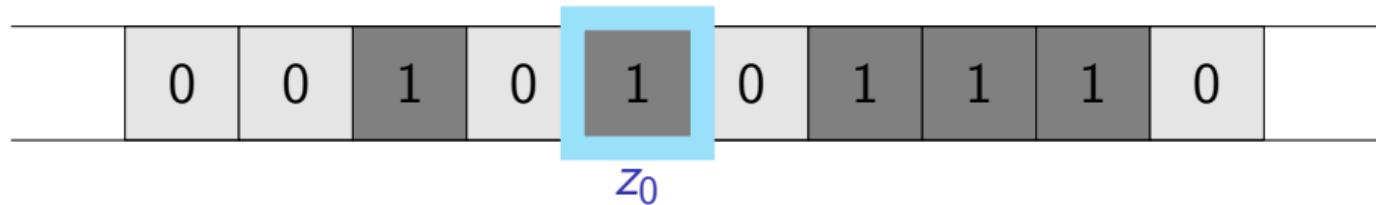


$$\delta(z_0, 0) = (z_0, 0, R)$$

$$\delta(z_0, 1) = (z_1, 0, L)$$

$$\delta(z_1, 0) = (z_0, 1, R)$$

Arbeitsweise einer Turing-Maschine

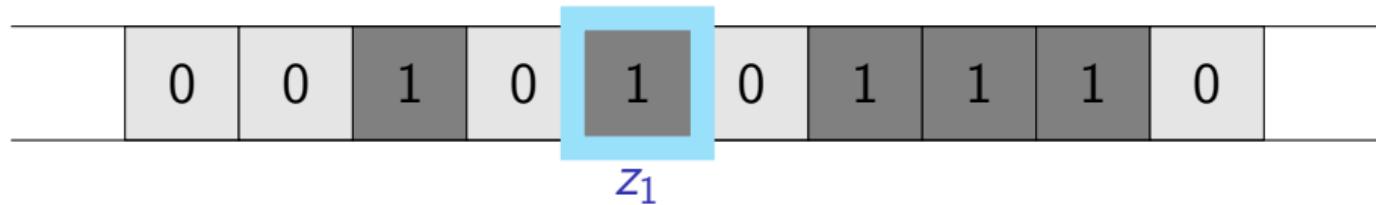


$$\delta(z_0, 0) = (z_0, 0, R)$$

$$\delta(z_0, 1) = (z_1, 0, L)$$

$$\delta(z_1, 0) = (z_0, 1, R)$$

Arbeitsweise einer Turing-Maschine

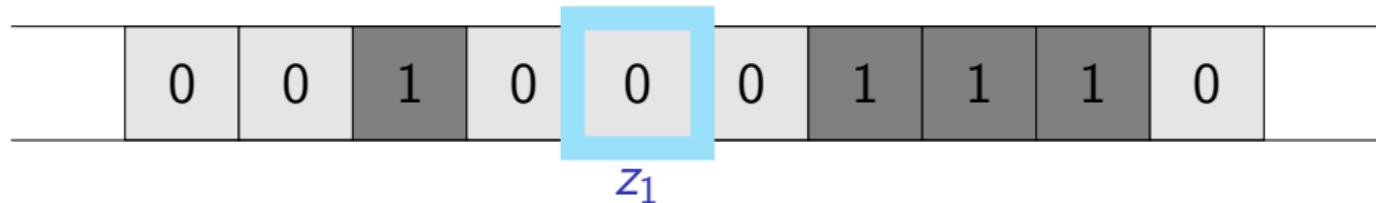


$$\delta(z_0, 0) = (z_0, 0, R)$$

$$\delta(z_0, 1) = (z_1, 0, L)$$

$$\delta(z_1, 0) = (z_0, 1, R)$$

Arbeitsweise einer Turing-Maschine

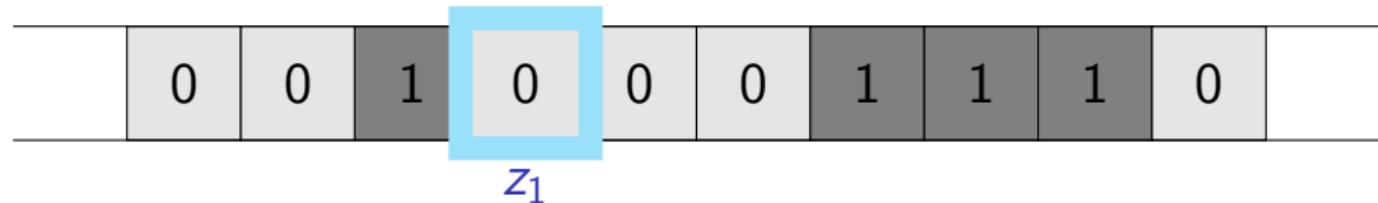


$$\delta(z_0, 0) = (z_0, 0, R)$$

$$\delta(z_0, 1) = (z_1, 0, L)$$

$$\delta(z_1, 0) = (z_0, 1, R)$$

Arbeitsweise einer Turing-Maschine

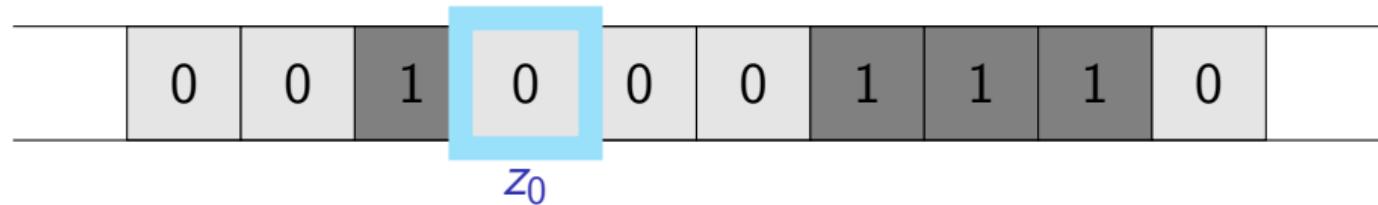


$$\delta(z_0, 0) = (z_0, 0, R)$$

$$\delta(z_0, 1) = (z_1, 0, L)$$

$$\delta(z_1, 0) = (z_0, 1, R)$$

Arbeitsweise einer Turing-Maschine

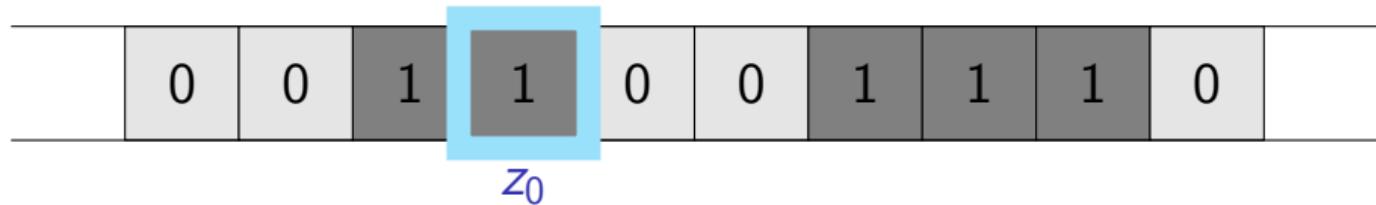


$$\delta(z_0, 0) = (z_0, 0, R)$$

$$\delta(z_0, 1) = (z_1, 0, L)$$

$$\delta(z_1, 0) = (z_0, 1, R)$$

Arbeitsweise einer Turing-Maschine

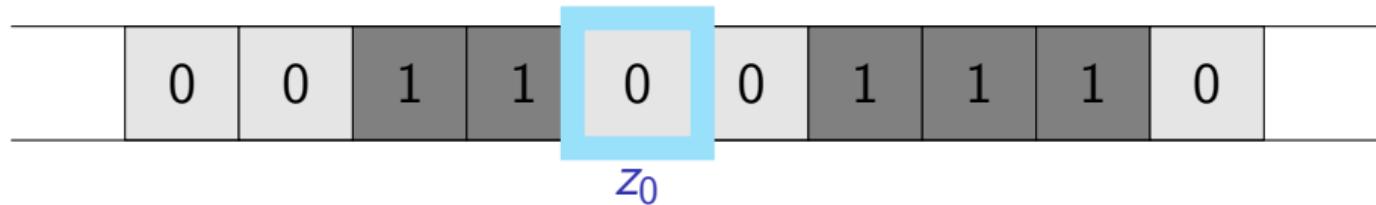


$$\delta(z_0, 0) = (z_0, 0, R)$$

$$\delta(z_0, 1) = (z_1, 0, L)$$

$$\delta(z_1, 0) = (z_0, 1, R)$$

Arbeitsweise einer Turing-Maschine

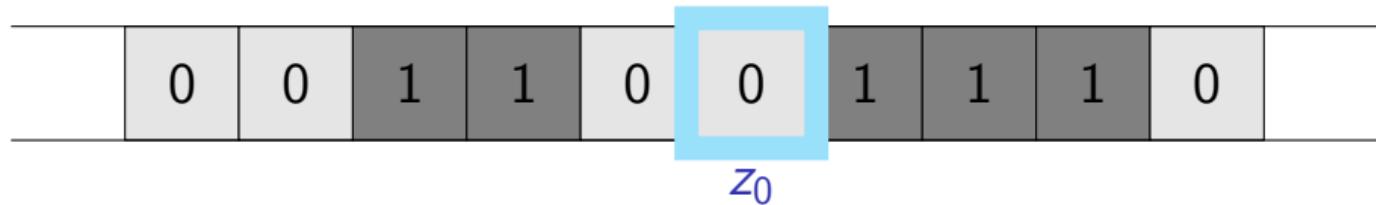


$$\delta(z_0, 0) = (z_0, 0, R)$$

$$\delta(z_0, 1) = (z_1, 0, L)$$

$$\delta(z_1, 0) = (z_0, 1, R)$$

Arbeitsweise einer Turing-Maschine



$$\delta(z_0, 0) = (z_0, 0, R)$$

$$\delta(z_0, 1) = (z_1, 0, L)$$

$$\delta(z_1, 0) = (z_0, 1, R)$$

Konfigurationen

Definition (Konfiguration, Folgekonfiguration)

Sei $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ eine TM. Eine **Konfiguration** von M ist ein Wort \underline{azb} mit $a, b \in \Gamma^*$ und $\underline{z} \in Z$.

Konfigurationen

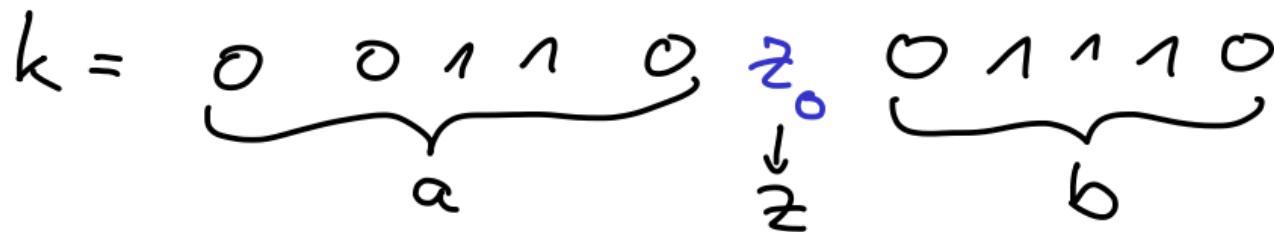
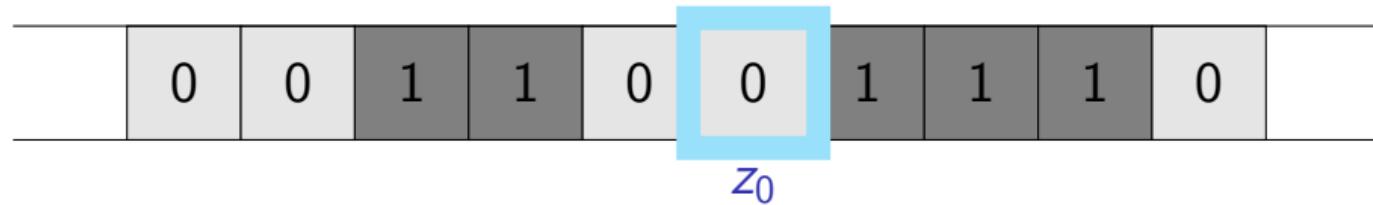
Definition (Konfiguration, Folgekonfiguration)

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)

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)



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 .

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 $\underline{n} = 0$, dann $\underline{b}_1 := \square$). Dann

Folgekonf. k
 $k \vdash_M^1 k'$

$k \vdash_M^0 k$

$a_1 \dots a_m z' c b_2 \dots b_n$

falls $\underline{\delta(z, b_1)} = (z', c, \underline{N})$

$k \vdash_M^1$

$a_1 \dots a_m c z' b_2 \dots b_n$

falls $\underline{\delta(z, b_1)} = (z', c, R)$

$k \vdash_M^1$

$a_1 \dots a_{m-1} z' a_m c b_2 \dots b_n$

falls $\underline{\delta(z, b_1)} = (z', c, \underline{L})$ und $m > 0$

$k \vdash_M^1$

$z' \square c b_2 \dots b_n$

falls $\underline{\delta(z, b_1)} = (z', c, \underline{L})$ und $m = 0$.

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

$$k \vdash_M^0 k$$

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

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

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

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

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

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

Weiter sei $\underline{k} \vdash_M^{i+1} \underline{k}' \iff \exists_q \underline{k} \vdash_M^1 \underline{q} \vdash_M^i \underline{k}'$ für alle i und $\underline{k} \vdash_M^* \underline{k}' \iff \exists_{i \in \mathbb{N}} \underline{k} \vdash_M^i \underline{k}'$

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 $k \vdash_M^{i+1} k' \iff \exists_q k \vdash_M^1 q \vdash_M^i k'$ für alle i und $k \vdash_M^* k' \iff \exists_{i \in \mathbb{N}} k \vdash_M^i k'$

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)

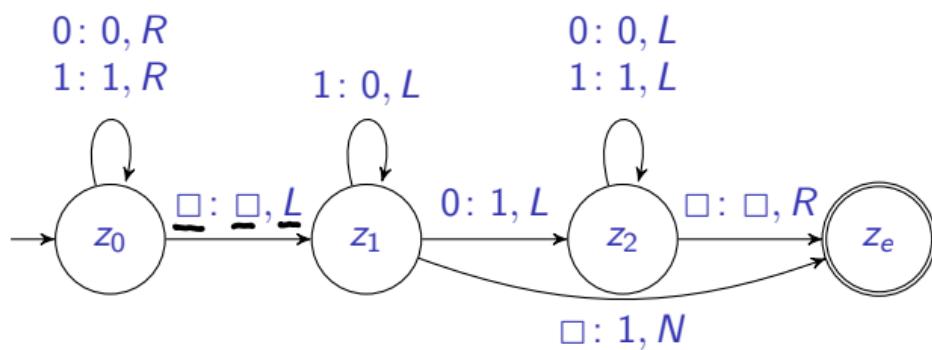
$\square 1011\square$
↓
 $\square 1100\square$

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

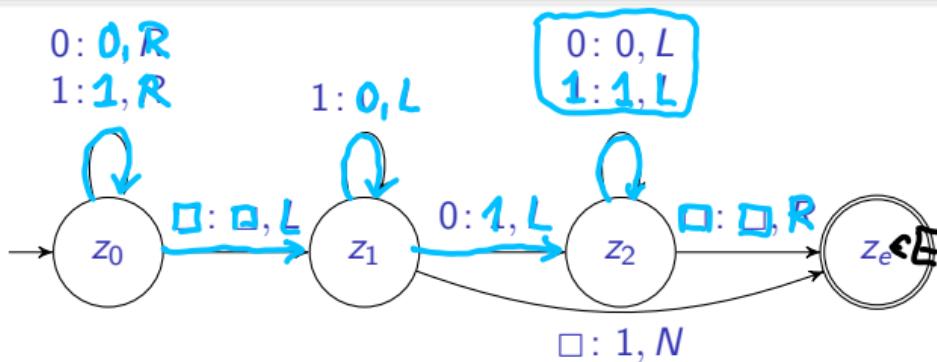


Beispiel Turing-Maschine: Binärzahl inkrementieren

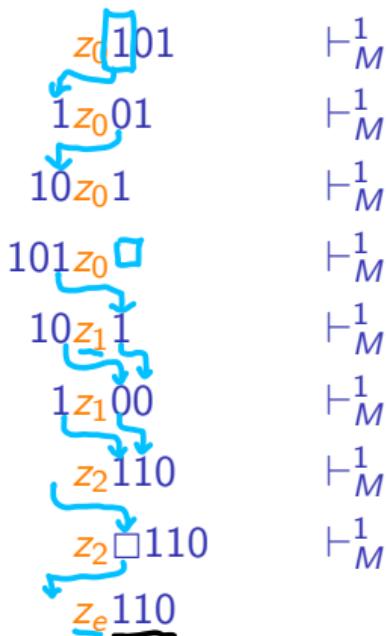
$$M = (\{z_0, z_1, z_2, z_e\}, \Sigma = \{0, 1\}, \Gamma = \{0, 1, \square\}, \delta, z_0, \square, \{z_e\})$$

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



Beispiel: Eingabe 101

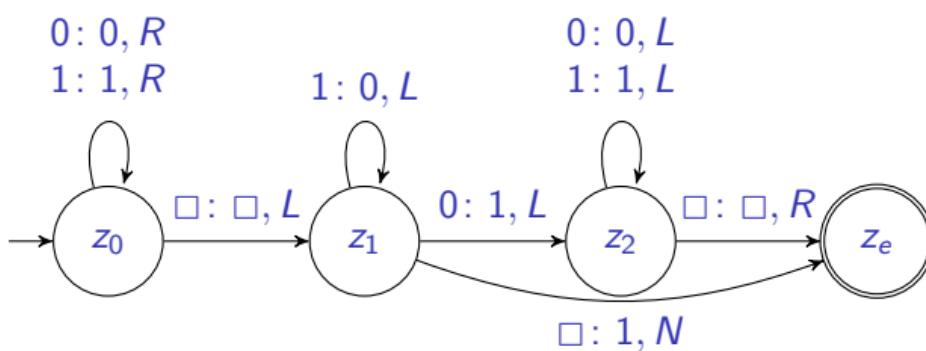


Beispiel Turing-Maschine: Binärzahl inkrementieren

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

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



Beispiel: Eingabe 101

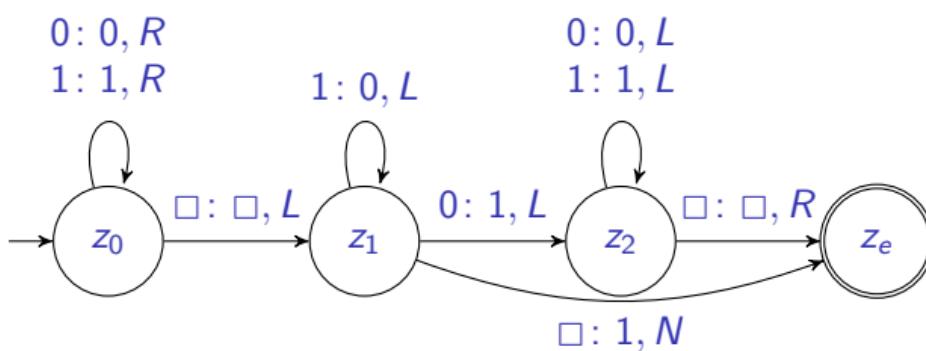
$z_0 101$	\vdash_M^1
$1 z_0 01$	\vdash_M^1
$10 z_0 1$	\vdash_M^1
$101 z_0$	\vdash_M^1
$101 z_1 1$	\vdash_M^1
$1 z_1 00$	\vdash_M^1
$z_2 110$	\vdash_M^1
$z_2 \square 110$	\vdash_M^1
$z_e 110$	
$z_e 110$ haltend & akzeptierend	

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

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

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

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

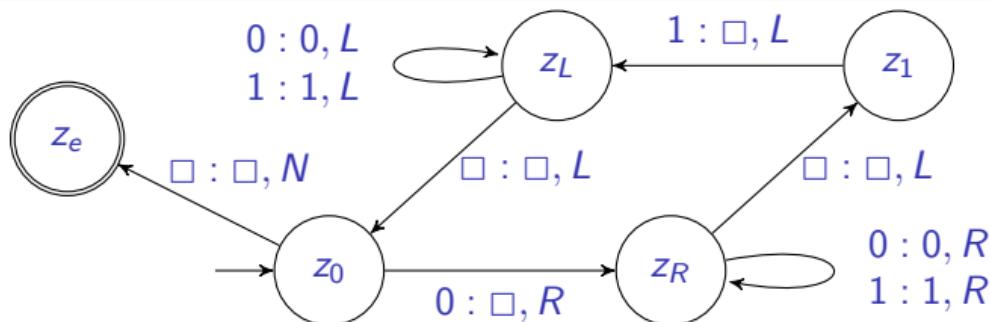
δ	0	1	\square
z_0	(z_R, \square, R)	\perp	(z_e, \square, N)
z_1	\perp	(z_L, \square, L)	\perp
z_R	$(z_R, 0, R)$	$(z_R, 1, R)$	(z_1, \square, L)
z_L	$(z_L, 0, L)$	$(z_L, 1, L)$	(z_0, \square, R)

Beispiel Turing-Maschine: akzeptiere $\{0^n 1^n \mid n \in \mathbb{N}\}$

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

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

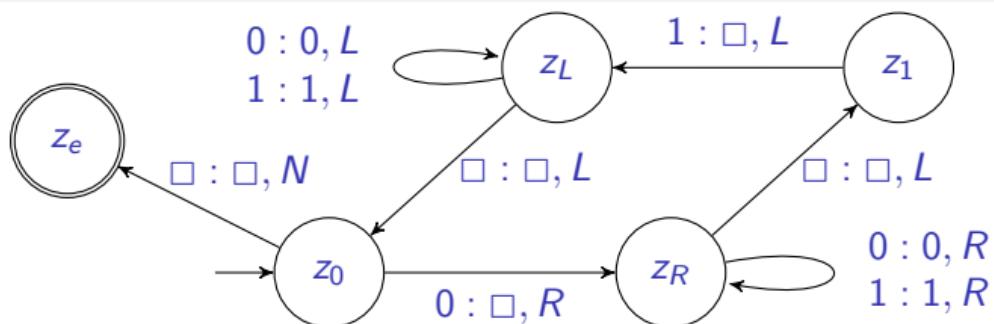


Beispiel Turing-Maschine: akzeptiere $\{0^n1^n \mid n \in \mathbb{N}\}$

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

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



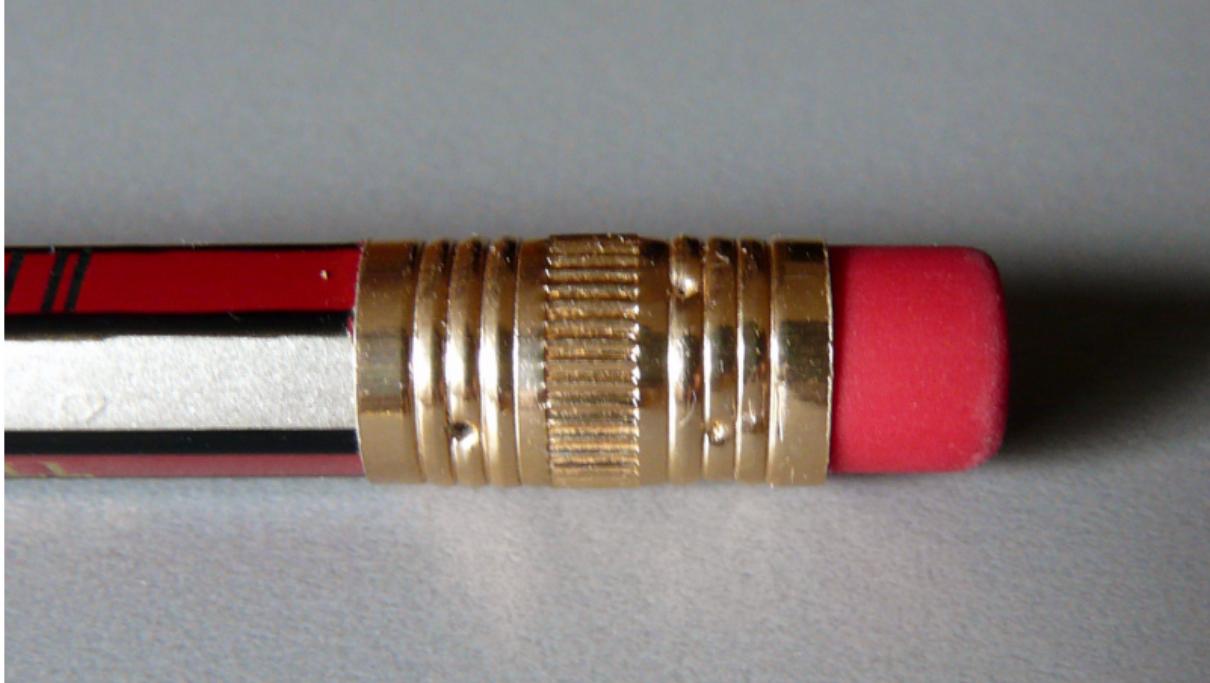
Beispiel: Eingabe 0011

z_0 0011	\vdash_M^1
0 z_R 11	\vdash_M^2
011 z_R	\vdash_M^1
01 z_1 1	\vdash_M^1
0 z_L 1	\vdash_M^2
z_L □ 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	\vdash_M^1

Gliederung

1. Einführung
1. Berechenbarkeitsbegriff
2. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Die Ackermannfunktion
6. (Un-)Entscheidbarkeit, Halteproblem und Reduzierbarkeit
7. Das Postsche Korrespondenzproblem
8. Komplexität – Einführung
9. NP-Vollständigkeit
10. PSPACE

Berechenbarkeitsbegriff



Ziel: Intuitiver Begriff \rightsquigarrow mathematische Formalisierung.

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

Diskussion intuitiver Berechenbarkeitsbegriff

(Intuitive) Berechenbarkeit von Funktionen:



Diskussion intuitiver Berechenbarkeitsbegriff

(Intuitive) Berechenbarkeit von Funktionen:

$$\{a^u b^u \mid u \geq 0\}$$



Diskussion intuitiver Berechenbarkeitsbegriff

(Intuitive) Berechenbarkeit von Funktionen:



► DFA \leadsto **endlicher Speicher** nicht "berechnungsmächtig" genug

Diskussion intuitiver Berechenbarkeitsbegriff

(Intuitive) Berechenbarkeit von Funktionen:



- ▶ DFA \rightsquigarrow **endlicher Speicher** nicht "berechnungsmächtig" genug
- ▶ Turing-Maschine

Diskussion intuitiver Berechenbarkeitsbegriff

(Intuitive) Berechenbarkeit von Funktionen:



- ▶ DFA \rightsquigarrow endlicher Speicher nicht "berechnungsmächtig" genug
- ▶ Turing-Maschine \rightsquigarrow Speichergröße unbeschränkt

Diskussion intuitiver Berechenbarkeitsbegriff

(Intuitive) Berechenbarkeit von Funktionen:



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

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

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
- ▶ ...

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

~~32 bit~~
~~64 bit~~

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?

Eingabe Längen $\rightarrow \leq 2n$ Zellen ; $\in n$?

Berechenbarkeitsbegriff

Definition

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

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

$$\iff$$

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

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

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

$$\iff$$

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

Bemerkung: existenzielle Aussage!

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

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

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 {}
```

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

$$\underbrace{f(n_1, \dots, n_k)}_{\Leftrightarrow} = m$$

falsch
↔
falsch

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 {}
```

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

Berechenbarkeitsbegriff - Beispiele

Beispiel 2

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

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

$314 \rightarrow 1$

$109 \rightarrow 0$

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

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

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

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

3, 1415

$n = 41$

$n = 926$

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

?

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

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

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 - 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

- approximiere π "ausreichend genau"
- vergleiche mit Eingabe

12 1en in π : ~~$f(n)$~~ $\overline{1|2|3|\dots|12|13|\dots}$

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 II

Problem: Berechenbarkeitsbegriff basiert auf Definition von “Algorithmus” . . .

Berechenbarkeitsbegriff V

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

Church'sche **These**

Intuitive Berechenbarkeit = Turing-Berechenbarkeit

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

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 **berechenbar**, wenn es einen endlichen Algorithmus \mathcal{A} gibt, sodass für alle $n_1, \dots, n_k, m \in \mathbb{N}$ gilt
 $f(n_1, \dots, n_k) = m \iff$ bei Eingabe (n_1, \dots, n_k) hält \mathcal{A} nach endlicher Zeit mit Ausgabe m .

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
 $f(n_1, \dots, n_k) = m \iff$ bei Eingabe (n_1, \dots, n_k) hält M nach endlicher Zeit mit Ausgabe m .

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
$$f(n_1, \dots, n_k) = m \iff \text{bei Eingabe } (n_1, \dots, n_k) \text{ hält } M \text{ nach endlicher Zeit mit Ausgabe } m.$$
$$\iff \exists_{z \in E} \underbrace{z_0 \text{ BIN}(n_1) \# \dots \# \text{BIN}(n_k)}_{\# \in \Sigma} \vdash_M^* \underbrace{z \text{ BIN}(m)}_{\# \in \Sigma}$$
wobei **BIN** zahlen auf ihre Binärdarstellung abbildet.

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
$$f(n_1, \dots, n_k) = m \iff \text{bei Eingabe } (n_1, \dots, n_k) \text{ hält } M \text{ nach endlicher Zeit mit Ausgabe } m.$$
$$\iff \exists_{z \in E} z_0 \text{ BIN}(n_1) \# \dots \# \text{BIN}(n_k) \vdash_M^* z \text{ BIN}(m)$$
wobei **BIN** Zahlen auf ihre Binärdarstellung abbildet.

- Eine (eventuell partielle) Funktion $f: \underline{\Sigma^*} \rightarrow \Sigma^*$ 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

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

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
$$f(n_1, \dots, n_k) = m \iff \text{bei Eingabe } (n_1, \dots, n_k) \text{ hält } M \text{ nach endlicher Zeit mit Ausgabe } m.$$
$$\iff \exists_{z \in E} z_0 \text{ BIN}(n_1) \# \dots \# \text{BIN}(n_k) \vdash_M^* z \text{ BIN}(m)$$
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

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

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

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
$$f(n_1, \dots, n_k) = m \iff \text{bei Eingabe } (n_1, \dots, n_k) \text{ hält } M \text{ nach endlicher Zeit mit Ausgabe } m.$$
$$\iff \exists_{z \in E} z_0 \text{ BIN}(n_1)\# \dots \# \text{BIN}(n_k) \vdash_M^* z \text{ BIN}(m)$$
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
$$f(x) = y \iff \exists_{z \in E} z_0 x \vdash_M^* z y$$

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

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 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
$$f(n_1, \dots, n_k) = m \iff \text{bei Eingabe } (n_1, \dots, n_k) \text{ hält } M \text{ nach endlicher Zeit mit Ausgabe } m.$$
$$\iff \exists_{z \in E} z_0 \text{ BIN}(n_1)\# \dots \# \text{BIN}(n_k) \vdash_M^* z \text{ BIN}(m)$$
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
$$f(x) = y \iff \exists_{z \in E} z_0 x \vdash_M^* z y$$

- 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?

Turing-Berechenbarkeit - Beispiele

Turing-berechenbar?

1. Nachfolgerfunktion $\text{succ}: \mathbb{N} \rightarrow \mathbb{N}, n \mapsto n + 1$

Turing-Berechenbarkeit - Beispiele

Turing-berechenbar?

1. Nachfolgerfunktion $\text{succ}: \mathbb{N} \rightarrow \mathbb{N}, n \mapsto n + 1$



Turing-Berechenbarkeit - Beispiele

Turing-berechenbar?

1. Nachfolgerfunktion $\text{succ}: \mathbb{N} \rightarrow \mathbb{N}, n \mapsto n + 1$
2. nirgends definierte „Funktion“ Ω



Turing-Berechenbarkeit - Beispiele

Turing-berechenbar?

1. Nachfolgerfunktion $\text{succ}: \mathbb{N} \rightarrow \mathbb{N}, n \mapsto n + 1$
2. nirgends definierte „Funktion“ Ω

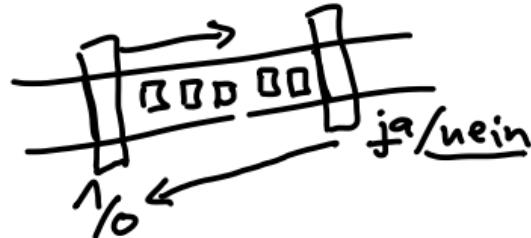
✓

✓

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?



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? ✓

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 $L = \{0^n 1^n \mid n \in \mathbb{N}\}$? ✓

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 $L = \{0^n 1^n \mid n \in \mathbb{N}\}$? ✓

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 $L = \{0^n1^n \mid n \in \mathbb{N}\}$?

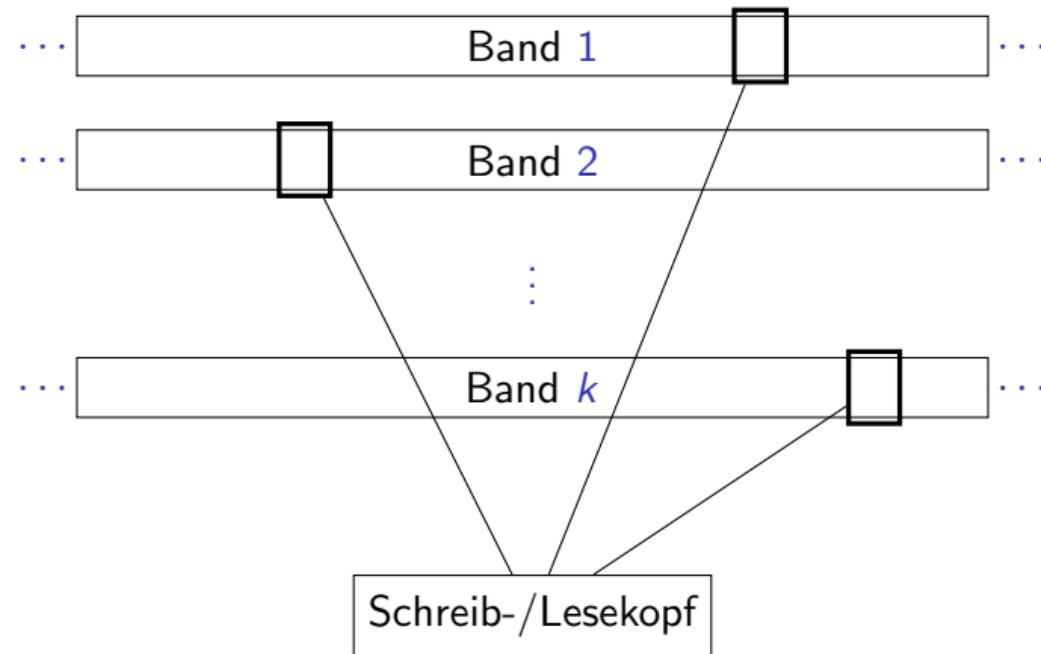
✓ ✓ ✓ ✓

Turing

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

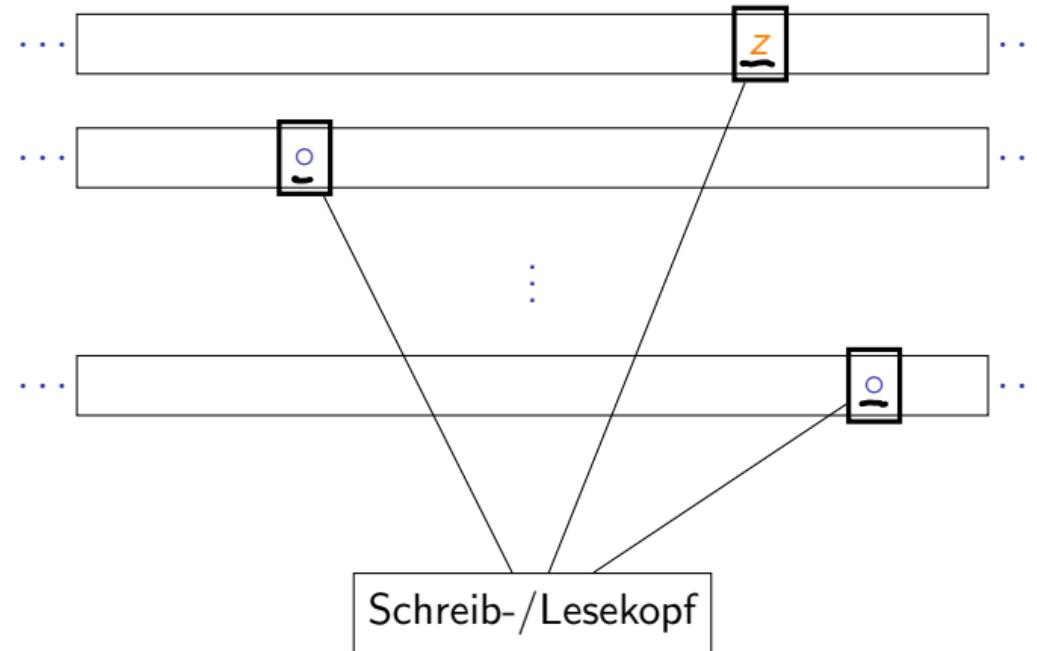
Mehrband-Turing-Maschinen I

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



Mehrband-Turing-Maschinen I

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



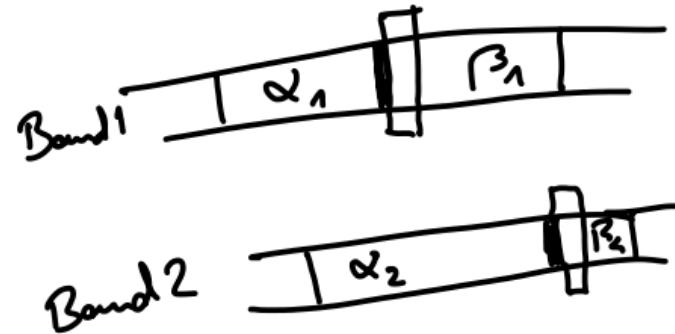
(Deterministische) k -Band Turing-Maschine M

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

(Deterministische) k -Band Turing-Maschine M

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

Konfiguration $\underline{\alpha_1 z \beta_1}, \underline{\alpha_2 \dots \beta_2}, \dots, \underline{\alpha_k \dots \beta_k}$



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

(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 $\underline{z_0} \underline{x_1}, \underline{\circ} \underline{x_2}, \dots, \underline{\circ} \underline{x_k}$

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

(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...

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

(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

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

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

(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

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

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

Mehrband-TM Äquivalenz

Theorem

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

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“:



$$\begin{aligned} (a,a) &\in T_Q \\ (b,a) &\in T_Q \end{aligned}$$



$$T_Q = T_M^k$$

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“:

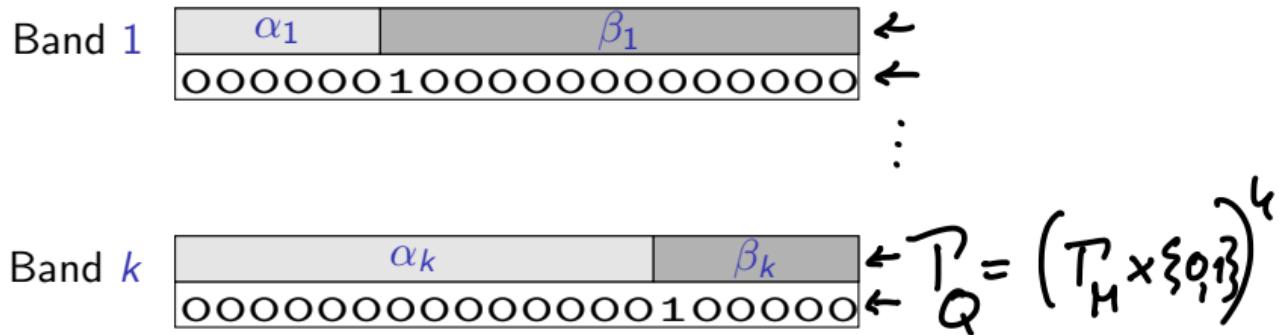


Mehrband-TM Äquivalenz

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“:

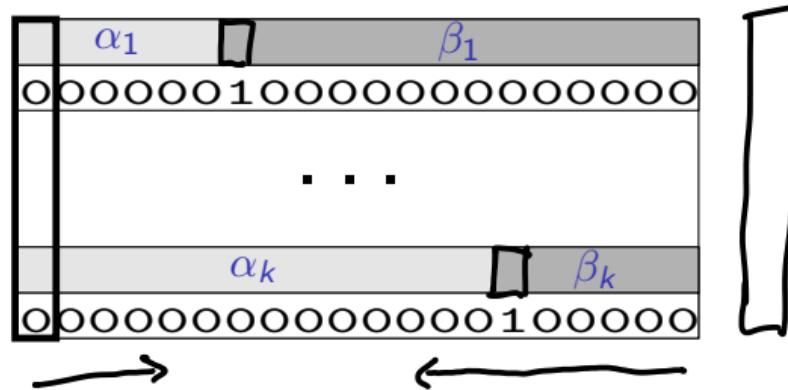


Mehrband-TM Äquivalenz

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 \underline{M} mithilfe eines „fetten Bandes“ mit $2k$ „Spuren“:



Mehrband-TM Äquivalenz

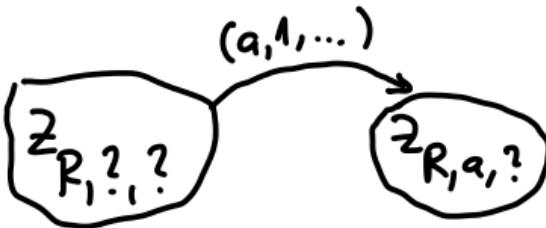
Theorem

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“:

$\exists_{R,x,y} \quad x,y \in T \cup \{?\}$

α_1	a	β_1
oooooooooooo	1	oooooooooooooooooooo
		...
	α_k	β_k
oooooooooooo	oooooooooooo	1oooooooooooo



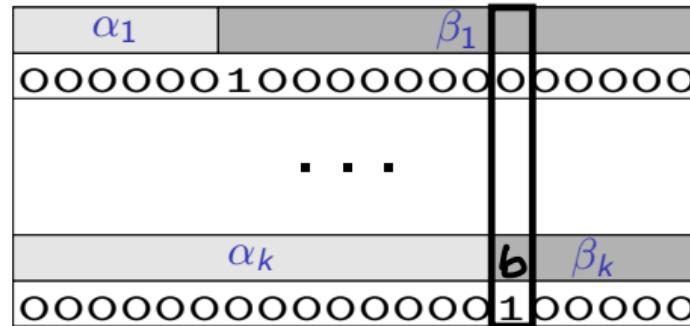
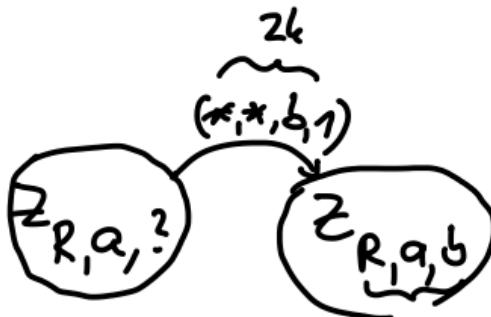
“speichere” das ersten Zeichen $\underline{\beta_i[0]} \in \Gamma$ von $\underline{\beta_i}$ für alle $i \leq k$ im Zustand \underline{Z}

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“:



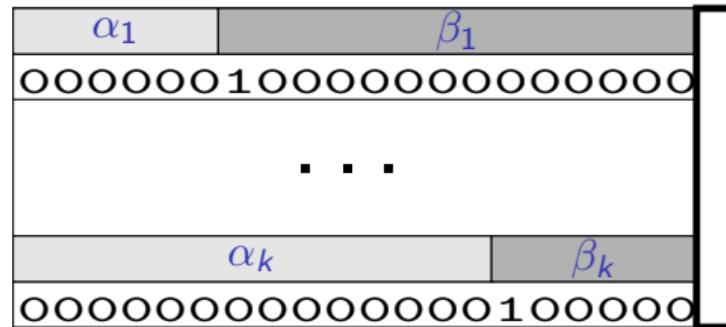
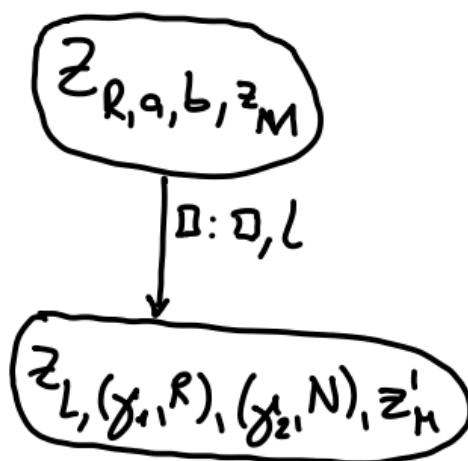
„speichere“ das ersten Zeichen $\beta_i[0] \in \Gamma$ von β_i für alle $i \leq k$ im Zustand

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“:



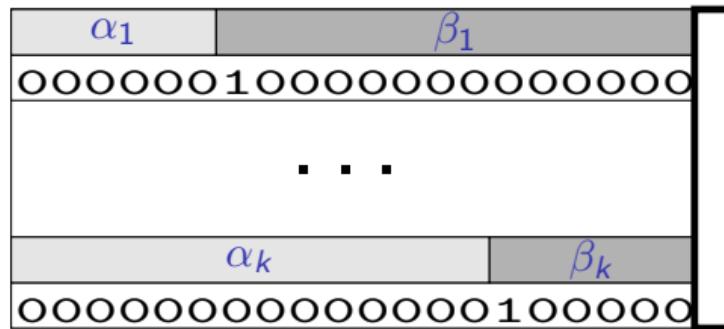
$$\sum_{\beta_i} \left(\sum_{\alpha_i} E \right) \times T_M^k \rightarrow \sum_M \left(P_N \times (L, R, N) \right)^k$$

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

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

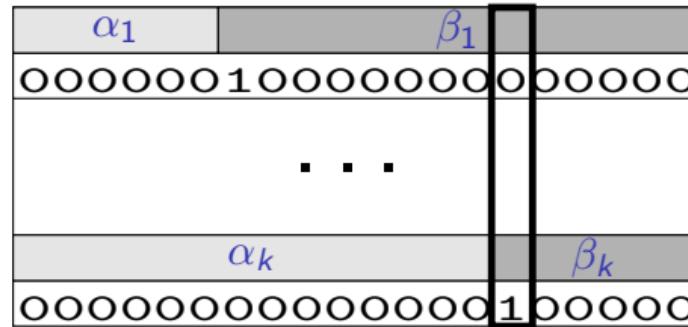
„speichere“ neues Zeichen $\underline{\gamma_i} \in \Gamma$ & Kopfrichtung $\underline{d_i} \in \{L, R, N\}$ für alle $i \leq k$ im Zustand

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

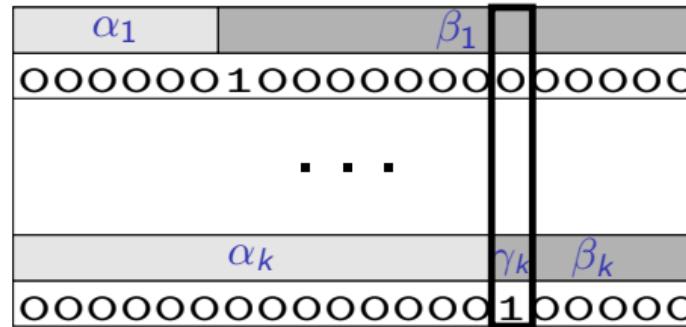
„speichere“ neues Zeichen $\gamma_i \in \Gamma$ & Kopfrichtung $d_i \in \{L, R, N\}$ für alle $i \leq k$ im Zustand

Mehrband-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

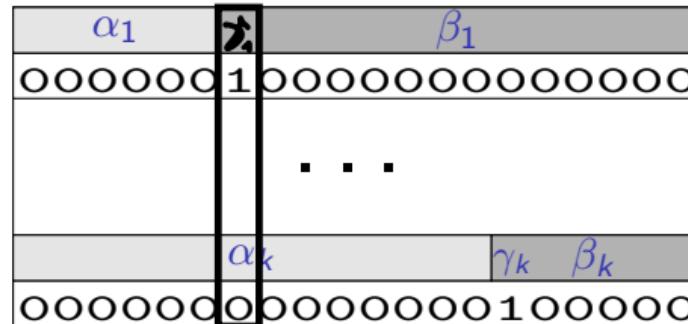
„speichere“ neues Zeichen $\gamma_i \in \Gamma$ & Kopfrichtung $d_i \in \{L, R, N\}$ für alle $i \leq k$ im Zustand

Mehrband-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“:



$z_{L,(\gamma_1,R),..}$

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

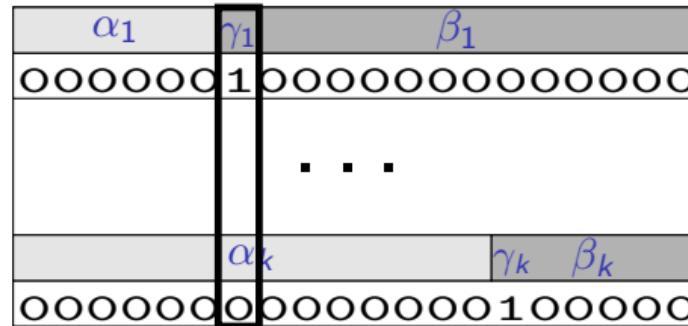
„speichere“ neues Zeichen $\gamma_i \in \Gamma$ & Kopfrichtung $d_i \in \{L, R, N\}$ für alle $i \leq k$ im Zustand

Mehrband-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

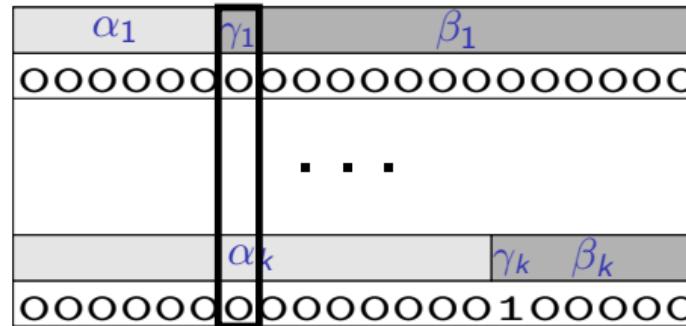
„speichere“ neues Zeichen $\gamma_i \in \Gamma$ & Kopfrichtung $d_i \in \{L, R, N\}$ für alle $i \leq k$ im Zustand

Mehrband-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

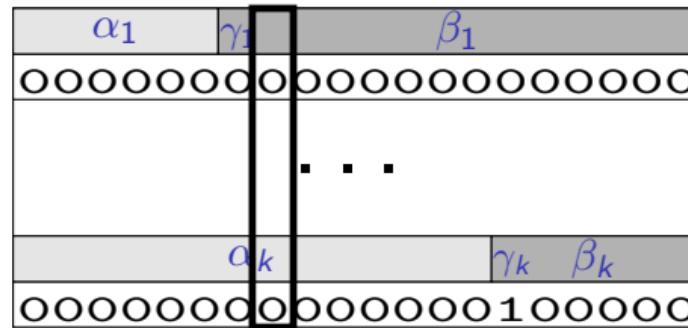
„speichere“ neues Zeichen $\gamma_i \in \Gamma$ & Kopfrichtung $d_i \in \{L, R, N\}$ für alle $i \leq k$ im Zustand

Mehrband-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

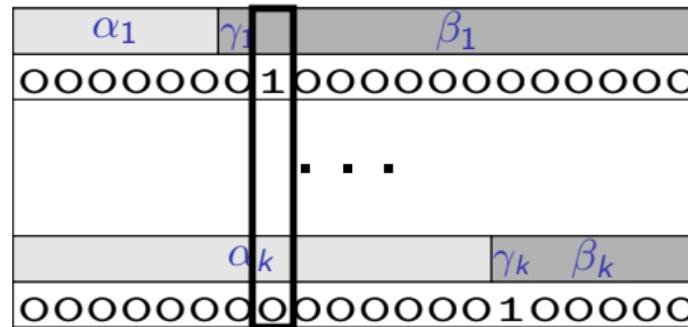
„speichere“ neues Zeichen $\gamma_i \in \Gamma$ & Kopfrichtung $d_i \in \{L, R, N\}$ für alle $i \leq k$ im Zustand

Mehrband-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

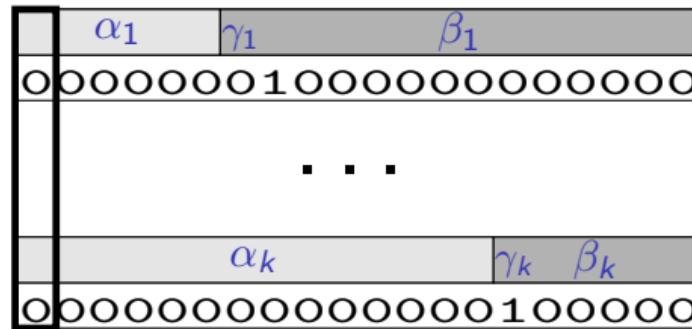
„speichere“ neues Zeichen $\gamma_i \in \Gamma$ & Kopfrichtung $d_i \in \{L, R, N\}$ für alle $i \leq k$ im Zustand

Mehrband-TM Äquivalenz

Theorem

Zu jeder k -Band-TM \underline{M} gibt es eine Einband-TM \underline{Q} mit $T(\underline{M}) = T(\underline{Q})$ (bzw. $f_{\underline{M}} = f_{\underline{Q}}$).

Beweisidee: \underline{Q} simuliert \underline{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

LOOP-, WHILE-, und GOTO-Berechenbarkeit



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

LOOP-Programme I

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

Variablen: $\underline{x_0}, \underline{x_1}, \dots$

Konstanten: $\underline{0}, \underline{1}, \underline{2}, \dots$

Trennsymbole: $\underline{:}, \underline{:=}$

Operationen: $\underline{+}, \underline{-}$

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

LOOP-Programme I

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

Variablen: $x_0, x_1, \dots \in \mathbb{N}$

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

Konstanten: 0, 1, 2, ...

Berechnungsergebnis: Wert von x_0 am Programmende

Trennsymbole: ;, :=

Operationen: +, -

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

LOOP-Programme I

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

Variablen: $x_0, x_1, \dots \in \mathbb{N}$

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

Konstanten: 0, 1, 2, ...

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 $x_2 := x_1 + 5$
$x_i := x_j - c$ (x_i, x_j Variablen, $c \in \mathbb{N}$)	Modifizierte Subtraktion $x_2 := x_1 - 5$ $\max\{x_j - c, 0\}$
$P_1; P_2$ (P_1, P_2 LOOP Programme)	Sequenz erst P_1 , dann P_2
LOOP x_i DO <u>P</u> END (x_i Variable, P LOOP Programm)	<u>Schleife</u> #Durchläufe = Wert von x_i vor der Anweisung!

LOOP-Programme II

Simulation anderer Rechenoperationen durch LOOP-Programme:

$x_i := x_j$	$x_i := x_j + 0$
$x_i := c$	$x_i := \underline{x_j} + c$ (für ein x_j mit $x_j = 0$)
IF $x_i = 0$ THEN <u>P</u> END	<u>$x_j := 1;$</u> <u>LOOP</u> x_i DO <u>$x_j := 0$</u> END; <u>LOOP</u> x_j DO <u>P</u> END
$x_0 := x_1 + x_2$	$x_0 := x_1;$ LOOP x_2 DO $x_0 := x_0 + 1$ END

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	$x_j := 1;$ $\text{LOOP } x_i \text{ DO } x_j := 0 \text{ END};$ $\text{LOOP } x_j \text{ DO } P \text{ END}$
$x_0 := x_1 + x_2$	$x_0 := x_1;$ $\text{LOOP } x_2 \text{ DO } x_0 := x_0 + 1 \text{ END}$

Analog: Multiplikation, ganzzahlige Division, Modulo.

$x_k := x_i \cdot x_j$ $x_k := 0 ;$
 $\text{Loop } x_i \text{ DO } x_k := x_k + x_j \text{ END}$

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

WHILE-Programme

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

Syntax

WHILE $x_i \neq 0$ **DO** P **END**

(x_i Variable, P WHILE Programm) • wiederhole P bis $x_i = 0$; x_i darf durch P modifiziert werden!

Semantik

While Schleife

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

$x_1 := 1$;

WHILE $x_1 \neq 0$ **DO** $x_1 := 1$ **END**

WHILE-Programme

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

Syntax

WHILE $x_i \neq 0$ **DO** P **END**

(x_i Variable, P WHILE Programm)

Semantik

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

$$\underline{f(n_1, \dots, n_k) = m \iff P \text{ hält bei Eingabe } n_1, \dots, n_k \text{ mit Berechnungsergebnis } x_0 = m}$$

wenn ein allwissendes Oracle
 P angeben kann

WHILE-Programme

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

Loop x_i DO P END
=

Syntax

WHILE $x_i \neq 0$ **DO** P **END**

(x_i Variable, P WHILE Programm)

Semantik

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

$$f(n_1, \dots, n_k) = m \iff P \text{ hält bei Eingabe } n_1, \dots, n_k \text{ mit Berechnungsergebnis } x_0 = m$$



Beobachtung: LOOP-Programme berechnen nur totale Funktionen.

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
 $f(n_1, \dots, n_k) = m \iff P$ 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?** **NEIN**
WHILE

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
 $f(n_1, \dots, n_k) = m \iff P$ 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**.

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
 $f(n_1, \dots, n_k) = m \iff P$ 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.

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 :

Annahme: $M(P)$ hat „ausreichend viele“ Bänder
(für jede Variable ein eigenes Band)

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

1. x_j nach x_i kopieren
Inhalt v. Band j auf Band i kopieren
2. c mal Inkrementierer auf Band i laufen lassen

WHILE-Programme & Turing-Maschinen

Theorem

Jede WHILE-berechenbare Funktionen ist Turing-berechenbar.

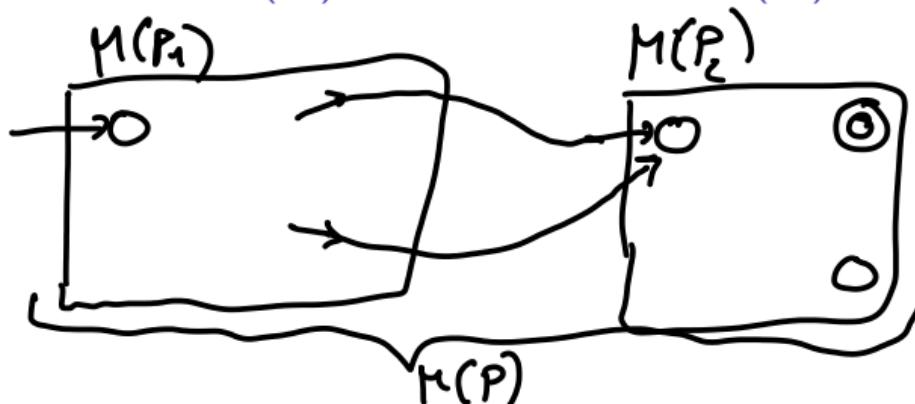
Beweis (Skizze)

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: $P = \underline{P_1}; \underline{P_2}$ \rightsquigarrow Hintereinanderschaltung von $M(P_1)$ und $M(P_2)$

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



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

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 $M(P_2)$

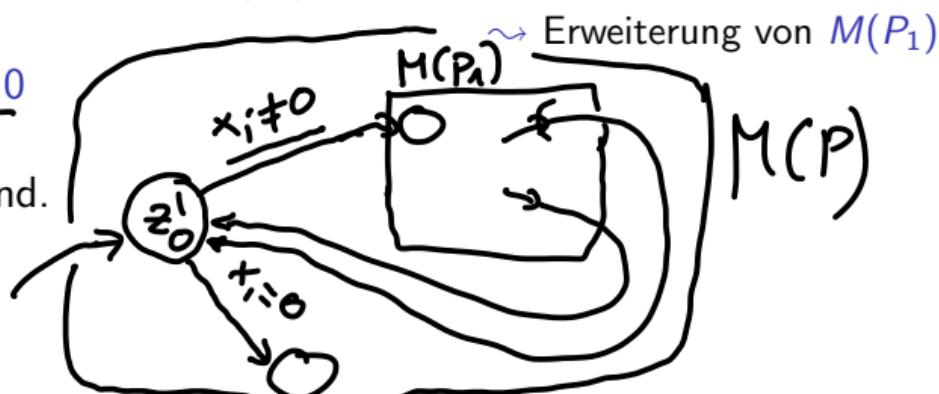
Identifizierte Endzustände von $M(P_1)$ mit Startzustand von $M(P_2)$.

Fall 3: $P = \text{WHILE } x_i \neq 0 \text{ DO } P_1 \text{ END}$

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

ja \rightsquigarrow Wechsel in Startzustand von $M(P_1)$

nein \rightsquigarrow Stoppe in einem neuen Endzustand.



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:

$$P = \underline{M_1} : \underline{A_1};$$

$$M_2 : A_2;$$

⋮

$$M_k : A_k$$

Konvention: Nicht benutzte Marken weglassen!

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!

Syntax: Mögliche Anweisungen A_i :

► $x_i := x_j \pm c$

► **GOTO** M_i

► **IF** $x_i = c$ **THEN GOTO** M_i

Semantik klar!

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.

Syntax: Mögliche Anweisungen A_i :

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

Semantik klar!

GOTO-Programme I

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

$P = M_1 : A_1;$

$M_2 : A_2;$

\vdots

$M_k : A_k$

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!

WHILE ...

WHILE ...

END
END

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)

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

\vdots
 $M_3 : \underline{P};$

$M_4 : \text{GOTO } M_1;$

$M_2 : \underline{x_i := 0}$

GOTO-Programme II

Theorem

Jede GOTO-berechenbare Funktion ist WHILE-berechenbar.

GOTO-Programme II

Theorem

Jede GOTO-berechenbare Funktion ist WHILE-berechenbar.

nächste Sprunganrede

Beweis

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



GOTO-Programme II

Theorem

Jede GOTO-berechenbare Funktion ist WHILE-berechenbar.

Beweis

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

i $\rightsquigarrow A_i$
 \leadsto Loop-Berechnung der
IF $x_i = 0$ **THEN**
frage überlegen Sie sich, dass
IF $x_i = c$ **THEN** Loop ber.

GOTO-Programme II

Theorem

Jede GOTO-berechenbare Funktion ist WHILE-berechenbar.

Beweis

$P = M_1 : A_1; \dots; M_k : A_k$ 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-Programme II

Theorem

Jede GOTO-berechenbare Funktion ist WHILE-berechenbar.

Beweis

$P = M_1 : A_1; \dots; M_k : A_k$ 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**;

\dots

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	<u>$x_N := j$</u>

GOTO-Programme II

Theorem

Jede GOTO-berechenbare Funktion ist WHILE-berechenbar.

Beweis

$P = M_1 : A_1; \dots; M_k : A_k$ 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 $\underline{x_N := n}$

GOTO-Programme II

Theorem

Jede GOTO-berechenbare Funktion ist WHILE-berechenbar.

Beweis

$P = M_1 : A_1; \dots; M_k : A_k$ 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

WHILE berechenbar



Theorem

Jede Turing-berechenbare Funktion ist GOTO-berechenbar.

GOTO-Programme III

Theorem

Jede Turing-berechenbare Funktion ist GOTO-berechenbar.

Beweis (Skizze)

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

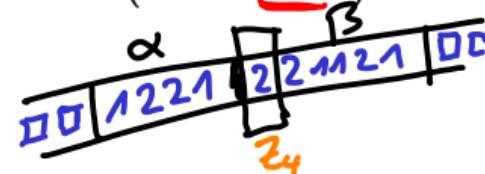
Idee: Darstellen der Konfiguration $\alpha z \beta$ als drei Zahlen (wobei $\square \triangleq 0$):

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

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

$x_3 = z = \ell \rightsquigarrow$ Zustandsnummer

$$\downarrow \quad \underbrace{\subseteq \mathbb{N}}$$



$$T = \{0, 1, 2\}$$

$$x = [1221]_3 = 52$$

$$y = [121122]_3 = 449 \quad z = 4$$

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 $\alpha z_\ell \beta$ 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:

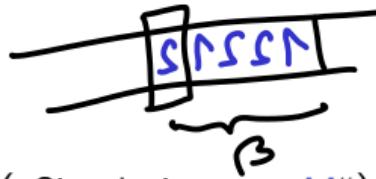
Phase 1: „Erzeugung von x, y, z aus der Startkonfiguration“ (LOOP-berechenbar)

Phase 2: „Simulation von M “ 

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

GOTO-Programme IV

Beweis (Fortsetzung)



$$y = [12212]_3$$

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

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

Beweis (Fortsetzung)

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

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

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

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

...

IF $\underline{z = k}$ AND $\underline{x_4 = m - 1}$ **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 ;

GOTO-Programme IV

Beweis (Fortsetzung)

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

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

$y = 112k_1$

IF $z = 1$ AND $x_4 = 0$ **THEN GOTO** $M_{(1,0)}$; $x = 122$

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

...

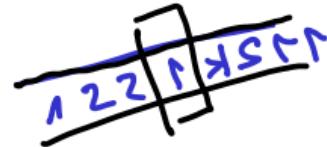
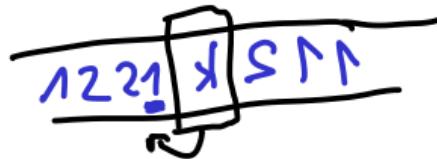
IF $z = k$ AND $x_4 = m - 1$ **THEN GOTO** $M_{(k,m-1)}$;

$M_{(1,0)}$: 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 $\underline{\delta}$

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

$\overline{1} \overline{2} \overline{3}$

① $z := j;$

② $y := \underline{y \text{ DIV } m};$

$y := \underline{y \cdot m + k};$

③ $y := y \cdot m + (\underline{x \text{ MOD } m});$

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

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

$\rightarrow \delta(\underline{z_\ell}, i) = \perp \rightsquigarrow \text{Endlosschleife}$

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.

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

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 $f : \underline{\mathbb{N}^k} \rightarrow \underline{\mathbb{N}}$ mit $f(x_1, \dots, x_k) = \underline{c} \in \mathbb{N}$

$O_1 \dots$ die eine
stellige
Nullfunktion

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 $\underline{\text{succ}}(\underline{n}) = \underline{n + 1}$

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 $\pi_i^k : \underline{\mathbb{N}}^k \rightarrow \underline{\mathbb{N}}$ mit $\pi_i^k(x_1, \dots, x_k) = x_i$

$$\pi_{\textcircled{i}}^k(x_1, \dots, x_k) = x_1$$

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:

Primitiv-rekursive Funktionen I

Definition

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

(a) folgende **Grundfunktionen** enthält:

- alle konstanten Funktionen $f : \mathbb{N}^k \rightarrow \mathbb{N}$ mit $f(x_1, \dots, x_k) = c$
- die Nachfolgerfunktion $\text{succ} : \mathbb{N} \rightarrow \mathbb{N}$ mit $\text{succ}(n) = n + 1$
- 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 $\underline{g_1, \dots, g_m} : \underline{\mathbb{N}^k} \rightarrow \underline{\mathbb{N}}$ und $\underline{h} : \underline{\mathbb{N}^m} \rightarrow \underline{\mathbb{N}}$:

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

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

$$f = \underbrace{\text{succ} \circ \pi_2^2}_{\begin{array}{c} h \\ \uparrow \\ g_1 \end{array}} \quad \begin{array}{l} "h \text{ nach } \underline{g_1 \dots g_m}" \\ = x_2 \end{array}$$
$$f(x_1, x_2) = \text{succ}(\overbrace{\pi_2^2(x_1, x_2)}^{= x_2}) = \underline{\text{succ}(x_2)}$$

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 $g : \underline{\mathbb{N}^k} \rightarrow \mathbb{N}$ und $h : \underline{\mathbb{N}^{k+2}} \rightarrow \mathbb{N}$:

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

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

Rekursion * $f(\underline{n+1}, x_1, \dots, x_k) = h(\underline{n}, \underline{f(n, x_1, \dots, x_k)}, x_1, \dots, x_k).$

Primitiv-rekursive Funktionen II

Erinnerung: primitive Rekursion

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

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

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

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

Beispiel 1 – add : $\mathbb{N}^2 \rightarrow \mathbb{N}$ $\text{add}(x, y) = x + y$

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

$$\underbrace{\text{h} : \mathbb{N}^3 \rightarrow \mathbb{N}}_{\downarrow} \quad \underbrace{\text{g} : \mathbb{N} \rightarrow \mathbb{N}}$$

Primitiv-rekursive Funktionen II

Erinnerung: primitive Rekursion

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

* $f(0, \underline{x_1}, \dots, x_k) = g(\underline{x_1}, \dots, x_k)$

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

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

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

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

Rek $\text{add}(\underline{n+1}, x) = (\underline{\text{succ}} \circ \underline{\pi_2^3})(\underline{n}, \underline{\text{add}(n, x)}, x)$
 $= \text{succ}(\underline{\text{add}(n, x)})$

Primitiv-rekursive Funktionen II

Erinnerung: primitive Rekursion

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

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

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

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

add := $\text{pr}(\text{succ} \circ \pi_2^3, \pi_1^1)$ 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 : $\mathbb{N}^2 \rightarrow \mathbb{N}$ $\text{mul}(x, y) = x \cdot y$

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

$$\begin{array}{l}g : \mathbb{N} \rightarrow \mathbb{N} \\ g(x) = 0\end{array}$$

Primitiv-rekursive Funktionen II

Erinnerung: primitive Rekursion

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

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

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

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

add := $\text{pr}(\text{succ} \circ \pi_2^3, \pi_1^1)$ 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 : $\mathbb{N}^2 \rightarrow \mathbb{N}$

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

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

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

Primitiv-rekursive Funktionen II

Erinnerung: primitive Rekursion

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

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

add := $\text{pr}(\text{succ} \circ \pi_2^3, \pi_1^1)$ 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 : $\mathbb{N}^2 \rightarrow \mathbb{N}$

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

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

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

Primitiv-rekursive Funktionen II

Erinnerung: primitive Rekursion

$f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ mit $f = \text{pr}(h, g)$ 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 : $\mathbb{N}^2 \rightarrow \mathbb{N}$

add := $\text{pr}(\text{succ} \circ \pi_2^3, \pi_1^1)$ 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 : $\mathbb{N}^2 \rightarrow \mathbb{N}$

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

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

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

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

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

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 $\underline{g : \mathbb{N}^{k+1} \rightarrow \mathbb{N}}$:

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

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

↓ „kleinste Nullstelle von g s.d. g vorher
voll definiert“

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 $g : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$:

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

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

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

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 $\mu(1_2)$?

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!

Theorem (ohne Beweis)

μ -rekursiv \equiv

Turing/WHILE/GOTO-berechenbar

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!

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

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

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 $g : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$:

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

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

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

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

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

\rightsquigarrow es reicht immer **ein** μ -Operator

Grenzen der LOOP-Berechenbarkeit

Wissen: alle LOOP-berechenbaren Funktionen sind total

Frage: gibt es totale Funktionen die nicht LOOP-berechenbar sind?

Grenzen der LOOP-Berechenbarkeit

Wissen: alle LOOP-berechenbaren Funktionen sind total

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

reelle Zahlen in $[0,1]$

$$\Sigma_{\text{Loop}} = \{0, 1, \dots, 9, ., :, \times, +, -, \text{LOOP}, \text{DO}, \text{END}\}$$

$r_1 0, \textcolor{red}{0}010121\dots$

$$|\Sigma_{\text{Loop}}| = 18$$

$r_2 0, \textcolor{red}{1}515729\dots$

$r_3 0, 92\textcolor{red}{5}1101\dots$

$r_4 0, 210\textcolor{red}{2}105\dots$

\vdots

$r = 0, 1663\dots$

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

$$g(n) := \underline{L_n(n)} + 1$$

total und **nicht** LOOP-berechenbar.

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
$$g(n) := L_n(n) + 1 \quad (1)$$

total und nicht LOOP-berechenbar.

Beweis

Wäre g LOOP-berechenbar, so gäbe es ein $k \in \mathbb{N}$ mit $\underline{L_k = g}$. (2)

$$\sim g(k) \stackrel{(1)}{=} \underline{L_k(k) + 1} \stackrel{(2)}{=} g(k) + 1$$

g

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):

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

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

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

„verallgemeinerte Exponentialfunktion“

$$\varphi(0, y) \rightsquigarrow \underbrace{2 + 2 + \dots + 2}_y = \underline{2^y}$$

$$\varphi(1, y) \rightsquigarrow \underbrace{2 \cdot 2 \cdot \dots \cdot 2}_y = \underline{2^y}$$

$$\varphi(2, y) \rightsquigarrow \underbrace{2^{2^{\dots^2}}}^{2^y} = \underline{2^{2^y}}$$

⋮

⋮

⋮

Ackermannfunktion I

Die Ackermannfunktion (Variante Rósza Péter):

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

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

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

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

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 $\text{ack}(4, 2) \approx 2 \cdot 10^{19728}$).

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:

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

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

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

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

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.

Ackermannfunktion II

Theorem

Die Ackermannfunktion a ist nicht LOOP-berechenbar.

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

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.

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.

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.

$$\begin{matrix} n_0 & n_1 \\ \parallel & \parallel \\ n_k & \xrightarrow{P} x_i ? \end{matrix}$$

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

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

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: $P = "x_i := x_j \pm c"$

$$\leadsto f_P(n) \leq 2n + c$$

Aufang : $x_i + x_j \leq n$

Ende : $x_i = x_j \pm c$

$$+ x_j$$

$$\frac{2x_j + c}{2} \leq 2n + c$$

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: $P = „x_i := x_j \pm c“$

$$\leadsto f_P(n) \leq 2n + c \leq 3n < 3n + 1$$

$$\begin{array}{c} \uparrow \\ \ell \geq c \Rightarrow \boxed{n \geq c} \end{array}$$

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: $P = „x_i := x_j \pm c“$

$$\leadsto f_P(n) \leq 2n + c \leq 3n < 3n + 1 = a(1, n).$$

$\begin{matrix} 1 \\ \text{dfl.} \end{matrix}$

Ackermannfunktion III

Lemma

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

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 $\ell := \max\{c, 1\}$.

$$\leq a(\ell, n)$$

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: $P = "\underline{P_1}; \underline{P_2}"$

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

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: $P = „x_i := x_j \pm c“$

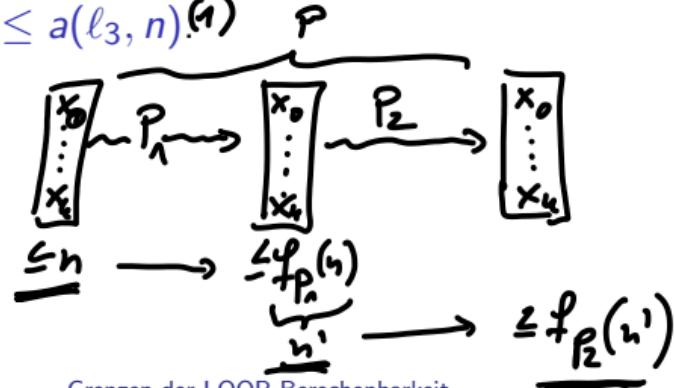
$\rightsquigarrow f_P(n) \leq 2n + c \leq 3n < 3n + 1 = a(1, n)$. \rightsquigarrow Wähle $\ell := \max\{c, 1\}$.

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 $f_{P_2}(n) < a(\ell_2, n) \leq a(\ell_3, n)$.⁽¹⁾

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

$f_P(n) < a(\ell_3, f_{P_1}(n))$
⁽¹⁾



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:

(2) $f_{P_1}(n) < a(\ell_1, n) \leq a(\ell_3, n)$ 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 $\ell_3 \geq 2$):

$f_P(n) < a(\ell_3, f_{P_1}(n)) \leq a(\ell_3, a(\ell_3, n))$

(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 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 $\ell_3 \geq 2$):

$$f_P(n) < a(\ell_3, f_{P_1}(n)) \leq a(\ell_3, a(\ell_3, \underbrace{n}_{\text{n-mal}})) \leq \underbrace{a(\ell_3, a(\ell_3, \dots, a(\ell_3, \overbrace{n}^{\geq n} \dots)))}_{n-\text{mal}}$$

Ackermannfunktion III

Lemma

Zu jedem LOOP-Programm P existiert ein $\underline{\ell \in \mathbb{N}}$ 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 $\underline{\ell_3 \geq 2}$):

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

\uparrow
dp. $a(2, n)$

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

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: $P = \text{"LOOP } \underline{x_i} \text{ DO } \underline{P'} \text{ END"}$

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

Ackermannfunktion III

Lemma

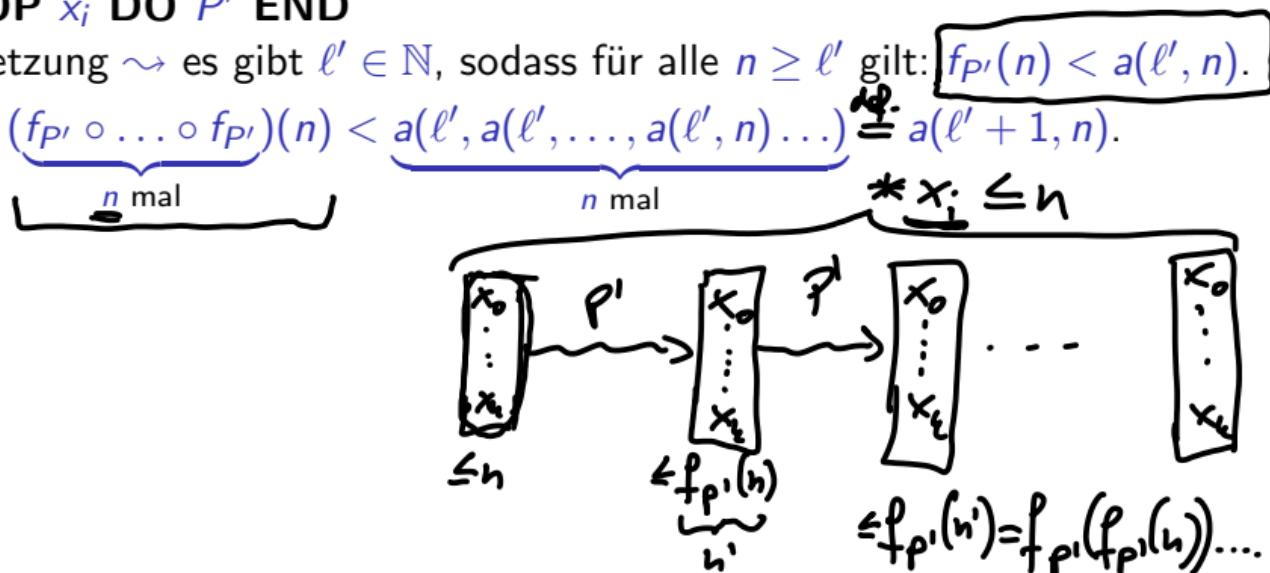
Zu jedem LOOP-Programm P existiert ein $\ell \in \mathbb{N}$ sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$.

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 $f_P(n) \leq (f_{P'} \circ \dots \circ f_{P'})(n) < a(\ell', a(\ell', \dots, a(\ell', n) \dots)) \stackrel{\text{d.h.}}{=} a(\ell' + 1, n)$.



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 $\ell := \max\{\ell' + 1, 2\}$

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.

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

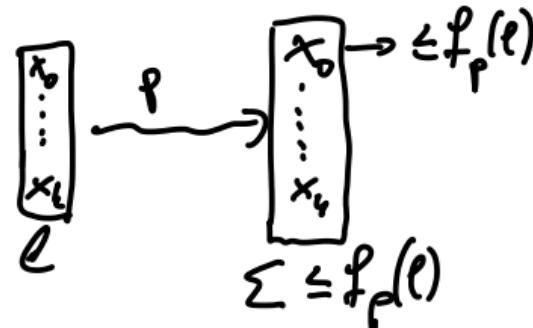
Annahme: a LOOP-berechenbar.

$\sim g(n) := \underline{a(n, n)}$ LOOP-berechenbar vermöge LOOP-Programm P .

\sim es gibt ein $\ell \in \mathbb{N}$, sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$. (1)

$\sim \underline{g(\ell)} \leq f_P(\ell) \stackrel{(1)}{<} a(\ell, \ell) = \underline{g(\ell)}$.

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



Ein WHILE-Programm für die Ackermannfunktion

Theorem

a ist WHILE-berechenbar.

Beweis

Idee: Rekursion in WHILE-Schleife pressen

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

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

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

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!

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. $c(x, y) := \underline{2^{x+y}} + \underline{x}$

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

$$\begin{array}{c} x+y \\ \swarrow \quad \searrow \\ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \\ \swarrow \quad \searrow \\ \underline{\text{BIN}(x)} \end{array}$$

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

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 first($c(x, y)$) := x und 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   else if x = 1 then
8     | PUSH(3 · y + 1)
9   else
10    | LOOP y DO PUSH(x - 1) END;
11    | PUSH(y)
12 x₀ ← POP;
```

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

$$a(4, 2) = a(3, a(3, 2))$$

Stack: 3 3 2 *
a(3, 2)

$$3 \quad z$$

$$\downarrow \quad \downarrow$$

$$a(3, z)$$

$$\downarrow \quad \downarrow$$

Frage 1: wenn wir versuchen mit dem gleichen Beweis zu zeigen dass a nicht WHILE-berechenbar ist, muss dies fehlschlagen wo genau schlägt es fehl?

Frage 2: können Sie mit Hilfe von Diagonalisierung zeigen, dass es nicht berechenbare (zf) Funktionen gibt? (zf)

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. Das Postsche Korrespondenzproblem
10. Komplexität – Einführung
11. NP-Vollständigkeit
12. 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

$$Z = \{z_0, z_1, \dots, z_n = z_e\}$$

$$\Sigma = \{0, 1\}$$

$$\Gamma = \{a_0 = \square, a_1, \dots, a_k\}$$

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

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

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 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 $\{0, 1, \#\}$:

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

↪ 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 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 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 $\underline{\{0, 1\}^*}$ kodieren eine Turing-Maschine.

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.

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

$\underline{w} \in \{0, 1\}^*$ keine valide Kodierung

↪ feste Maschine M_Ω , die die nigends definierte Funktion berechnet → M_Ω hält auf keiner Eingabe!

Spezielles Halteproblem

Definition

Das spezielle Halteproblem ist die Sprache

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

$$w = 110110$$

$M_{\underline{\underline{110110}}}$ bei Eingabe $\underline{\underline{110110}}$? hält $\rightarrow \in K$
nicht hält $\rightarrow \notin K$

Falls kein Codewort: $M_w = M_\varnothing \rightarrow \notin K$

Spezielles Halteproblem

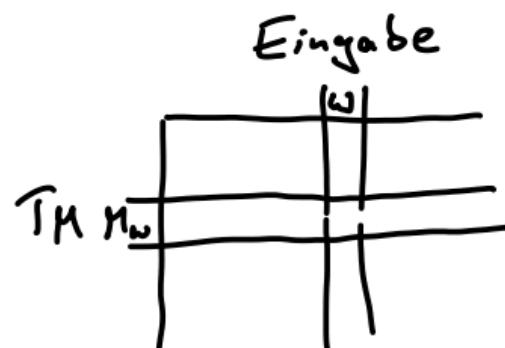
Definition

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.



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 \underline{M} .

Spezielles Halteproblem

Definition

Das **spezielle Halteproblem** ist die Sprache

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

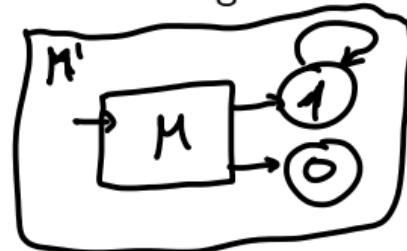
Theorem

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.



\rightsquigarrow codewort w'

$$M' = M \cup$$

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. $M' = M_{w'}$.

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'

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

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

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

Spezielles Halteproblem

Definition

Das **spezielle Halteproblem** ist die Sprache

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

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

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

$\Leftrightarrow w' \notin K$

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 \leadsto 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. $M' = M_{w'}$.

\leadsto 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 $\langle M' \rangle = w'$. ↴

ABER: halbe charakteristische Funktion χ'_K kann durchaus berechenbar sein

Frage: wo zerbricht der Beweis wenn wir χ'_K verwenden?

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. Das Postsche Korrespondenzproblem
10. Komplexität – Einführung
11. NP-Vollständigkeit
12. PSPACE

(Semi-)Entscheidbarkeit und Aufzählbarkeit

Definition (Erinnerung)

- a) Eine Sprache $A \subseteq \Sigma^*$ heißt **entscheidbar**, falls die charakteristische Funktion $\chi_A : \Sigma^* \rightarrow \{0, 1\}$ berechenbar ist.

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

(Semi-)Entscheidbarkeit und Aufzählbarkeit

Definition (Erinnerung)

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

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

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

(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 $A = \{f(0), f(1), f(2), \dots\} = \underline{f(\mathbb{N})}$.

$$\bigcup_{i \in \mathbb{N}} f(i)$$

(Semi-)Entscheidbarkeit und Aufzählbarkeit

Definition (Erinnerung)

- a) Eine Sprache $A \subseteq \Sigma^*$ heißt **entscheidbar**, falls die charakteristische Funktion $\chi_A : \Sigma^* \rightarrow \{0, 1\}$ berechenbar ist.
- b) 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 $A = \{f(0), f(1), f(2), \dots\} = \underline{\underline{f(\mathbb{N})}}$. Das heißt, f zählt A auf.

Beachte: f muss weder injektiv noch monoton sein!

(Semi-)Entscheidbarkeit und Aufzählbarkeit

Definition (Erinnerung)

- a) Eine Sprache $A \subseteq \Sigma^*$ heißt **entscheidbar**, falls die charakteristische Funktion $\chi_A : \Sigma^* \rightarrow \{0, 1\}$ berechenbar ist.
- b) 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.

Entscheidbare und Semi-Entscheidbare Sprachen

Theorem

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

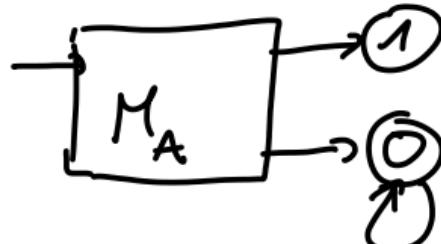
Beweis

" \Rightarrow ": A entscheidbar \Rightarrow

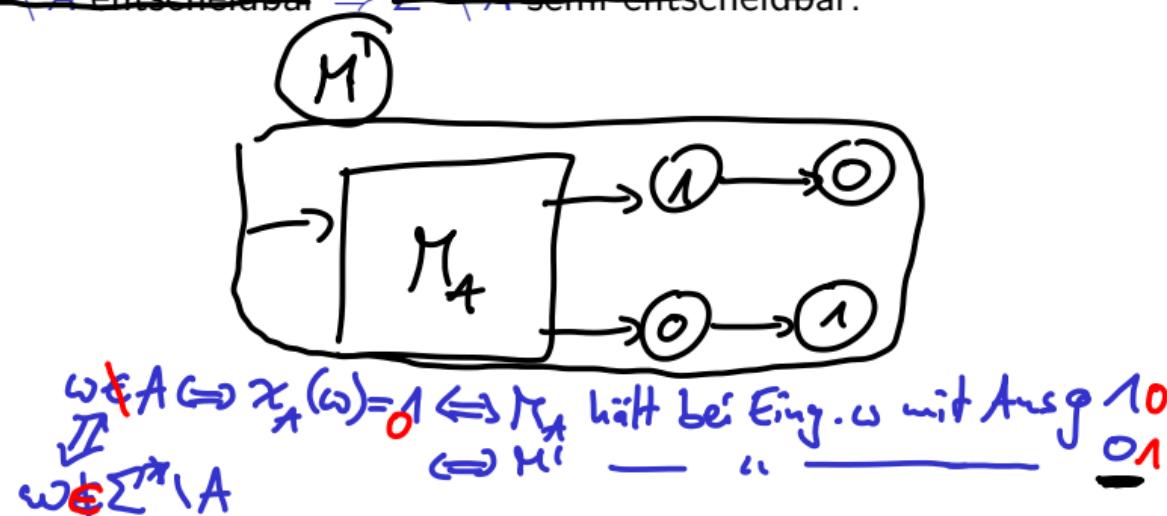
1. A semi entscheidbar.

2. $\Sigma^* \setminus A$ entscheidbar \Rightarrow $\Sigma^* \setminus A$ semi entscheidbar.

x_A berechenbar



$\Rightarrow x'_A$ berechenbar



Entscheidbare und Semi-Entscheidbare Sprachen

Theorem

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

Beweis

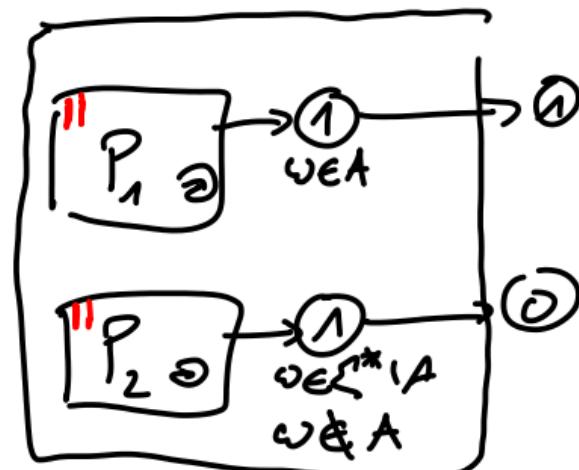
„ \Rightarrow “: A entscheidbar \Rightarrow 1. \underline{A} semi-entscheidbar.
2. $\Sigma^* \setminus A$ entscheidbar $\Rightarrow \Sigma^* \setminus A$ semi-entscheidbar.

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

$[x_i := 1; \text{WHILE } x_i \neq 0 \text{ DO } \underline{P_A} \text{ END; } x_0 := 1] P_1$

sowie

$[x_j := 1; \text{WHILE } x_j \neq 0 \text{ DO } \underline{P_{\bar{A}}} \text{ END; } x_0 := 1] P_2$



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 L ist aufzählbar
gdw.

L is semi-entscheidbar.

(Rekursiv) Aufzählbare Sprachen

Theorem

Eine Sprache L ist aufzählbar
gdw.

L is semi-entscheidbar.

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

(Rekursiv) Aufzählbare Sprachen

Theorem

Eine Sprache L ist aufzählbar
gdw.

L is semi-entscheidbar.

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

Beweis

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

$\sim \underline{\chi'_A}$ berechnet durch

$$f(0) = \underline{\chi_A}^2$$

1 $x_2 := 0;$

$$f(1) = \underline{x_1}^2$$

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

$$f(2) = \underline{x_2}^2$$

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

$$\vdots$$

4 $x_2 := x_2 + 1;$

5 **END**

(Rekursiv) Aufzählbare Sprachen

Theorem

Eine Sprache L ist aufzählbar
gdw.

L 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 1: Berechne erst $\chi'_A(0)$, dann $\chi'_A(1), \dots$

Problem: $\chi'_A(0)$ kann endlos laufen!

$$\mathbb{N} = \{0, 1, 2, 3, 4, 5, \dots\}$$

EA?

$$A = \{\}$$

(Rekursiv) Aufzählbare Sprachen

Theorem

Eine Sprache L ist aufzählbar
gdw.

L 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 $x_2 := x_2 + 1;$

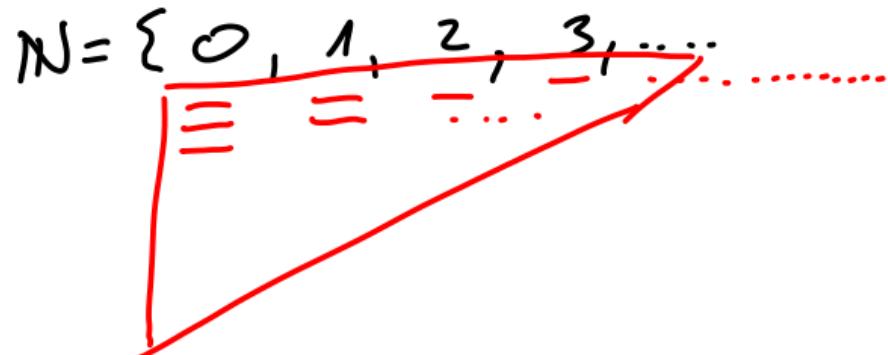
5 **END**

„ \Leftarrow “ (Skizze):

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

Versuch 2: Simuliere $\chi'_A(i)$ für jedes $i \in \mathbb{N}$ einen Schritt, dann noch einen Schritt, ...

Problem: es gibt unendlich viele Eingaben $i \in \mathbb{N}$!



(Rekursiv) Aufzählbare Sprachen

Theorem

Eine Sprache L ist aufzählbar gdw.

L 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

$x_2 := 0;$

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

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

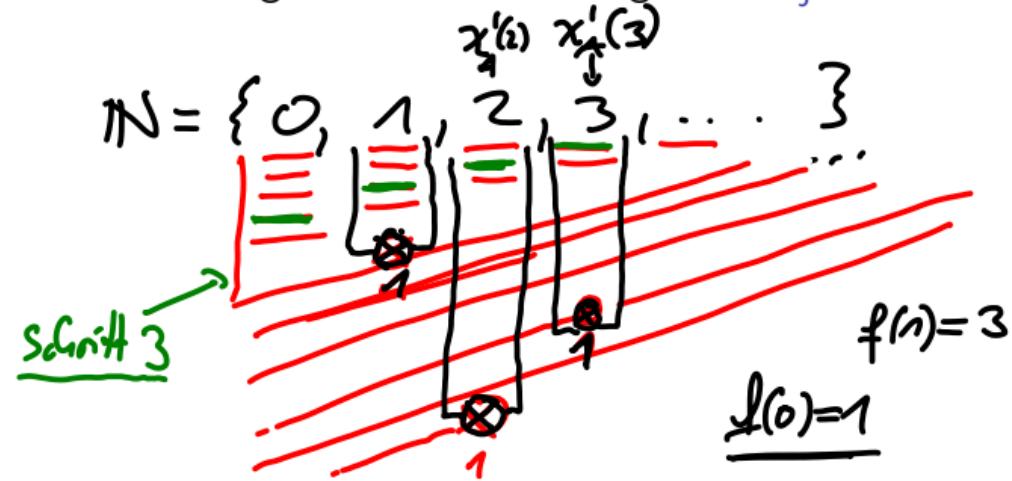
END

$$f(n) =$$

„ \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 nicht „Erfolge“ ($\chi'_A(j) = 1$) beobachtet wurden und gebe das letzte erfolgreiche w_j aus.



(Rekursiv) Aufzählbare Sprachen

Theorem

Eine Sprache L ist aufzählbar gdw.

L 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 ": Sei χ'_A berechnet durch WHILE-Programm mit $k+1$ Variablen y_0, \dots, y_k (P nutzt y_0 nicht):
 $y_k := 1; \text{ WHILE } y_k \neq 0 \text{ DO } P \text{ END; } y_0 := 1$

Das folgende Programm zählt A auf:

```

1   $x_1 := x_1 + 1;$ 
2  WHILE  $x_1 \neq 0$  DO
3    PUSH  $[0, x_2, 0, \dots, 0, x_2]$  TO  $n_1;$   $\chi'_A(x_2)$ 
4    WHILE  $n_1 \neq 0$  und  $x_1 \neq 0$  DO
5      (POP  $x_0, y_k, \dots, y_0$  FROM  $n_1;$ )
6      |  $P;$ 
7      | IF  $y_k = 0$  THEN  $x_1 := x_1 - 1;$   $\leftarrow \# \text{Erfolge}$ 
8      | ELSE PUSH  $y_0, \dots, y_k, x_0$  TO  $n_2;$   $\text{noch zu schen}$ 
9    END
10    $n_1 := n_2;$ 
11    $x_2 := x_2 + 1;$ 
12 END

```



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

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

Zusammenfassung (Semi-) Entscheidbarkeit

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

A ist semi-entscheidbar

$\Leftrightarrow \chi'_A$ ist berechenbar

$\Leftrightarrow \underline{A}$ 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 $g : \underline{\Pi^*} \rightarrow \Sigma^*$
(A lässt sich schreiben als $A = g(\Pi^*)$)

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)

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



Zusammenfassung (Semi-) Entscheidbarkeit

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

A ist semi-entscheidbar

A ist entscheidbar

$\Leftrightarrow \chi'_A$ ist berechenbar

$\Leftrightarrow \chi_A$ ist berechenbar

$\Leftrightarrow A$ ist aufzählbar

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

$\Leftrightarrow A = T(M)$ 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)

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 $A = g(\Pi^*)$)

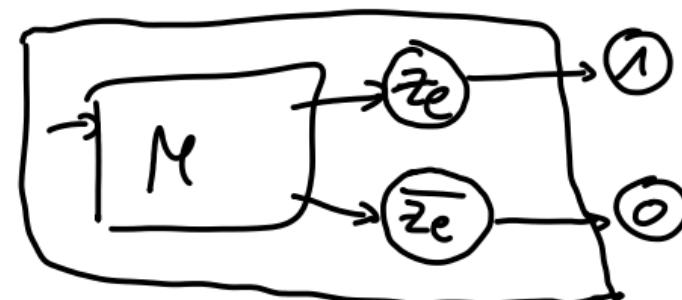
$\Leftrightarrow A$ ist Typ 0-Sprache (Chomsky-Hierarchie)

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**



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. PSPACE

Reduzierbarkeit I

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

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

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

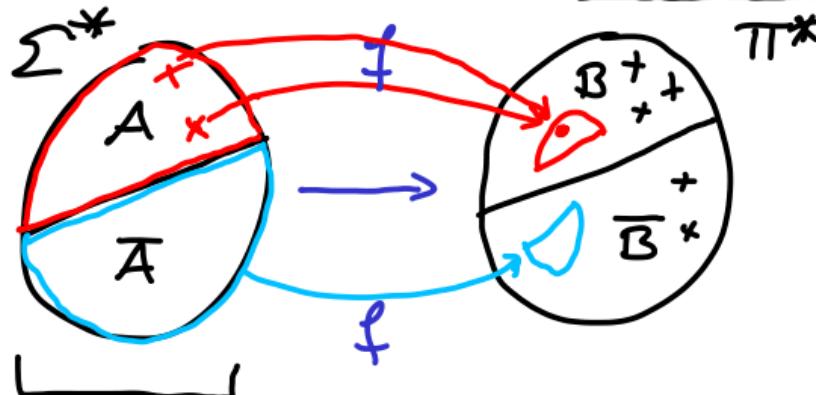
Definition

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

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



$$x \in A \Rightarrow f(x) \in B$$
$$x \notin A \Rightarrow f(x) \notin B$$

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"

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

$$\Sigma \quad \Pi$$

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

$$w \in K \Leftrightarrow M_w \text{ hält bei Eingabe } w \Leftrightarrow w\#w \in H \\ f(w)$$

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 $\overline{A} = \text{Co-}A = \Sigma^* \setminus A$)

Reduzierbarkeit III

$$\neg p \Leftrightarrow \neg q$$

Lemma

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

$$\begin{aligned} \underline{A \leq B} &\Leftrightarrow \exists \text{ Reduktion } f: \forall x \in \Sigma^*: \underline{x \in A} \Leftrightarrow \underline{f(x) \in B} \\ &\Leftrightarrow \exists \text{ Reduktion } f: \forall x \in \Sigma^*: \underline{x \notin A} \Leftrightarrow \underline{f(x) \notin B} \\ &\Leftrightarrow \exists \text{ Reduktion } f: \forall x \in \Sigma^*: \underline{x \in \overline{A}} \Leftrightarrow \underline{f(x) \in \overline{B}} \Leftrightarrow \underline{\overline{A} \leq \overline{B}} \end{aligned}$$

Reduzierbarkeit III

Lemma

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

„A leichter als B“

Lemma

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

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

$$\underline{x \in A} \Rightarrow (\underline{\chi_B \circ f})(x) = \underline{\chi_B}(f(x)) = 1$$

$$\underline{x \notin A} \Rightarrow (\underline{\chi_B \circ f})(x) = \underline{\chi_B}(f(x)) = 0$$

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 $x \in A \Leftrightarrow \underline{f(x)} \in B$).

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

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

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

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

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

$$U \equiv H$$

Lemma

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

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

Wortproblem für Turing-Maschinen

$w \# x \in H \Leftrightarrow M_w \text{ hält auf } x$
 $\Leftrightarrow M' \text{ akzeptiert } x$
 $\Leftrightarrow \underline{\langle M' \rangle \# x} \in U$

\emptyset
 \emptyset

Reduktionseigenschaft

Lemma

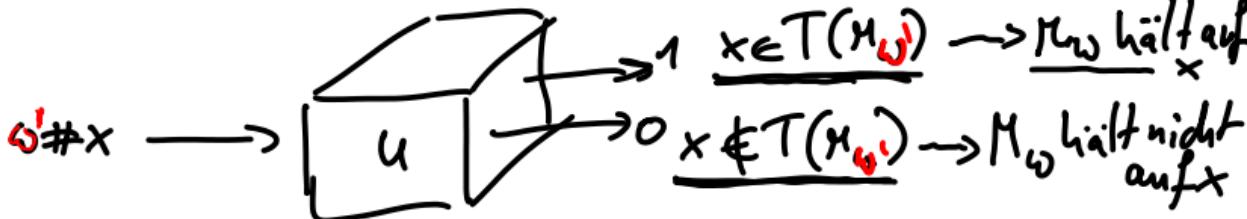
Für die Sprache $U := \{w \# x \mid x \in T(M_w)\}$ gilt: $U \leq H$ und $H \leq U$.

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).

$w \# x$



Wortproblem für Turing-Maschinen

Lemma

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

$$\begin{aligned} w\#x \in U &\Leftrightarrow x \in T(M_w) \\ &\Leftrightarrow M_{w'} \text{ hält auf } x \\ &\Leftrightarrow w'\#x \in H \\ &\Downarrow \\ f(w\#x) \end{aligned}$$

Reduktionseigenschaft

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.



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 $H_0 := \{ \underline{w} \mid \underline{w\#} \in H \}$.

Halteproblem auf leerem Band

Definition

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

Theorem

H_0 ist unentscheidbar.

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 $\underline{H \leq H_0}$ ("H leichter als H_0 ") durch Konstruktion einer Reduktion f von H auf H_0 .

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

Halteproblem auf leerem Band

$$\begin{aligned} w\#x \in H &\Leftrightarrow M_w \text{ hält auf } x \\ &\Leftrightarrow M_w \text{ hält auf } \epsilon \Leftrightarrow w' \in H_0 \end{aligned}$$

Definition

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

Theorem

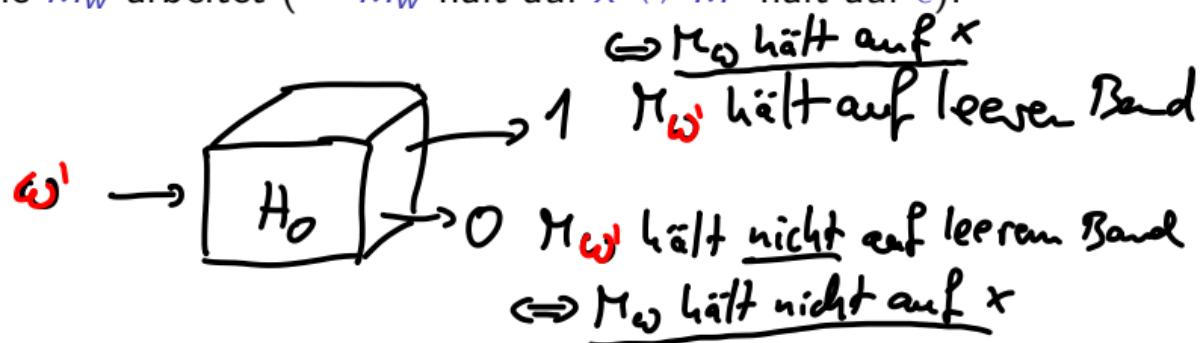
H_0 ist unentscheidbar.

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 ($\sim M_w \text{ hält auf } x \Leftrightarrow M' \text{ hält auf } \epsilon$).



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 ($\rightsquigarrow M_w \text{ hält auf } x \Leftrightarrow M' \text{ hält auf } \epsilon$).

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

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

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

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

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: $\underline{q \notin H}$ und $\underline{f(q) \notin H_0}$.

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. PSPACE

„Rice'sche Katastrophe“

Satz von Rice I

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

$\{\omega \mid M_\omega \text{ berechnet } \sigma_2\} \rightarrow \text{unentscheidbar}$

$\{\omega \mid M_\omega \text{ berechnet konst. Flf}\} \rightarrow \text{--}$

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

$$S = \{\underline{\Omega}\} \rightarrow C(S) \text{ unentscheidbar}$$

$$S = \{\text{konst.-Funk. } 0, \text{ konst.-Funk. } 1, \dots\}$$

$$C(\emptyset) = \{\underline{w} \mid \cancel{\text{die }} M_w \text{ berechnet keine Fkt.}\} = \emptyset$$

Frage: warum ist $C(\mathcal{R})$ entscheidbar?

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} .

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 $H_0 \leq \mathcal{C}(\mathcal{S})$.

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 g in \mathcal{S} liegt.

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.

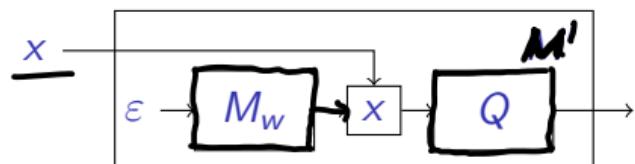
Beweis

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
2. nachdem M_w hält, Q auf Eingabe x simuliert.



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

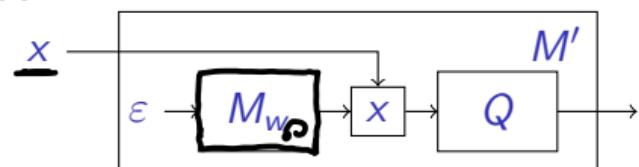
Fall 1: Die überall undefinierte Funktion Ω ist nicht in \mathcal{S} . Wir zeigen $H_0 \leq \mathcal{C}(\mathcal{S})$.

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
- * 2. nachdem M_w hält, Q auf Eingabe x simuliert.

$\sim M'$ berechnet $\begin{cases} q & \text{falls } w \in H_0 \\ \Omega & \text{sonst} \end{cases}$



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

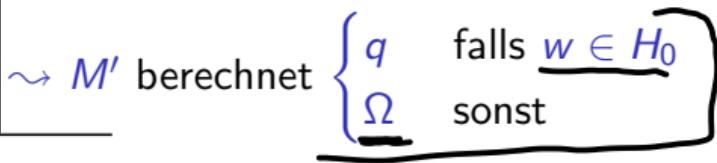
1. erst M_w auf leerem Band simuliert, und
2. nachdem M_w hält, Q auf Eingabe x simuliert.

$$\underline{w \in H_0} \Leftrightarrow M_w \text{ hält auf leerem Band}$$

$$\Leftrightarrow \underline{M' \text{ berechnet } q \in \mathcal{S}}$$

$$\Leftrightarrow \underline{\langle M' \rangle \in \mathcal{C}(\mathcal{S})}$$

Red. Eig.



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 $\underline{H_0}$ 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.

$$\{ 2, 2, 2, \textcircled{D} \}$$

$$x_0 := 1$$

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.

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")

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 $S \subseteq \mathcal{R}$:
zB. $\mathcal{C}(\{q \mid q \neq \Omega\})$ semi-entscheidbar, aber $\mathcal{C}(\{\Omega\})$ nicht



!!! ACHTUNG !!!

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" $\underline{Eq} := \{w \# w' \mid \underline{T}(M_w) = \underline{T}(M_{w'})\}$
 $\underline{H} \leq \underline{Eq}$ aber **nicht** $\underline{Eq} \leq \underline{H}$

Fleißige Bieber

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

~ "fleißige Bieber"

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

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

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

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 $M(n)$ die Menge aller Turing-Maschinen $M = (Z, \Sigma, \Gamma, \delta, \square, z_0, E)$ mit

$$\underline{|Z| = n}$$

$$\underline{\Sigma = \{0, 1\}}$$

$$\underline{\Gamma = \Sigma \cup \{\square\}}.$$

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

$s(w) = \underline{\text{Anzahl Schritte}}$ die M (bei leerer Eingabe) macht (0 falls M nicht hält) und

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

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

$$|Z| = \underline{n}$$

$$\Sigma = \underline{\{0, 1\}}$$

$$\Gamma = \underline{\Sigma \cup \{\square\}}.$$

Wir definieren die Funktionen

$$\underline{S(n)} := \max_{M \in \underline{\mathcal{M}(n)}} s(\langle M \rangle)$$

$$\underline{E(n)} := \max_{M \in \underline{\mathcal{M}(n)}} e(\langle M \rangle)$$

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

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

$$|Z| = n$$

$$\underline{\Sigma} = \{1\}$$

$$\underline{\Gamma} = \underline{\Sigma} \cup \{\square\}.$$

Wir definieren die Funktionen

$$\underline{S}_1(n) := \max_{M \in \mathcal{M}_1(n)} s(\langle M \rangle)$$

$$\underline{E}_1(n) := \max_{M \in \mathcal{M}_1(n)} e(\langle M \rangle)$$

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.

Fleißige Bieber sind Unberechenbar I

Theorem

Die Funktion S ist nicht Turing-berechenbar.

Beweis

Annahme: S ist Turing-berechenbar

~ S wird von einer Turing-Maschine berechnet.

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

Fleißige Bieber sind Unberechenbar I

110 \rightsquigarrow 11111

binär

unär

Theorem

Die Funktion S ist nicht Turing-berechenbar.

Beweis

Annahme: S ist Turing-berechenbar

$\rightsquigarrow S$ wird von einer Turing-Maschine berechnet.

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

Sei $\underline{M'}$ die Maschine, die wie folgt agiert:

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

Sei $\underline{n'}$ die Anzahl Zustände von $\underline{M'}$.

Fleißige Bieber sind Unberechenbar I

Theorem

Die Funktion S ist nicht Turing-berechenbar.

Beweis

Annahme: S ist Turing-berechenbar

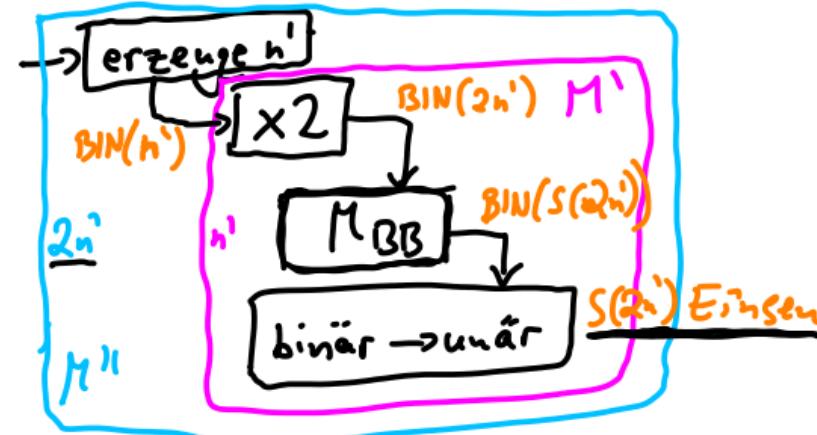
~ S wird von einer Turing-Maschine berechnet.

~ es gibt $n \in \mathbb{N}$ sodass \underline{S} von einer Turing-Maschine $\underline{M}_{BB} \in \underline{M}(n)$ 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 \underline{n}' die Anzahl Zustände von M' .



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

- "zurück"
1. erzeuge die Zahl \underline{n}' auf dem Band
2. simuliere \underline{M}' $\rightarrow \underline{n}'$ Zustände
sodass M'' genau $\underline{2}^{\underline{n}'}$ Zustände hat.

Fleißige Bieber sind Unberechenbar I

Theorem

Die Funktion S ist nicht Turing-berechenbar.

Beweis

Annahme: S ist Turing-berechenbar

~ S wird von einer Turing-Maschine berechnet.

~ 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' .

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

Fleißige Bieber sind Unberechenbar I

Theorem

Die Funktion S ist nicht Turing-berechenbar.

Beweis

Annahme: S ist Turing-berechenbar

~ S wird von einer Turing-Maschine berechnet.

~ 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' .

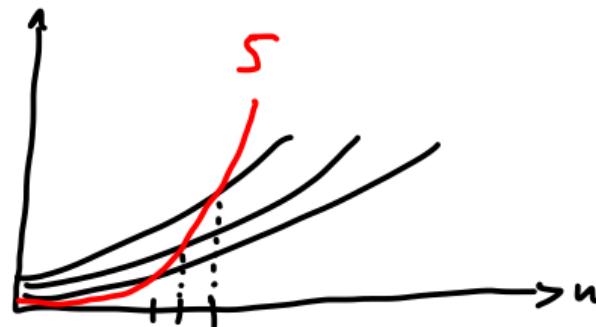
~ M'' hat $2n'$ Zustände aber M'' macht bei leerer Eingabe **echt mehr** als $S(2n')$ Schritte ↴

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

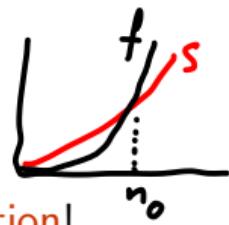
Fleißige Bieber sind Unberechenbar II

Theorem

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



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 $n > n_0$.

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) $\rightsquigarrow n > n_0$
- | 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 hielt

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 hielt → Ausgabe 0/1

→ 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

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]

universelle TM

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} ↳

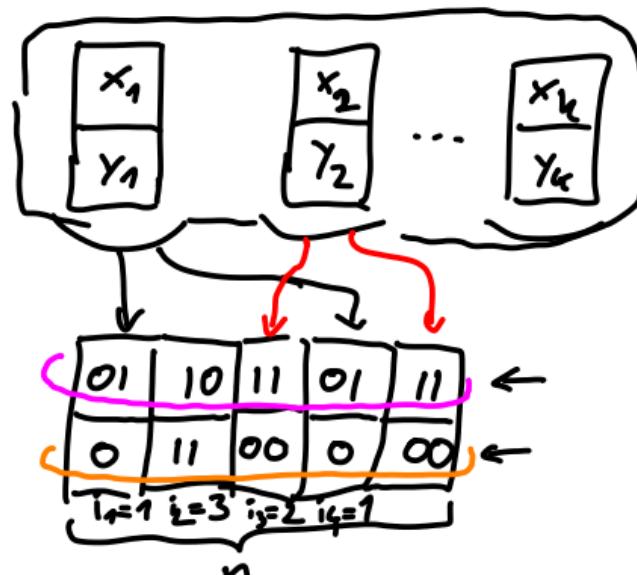
Das Postsche Korrespondenzproblem I

Definition

Für ein endliches Alphabet Σ ist das Postsche Korrespondenzproblem die Menge

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

$01 \cdot 10 \cdot 11 \cdot 01 \cdot 11 \neq 0 \cdot 11 \cdot 00 \cdot 0 \cdot 00$



Das Postsche Korrespondenzproblem I

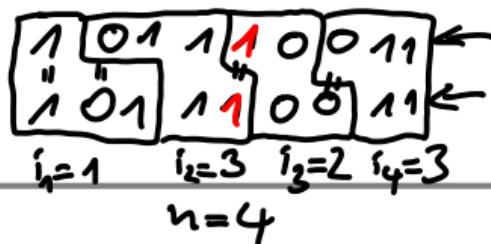
Definition

Für ein endliches Alphabet Σ ist das Postsche Korrespondenzproblem die Menge

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

Beispiel 1

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



Das Postsche Korrespondenzproblem I

Definition

Für ein endliches Alphabet Σ ist das Postsche Korrespondenzproblem die Menge

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

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

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

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

Das Postsche Korrespondenzproblem I

Definition

Für ein endliches Alphabet Σ ist das Postsche Korrespondenzproblem die Menge

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

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

$$x_1 \cdot x_3 \cdot x_2 \cdot x_3 = \boxed{1} \cdot 011 \cdot 10 \cdot \boxed{011} = 101110011$$

$$y_1 \cdot y_3 \cdot y_2 \cdot y_3 = \boxed{101} \cdot 11 \cdot 00 \cdot \boxed{11} = 101110011$$

Beispiel 2

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

Das Postsche Korrespondenzproblem I

Definition

Für ein endliches Alphabet Σ ist das Postsche Korrespondenzproblem die Menge

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

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

$\text{PCP} \leq \mathcal{Z} \Rightarrow \mathcal{Z}$ unentscheidbar

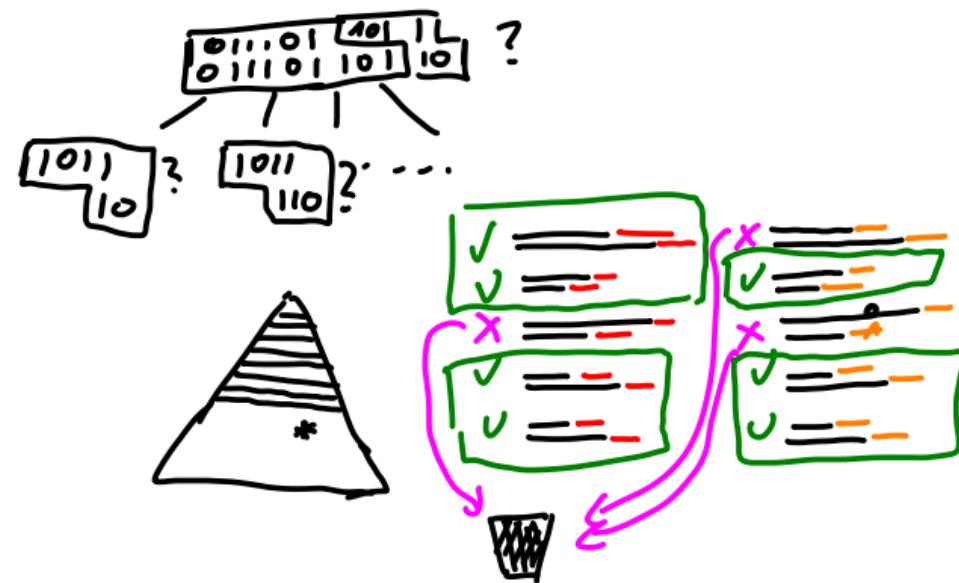
Hinweise:

1. PCP oft in Unentscheidbarkeitsbeweisen (Reduktionen) benutzt

Das Postsche Korrespondenzproblem II

Hinweise:

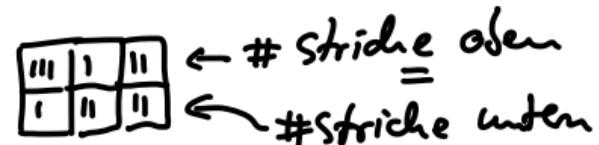
1. PCP oft in Unentscheidbarkeitsbeweisen (Reduktionen) benutzt
2. PCP semi-entscheidbar für beliebiges Σ (vermöge Brute-Force-Algorithmus).



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:



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

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

$$\begin{matrix} \{2, -5\} \\ \uparrow \quad \uparrow \\ 5x \quad 2x \end{matrix} \rightsquigarrow \sum = 0$$

↪ 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

$\exists n \geq 1$

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.

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



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

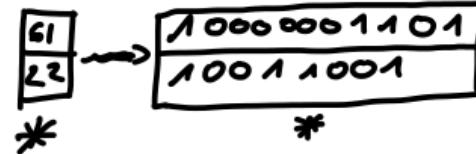
$$\sum_{j \leq n} |x_{ij}| = \sum_{j \leq n} |y_{ij}| \quad \text{also} \quad \sum_{j \leq n} (|x_{ij}| - |y_{ij}|) = 0.$$

↪ 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



Frage: warum funktioniert die Reduktion für 6. nicht so:



Das Modifizierte PCP

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

$$\text{MPCP} := \left\{ \left(\underbrace{\binom{x_1}{y_1}}, \binom{x_2}{y_2}, \dots, \binom{x_k}{y_k} \right) \in \text{PCP} \mid \exists_{n \geq 1} \exists_{\underbrace{i_2, \dots, i_n}_{\in \{1, 2, \dots, k\}}} : \underbrace{x_1}_{-} \cdot x_{i_2} \cdot \dots \cdot x_{i_n} = \underbrace{y_1}_{-} \cdot y_{i_2} \cdot \dots \cdot y_{i_n} \right\}$$

Das Modifizierte PCP

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

$$\text{MPCP} := \left\{ \left(\binom{x_1}{y_1}, \binom{x_2}{y_2}, \dots, \binom{x_k}{y_k} \right) \in \text{PCP} \mid \exists_{n \geq 1} \exists_{i_2, \dots, i_n \in \{1, 2, \dots, k\}} : x_1 \cdot x_{i_2} \cdot \dots \cdot x_{i_n} = y_1 \cdot y_{i_2} \cdot \dots \cdot y_{i_n} \right\}$$

Lemma

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

Das Modifizierte PCP

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

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

Lemma

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

Verwende neue Symbole $\underline{\$}$, $\underline{\#}$ $\notin \Sigma$

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

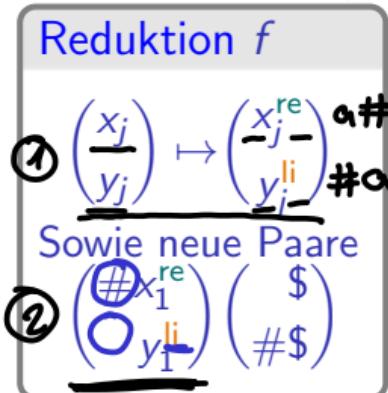
$$\underline{(aw)}^{\text{li}} = \underline{\#} \underline{aw}^{\text{li}}$$

$$\underline{(aw)}^{\text{re}} = \underline{a} \underline{\#} \underline{w}^{\text{re}}$$

$$\underline{\varepsilon^{\text{li}}} = \underline{\varepsilon^{\text{re}}} = \underline{\varepsilon}$$

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

$$\begin{aligned} \underline{ab} \underline{a}^{\text{li}} &\rightsquigarrow ab \underline{a}^{\text{li}} = \underline{\#} \underline{a} (\underline{ba})^{\text{li}} = \underline{\#} \underline{a} \underline{\#} \underline{b} \underline{\#} \underline{a} \\ (\underline{ba})^{\text{li}} &= \underline{\#} \underline{b} \underline{a} (\underline{a})^{\text{li}} = \underline{\#} \underline{b} \underline{\#} \underline{a} \\ \underline{a}^{\text{li}} &= \underline{\#} \underline{a} (\underline{\varepsilon})^{\text{li}} = \underline{\#} \underline{a} \end{aligned}$$



Das Modifizierte PCP

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

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

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 $P \in \text{MPCP} \Leftrightarrow f(P) \in \text{PCP}$.

Reduktion f

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

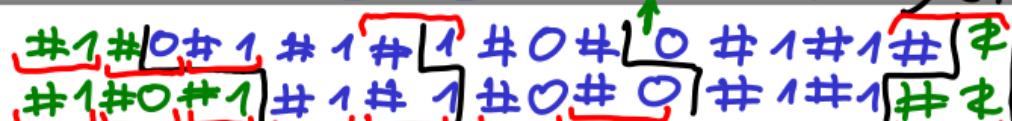
Sowie neue Paare

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

Beispiel (Lösung $(\underline{1, 3, 2, 3}) \mapsto (\underline{4, 3, 2, 3, 5})$)

$$\left(\begin{pmatrix} 1 \\ 101 \end{pmatrix}, \begin{pmatrix} 10 \\ 00 \end{pmatrix}, \begin{pmatrix} 011 \\ 11 \end{pmatrix} \right) \mapsto \left(\begin{pmatrix} 1\# \\ \#1\#0\#1 \end{pmatrix}, \begin{pmatrix} 1\#0\# \\ \#0\#0 \end{pmatrix}, \begin{pmatrix} 0\#1\#1\# \\ \#1\#1 \end{pmatrix}, \begin{pmatrix} \#1\# \\ \#1\#0\#1 \end{pmatrix}, \begin{pmatrix} \$ \\ \#\$ \end{pmatrix} \right) \in \text{PCP}?$$

MPCP?



Das Modifizierte PCP

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

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

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 $\underline{P} \in \text{MPCP} \Leftrightarrow f(P) \in \text{PCP}$.

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

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

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

Das Modifizierte PCP

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

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

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

Reduktion f

$$\binom{x_j}{y_j} \mapsto \binom{x_j^{\text{re}}}{y_j^{\text{li}}}$$

Sowie neue Paare

$$\binom{\#x_1^{\text{re}}}{y_1^{\text{li}}} \binom{\$}{\#\$}$$

Das Modifizierte PCP

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

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

Lemma

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

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 $\overbrace{P \in \text{MPCP} \Leftrightarrow f(P) \in \text{PCP.}}$

“ \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 $\rightsquigarrow (1, \underbrace{i_2, \dots, i_n})$ 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

H_0 reduzierbar auf MPCP

Beweis (Skizze)

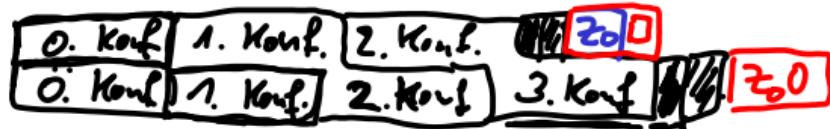
Reduktion f erhält das Codewortes von $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, \{z_e\})$ und erzeugt MPCP-Instanz.
oBdA: Eingabemaschine M hält $\Leftrightarrow M$ hält in akzeptierendem Zustand

Imitation der Konfigurationsfolge - Beispiel

Initiale Konfiguration $\square z_0 \square$, Maschine M mit $\underline{\delta(z_0, \square) = (z_0, 0, N)}$ und $\underline{\delta(z_0, 0) = (z_0, 1, R)}$

$(\# \underline{\square z_0 \square} \#), (\#), (\square), (0), (1), (\# \square), (\square \#), (\underline{z_0 \square}), (\underline{z_0 0})$

Start Kopieren Band-Extension Übergänge



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

Imitation der Konfigurationsfolge - Beispiel

Initiale Konfiguration $\square z_0 \square$, Maschine M_w mit $\delta(z_0, \square) = (z_0, 0, N)$ und $\delta(z_0, 0) = (z_0, 1, R)$

$(\#_{\# \square z_0 \square \#}), (\#), (\square), (0^0), (1^1), (\#_{\# \square}), (\#_{\square \#}), (\frac{z_0 \square}{z_0 0}), (\frac{z_0 0}{1 z_0})$

\square

\square $z_0 \square \#$,

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

Imitation der Konfigurationsfolge - Beispiel

Initiale Konfiguration $\square z_0 \square$, Maschine M_w mit $\delta(z_0, \square) = (z_0, 0, N)$ und $\delta(z_0, 0) = (z_0, 1, R)$

$(\#_{\# \square z_0 \square \#}), (\#_{\#}), (\underline{\square}_{\square}), ({}^0_0), ({}^1_1), (\#_{\# \square}), (\underline{\square \#}_{\square \#}), (\underline{z_0 \square}_{z_0 0}), (\underline{z_0 0}_{1 z_0})$

\square $\underline{z_0}$ \square

$\underline{z_0}$ \square # \square $\underline{z_0}$ 0

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

Imitation der Konfigurationsfolge - Beispiel

Initiale Konfiguration $\square z_0 \square$, Maschine M_w mit $\delta(z_0, \square) = (z_0, 0, N)$ und $\delta(z_0, 0) = (z_0, 1, R)$

$(\#_{\# \square z_0 \square \#}), (\#_{\#}), (\square_{\square}), (0^0), (1^1), (\#_{\# \square}), (\#_{\square \#}), (\textcolor{red}{z_0 \square}_{z_0 0}), (\textcolor{red}{z_0 0}_{1 z_0})$

$\# \square z_0 \square$

$\# \square z_0 \square \# \square z_0 0$

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

Imitation der Konfigurationsfolge - Beispiel

Initiale Konfiguration $\square z_0 \square$, Maschine M_w mit $\delta(z_0, \square) = (z_0, 0, N)$ und $\delta(z_0, 0) = (z_0, 1, R)$

$(\#_{\# \square z_0 \square \#}), (\#_{\underline{\#}}), (\square_0), (0_0), (1_1), (\#_{\# \square}), (\#_{\square \#}), (\textcolor{red}{z_0 \square}_{z_0 0}), (\textcolor{red}{z_0 0}_{1 z_0})$

$\#\square z_0 \square \# \square$

$\#\square z_0 \square \# \square z_0 0 \#$

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

Imitation der Konfigurationsfolge - Beispiel

Initiale Konfiguration $\square z_0 \square$, Maschine M_w mit $\delta(z_0, \square) = (z_0, 0, N)$ und $\delta(z_0, 0) = (z_0, 1, R)$

$(\#_{\# \square z_0 \square \#}), (\#_{\#}), (\square_{\square}), (0^0_0), (1^1_1), (\#_{\# \square}), (\#_{\square \#}), (\underline{z_0 \square}_{z_0 0}), (\underline{z_0 0}_{1 z_0})$

$\# \square z_0 \square \# \square$

$\# \square z_0 \square \# \square z_0 0 \# \square$

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

Imitation der Konfigurationsfolge - Beispiel

Initiale Konfiguration $\square z_0 \square$, Maschine M_w mit $\delta(z_0, \square) = (z_0, 0, N)$ und $\delta(z_0, 0) = (z_0, 1, R)$

$(\#_{\#\square z_0 \square \#}), (\#_{\#}), (\square_{\square}), (0^0_{0}), (1^1_{1}), (\#_{\#\square}), (\#_{\square\#}), (\frac{z_0 \square}{z_0 0}), (\frac{z_0 0}{1 z_0})$

$\#\square z_0 \square \# \square z_0 0$

$\#\square z_0 \square \# \square z_0 0 \# \square 1 z_0$

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

Imitation der Konfigurationsfolge - Beispiel

Initiale Konfiguration $\square z_0 \square$, Maschine M_w mit $\delta(z_0, \square) = (z_0, 0, N)$ und $\delta(z_0, 0) = (z_0, 1, R)$

$(\#_{\# \square z_0 \square \#}), (\#_{\#}), (\square_{\square}), (0^0_{0}), (1^1_{1}), (\#_{\# \square}), (\underline{\square \#}_{\square \#}), (\textcolor{red}{z_0 \square}_{z_0 0}), (\textcolor{red}{z_0 0}_{1 z_0})$

$\square z_0 \square \# \square z_0 0 \# \square$

$\square z_0 \square \# \square z_0 0 \# \square 1 z_0 \square \#$

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

Imitation der Konfigurationsfolge - Beispiel

Initiale Konfiguration $\square z_0 \square$, Maschine M_w mit $\delta(z_0, \square) = (z_0, 0, N)$ und $\delta(z_0, 0) = (z_0, 1, R)$

$(\#_{\#\square z_0 \square \#}), (\#), (\square), (0^0), (1^1), (\#_{\#\square}), (\#_{\square \#}), (\frac{z_0 \square}{z_0 0}), (\frac{z_0 0}{1 z_0})$

$\square z_0 \square$ # $\square z_0 0 \# \square$

$\square z_0 \square$ # $\square z_0 0 \# \square 1 z_0 \square \# \square$

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

Imitation der Konfigurationsfolge - Beispiel

Initiale Konfiguration $\square z_0 \square$, Maschine M_w mit $\delta(z_0, \square) = (z_0, 0, N)$ und $\delta(z_0, 0) = (z_0, 1, R)$

$(\#_{\#\square z_0 \square \#}), (\#), (\square), (0^0), (1^1), (\#_{\square}), (\#_{\square \#}), (\widehat{z_0 \square}_{z_0 0}), (\widehat{z_0 0}_{1 z_0})$

$\#\square z_0 \square \# \square z_0 0 \# \square 1$

$\#\square z_0 \square \# \square z_0 0 \# \square 1 z_0 \square \# \square 1$

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

Imitation der Konfigurationsfolge - Beispiel

Initiale Konfiguration $\square z_0 \square$, Maschine M_w mit $\delta(z_0, \square) = (z_0, 0, N)$ und $\delta(z_0, 0) = (z_0, 1, R)$

$(\#_{\#\square z_0 \square \#}), (\#_{\#}), (\square_{\square}), (0^0_{0}), (1^1_{1}), (\#_{\#\square}), (\#_{\square\#}), (\frac{z_0 \square}{z_0 0}), (\frac{z_0 0}{1 z_0})$

$\#\square z_0 \square \# \square z_0 0 \# \square 1 z_0 \square$

$\#\square z_0 \square \# \square z_0 0 \# \square 1 z_0 \square \# \square 1 z_0 0$

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

Imitation der Konfigurationsfolge - Beispiel

Initiale Konfiguration $\square z_0 \square$, Maschine M_w mit $\delta(z_0, \square) = (z_0, 0, N)$ und $\delta(z_0, 0) = (z_0, 1, R)$

$(\#_{\# \square z_0 \square \#}), (\#), (\square), (0^0), (1^1), (\#_{\# \square}), (\#_{\square \#}), (\frac{z_0 \square}{z_0 0}), (\frac{z_0 0}{z_0})$

$\boxed{\square z_0 \square}$ # $\boxed{\square z_0 0 \#}$ | $\boxed{\square 1 z_0 \square \#}$
$\boxed{\square z_0 \square}$ # $\boxed{\square z_0 0 \#}$ | $\boxed{\square 1 z_0 \square \#}$ $\boxed{\square 1 z_0 0 \#}$

H_0 reduzierbar auf MPCP

Beweis (Skizze)

Reduktion f erhält den Codeworten 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

Imitation der Konfigurationsfolge - Beispiel

Initiale Konfiguration $\square z_0 \square$, Maschine M_w mit $\delta(z_0, \square) = (z_0, 0, N)$ und $\delta(z_0, 0) = (z_0, 1, R)$

$$(\#_{\square z_0 \square \#}), (\#), (\square), (0), (1), (\#_{\square}), (\#_{\square \#}), (\textcolor{red}{z_0 \square})_{z_0 0}, (\textcolor{red}{z_0 0})_{1 z_0}, (\textcolor{teal}{z_e \square})_{z_e}, (\textcolor{teal}{z_e 0})_{z_e}, (\textcolor{teal}{z_e 1})_{z_e}, (\textcolor{teal}{z_e \# \#})_{\#}.$$

#□z₀□#□z₀0#□1z₀□#

- 3 -

Ende

2e # # |||
2e # # |||

#□z₀□#□z₀0#□1z₀□#□1z₀0#

•

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

- | $(\#_{\# \square z_0 \square \#})$ - **initiale Konfiguration**
- | $(\overset{a}{a})$ für alle $a \in \Gamma$ – **Kopierregeln**
- | $(\#_{\#}), (\#_{\# \square}), (\square \#_{\#})$ – **Randregeln**
- | $(z_e a)_{z_e}$ für alle $a \in \Gamma$ – **Löschregeln**
- | $(z_e \# \#)_{\#}$ – **Abschlussregel**

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

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

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

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

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

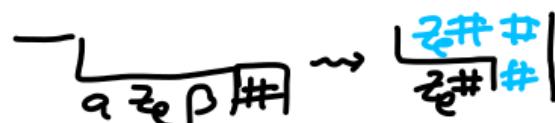
\rightsquigarrow es gibt Konfigurationenfolge

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

Simulation durch MPCP wie im Beispiel,

am Ende β entfernt durch Löschregel,

Abschluss durch Abschlussregel



$(\# \square z_0 \square \#)$ - **initiale Konfiguration**

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

$(\# \#), (\# \square), (\square \#)$ – **Randregeln**

$(z_e a)$ für alle $a \in \Gamma$ – **Löschregeln**

$(z_e \# \#)$ – **Abschlussregel**

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

$(z_i a)$ falls $\delta(z_i, a) = (z_j, b, N)$

$(z_i a)$ falls $\delta(z_i, a) = (z_j, b, R)$

$(a z_i b)$ falls $\delta(z_i, b) = (z_j, c, L)$

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 $M \in H_0 \Leftrightarrow f(\langle M \rangle) \in MPCP$

“ \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

z_e wird erreicht

$(\# \square z_0 \#)$ - 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

$$H \leq \underline{H_0} \leq \text{MPCP} \leq \text{PCP}$$

semi-entscheidbar

- ▶ PCP (und MPCP) sind unentscheidbar.
- ▶ H_0 ist semi-entscheidbar (und damit H und K)
- ▶ es gibt "universelle Turing-Maschine"

$$\underbrace{\{\underline{\omega} \# \underline{x} \mid x \in T(M_\omega)\}}_{\text{Semi entscheidbar}} \leq H$$

Gliederung

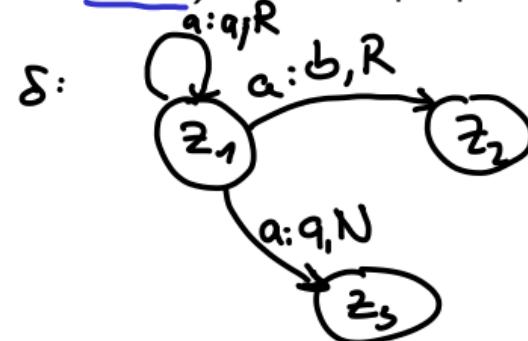
1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem
7. Aufzählbarkeit & (Semi-)Entscheidbarkeit
8. Reduzierbarkeit
9. Satz von Rice
10. Das Postsche Korrespondenzproblem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

Nichtdeterministische Turing-Maschinen I

Definition (Nichtdeterministische Turing-Machine)

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

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



Nichtdeterministische Turing-Maschinen I

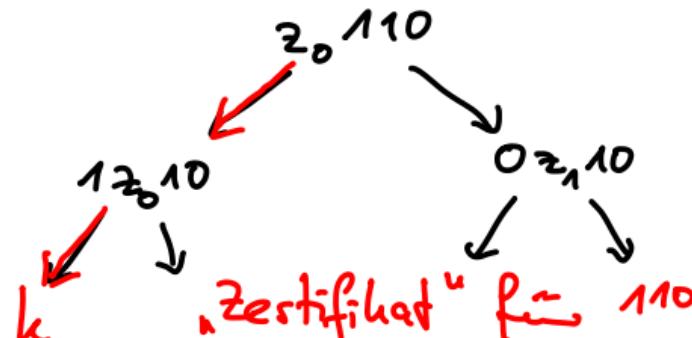
Definition (Nichtdeterministische Turing-Machine)

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)



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

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

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

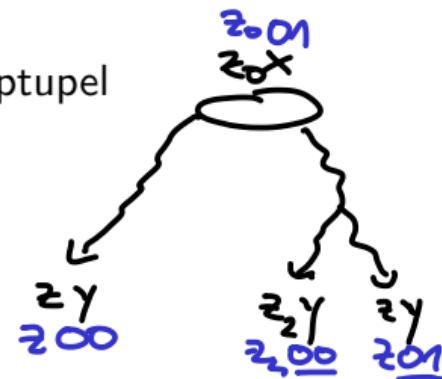
- ▶ $z_0 w \vdash_M^* k$ bedeutet: k' kann von Startkonfiguration erreicht werden (**Berechnungspfad**)
- ▶ haltende/akzeptierende Konfig., halten auf/akzeptieren von Wörtern analog zu DTM
- ▶ **Zertifikat** für w in $T(M)$ ist endlicher Pfad von $z_0 w$ in akzeptierende Konfiguration
- ▶ akzeptierte Sprache analog zu DTM: $T(M) := \{w \in \Sigma^* \mid \exists_{\alpha, \beta \in \Gamma^*} \exists_{z \in E} : z_0 w \vdash_M^* \underline{\alpha} z \underline{\beta}\}$

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 H_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 $f : \Sigma^* \rightarrow \Sigma^*$ sodass. für alle $x \in \Sigma^*$ und $y \in \Sigma^*$.

$$\underline{f(x) = y} \quad \Leftrightarrow \quad \underline{\{y' \in \Gamma^* \mid \exists_{z \in E} \underline{z_0 x} \vdash_M^* \underline{zy'}\}} = \{y\}$$

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)

Nichtdeterministische Turing-Maschinen II

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

Theorem

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

Nichtdeterministische Turing-Maschinen II

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

Theorem

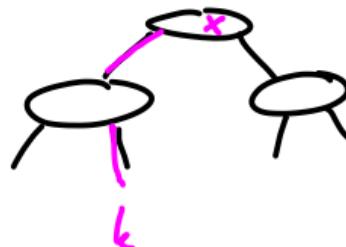
Für jede NTM N gibt es eine DTM M mit $T(M) = T(N)$.

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

traveling salesperson

naiv:
probieren alle Permutationen von
Städten

n Städte $\rightarrow n!$ Permutationen

\rightarrow Effizienz?

Polyonzeit



kürzeste Rundtour
„Optimierungsproblem“
„Entscheidungsproblem“
3 Route der Länge ≤ 100

$\{(D, k)\}$
es gibt eine Rundtour der
Länge $\leq k$ durch
alle Städte in $D\}$

(D, k) ?

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
- ▶ ...

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?

O -Notation zur Laufzeitanalyse

Problem: wie misst man Laufzeit von Algorithmen?

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

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

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

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

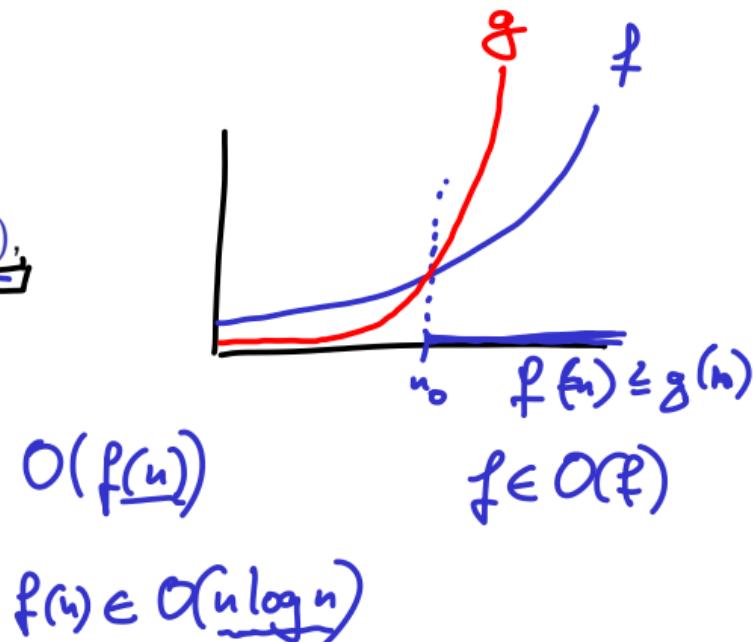
↪ "Landau-Symbole" / O -Notation

Definition

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

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

$$\text{zu } n \in O\left(\frac{1}{\sqrt{n}}\right)$$



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 $g \in O(f)$,

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

- ▶ $10\sqrt{n}$

O -Notation zur Laufzeitanalyse

Problem: wie misst man Laufzeit von Algorithmen?

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

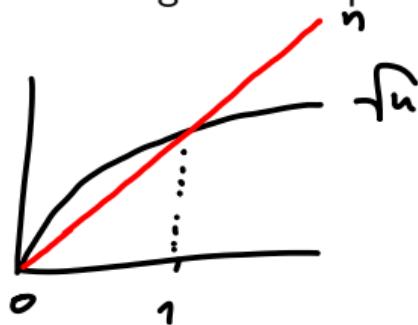
Ziel: "Effizienz" von Algorithmen unabhängig von Rechentechnik & Programmiersprache

↪ "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

- ▶ $10\sqrt{n} \in O(n)$

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

- ▶ $10\sqrt{n} \in O(n)$
- ▶ $2n$

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

- ▶ $10\sqrt{n} \in O(n)$
- ▶ $2n \in \Theta(n)$

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 $g \in O(f)$,

$$\log_2 n = \log_{10} n \cdot \log_{10} 2$$
$$\log_2 n \in \Theta(\log_{10} n)$$

Beispiele

- ▶ $10\sqrt{n} \in O(n)$
- ▶ $2n \in \Theta(n)$
- ▶ $\log_2 n$

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

- ▶ $10\sqrt{n} \in O(n)$
- ▶ $2n \in \Theta(n)$
- ▶ $\log_2 n \in O(\sqrt{n})$

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
- ▶ $2n \in \Theta(n)$
- ▶ $\log_2 n \in O(\sqrt{n})$

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,

$$10^6 \leq 10^6 \cdot 1$$

- ▶ $f \in O(g)$ 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

- ▶ $10\sqrt{n} \in O(n)$
- ▶ $10^6 \in \Theta(1)$
- ▶ $2n \in \Theta(n)$
- ▶ $\log_2 n \in O(\sqrt{n})$

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

- ▶ $10\sqrt{n} \in O(n)$
- ▶ $10^6 \in \Theta(1)$
- ▶ $2n \in \Theta(n)$
- ▶ $n \log_2 n$
- ▶ $\log_2 n \in O(\sqrt{n})$

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

- ▶ $10\sqrt{n} \in O(n)$
- ▶ $10^6 \in \Theta(1)$
- ▶ $2n \in \Theta(n)$
- ▶ $n \log_2 n \in O(n^{\frac{1}{2}})$
- ▶ $\log_2 n \in O(\sqrt{n})$

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

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

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

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

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} |
| ▶ $2n \in \Theta(n)$ | ▶ $n \log_2 n \in O(n^2)$ | |
| ▶ $\log_2 n \in O(\sqrt{n})$ | ▶ $n\sqrt{n} \in O(n^2)$ | |

O -Notation zur Laufzeitanalyse

Problem: wie misst man Laufzeit von Algorithmen?

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

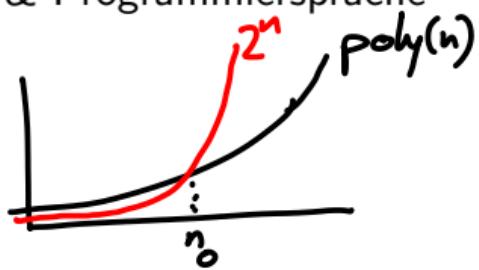
Ziel: "Effizienz" von Algorithmen unabhängig von Rechentechnik & Programmiersprache

↪ "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

- | | | |
|------------------------------|---------------------------|---|
| ▶ $10\sqrt{n} \in O(n)$ | ▶ $10^6 \in \Theta(1)$ | ▶ <u>$n^{10} \in O(2^n)$</u> |
| ▶ $2n \in \Theta(n)$ | ▶ $n \log_2 n \in O(n^2)$ | |
| ▶ $\log_2 n \in O(\sqrt{n})$ | ▶ $n\sqrt{n} \in O(n^2)$ | |

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

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

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

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

Deterministische Zeitklassen

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

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).

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,

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,

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

Deterministische Zeitklassen

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

Für jede (Mehrband-) DTM M sei $\underline{\text{time}}_M(n)$ 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 (\mathbf{P})

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

"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 .

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,

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,

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

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)

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

“**nicht**deterministisch, in Polynomzeit”

Nichtdeterministische Zeitklassen

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

Für jede (Mehrband-) **NTM** N sei $\underline{\text{time}}_N(n)$ 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 NP klar, da jede DTM eine NTM ist.

$$\text{DTIME}(f(n)) \subseteq \text{NTIME}(f(n))$$

Alternative Definition von NP

„raten“ ein Zertifikat

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

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

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 $T(N) = L$.

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

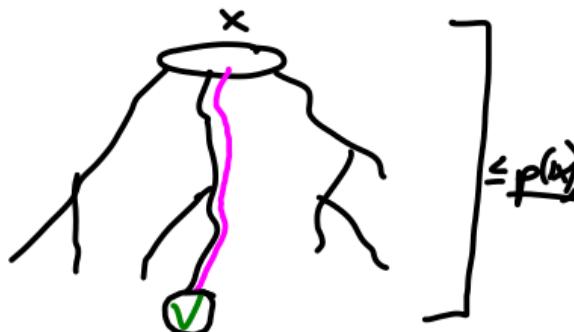
$$x \in L \Leftrightarrow \exists_{u \in \Sigma^{p(|x|)}} \langle x, u \rangle \in T(M).$$

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 $T(N)$.

$$\begin{cases} x \in L \Rightarrow \exists_{u \in \Sigma^{p(|x|)}} \langle x, u \rangle \in T(N) \\ x \notin L \Rightarrow \forall_{u \in \Sigma^{p(|x|)}} \langle x, u \rangle \notin T(N) \end{cases}$$



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.

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.

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.

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.

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.

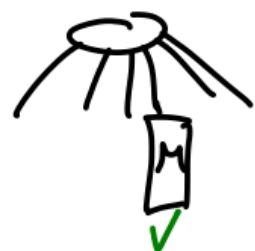
“ \Leftarrow ”: Sei M eine DTM wie im Theorem, zeitbeschränkt durch Polynom q .

Wir konstruieren eine NTM N die:

1. das Zertifikat u der Länge $p(|x|)$ nichtdeterministisch erzeugt (“räts”) 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

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



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 .

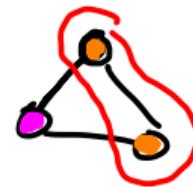
Wir konstruieren eine NTM N die:

1. das Zertifikat u der Länge $p(|x|)$ nichtdeterministisch erzeugt (“räts”) 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 $G = (\mathcal{V}, \mathcal{E})$

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?

3-Coloring =

$\{G \mid$ die Knoten von G lassen sich mit 3 Farben färben sd. $\}$

$\{\text{Eingabe} \mid \text{Frage} = "J\bar{a}\}$

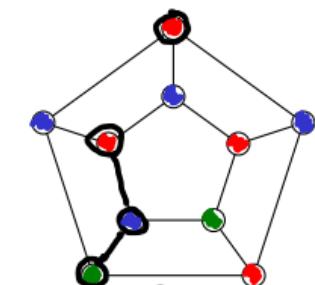
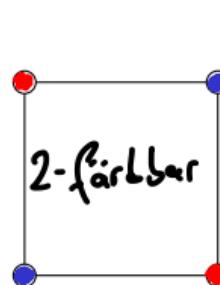
3-COLORING versus 2-COLORING

3-Coloring (2-Coloring)

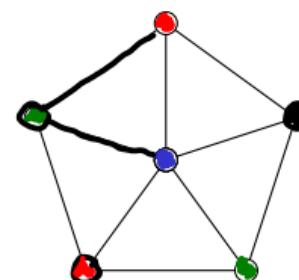
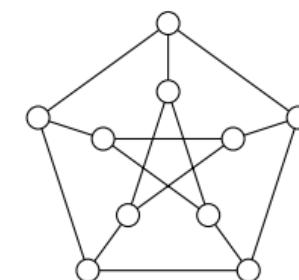
Eingabe: ungerichteter Graph $G = (V, E)$

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?



~~2-färbbar~~



3-färbbar?

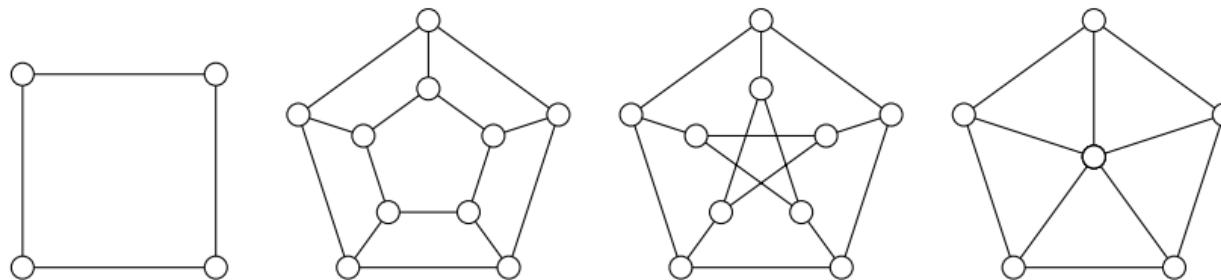
3-COLORING versus 2-COLORING

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 ?

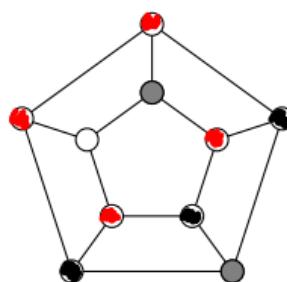
Longest Path versus Shortest Path

Shortest Path (Longest Path)

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

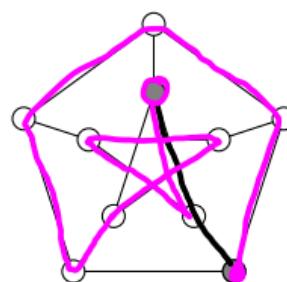
Frage: Existiert ein „einfacher“ Pfad zwischen s und t der Länge höchstens (mind.) k ?

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

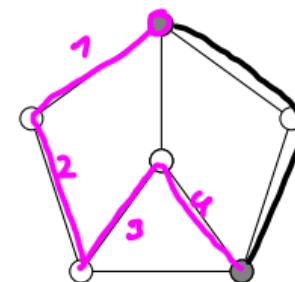


shortest path = 3

longest path = ?



longest path = 9



longest path = 4

Longest Path versus Shortest Path

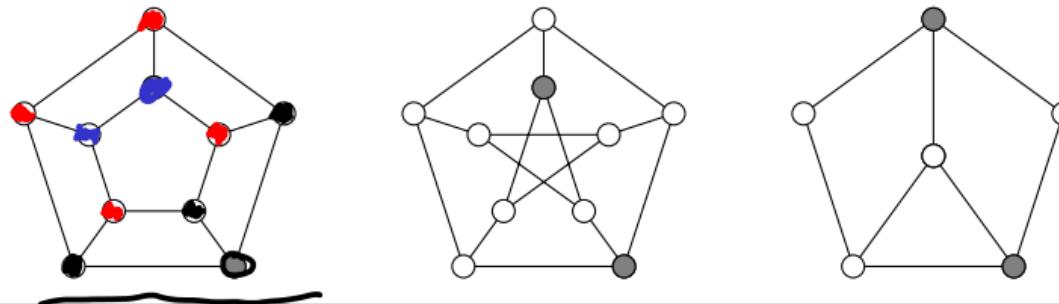
Shortest Path (Longest Path)

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)?

$\rightarrow 1$ Schritt
 $\rightarrow 2$ -
 $\rightarrow 3$ -



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.

↳ Konjunktion u. Disjunktion

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?

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

►
$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_3 \vee x_4) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_2 \vee x_3 \vee x_4)$$

$\#^1 \quad (x_2 = 0)$

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).

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

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

$$x_2 = 0$$

$$x_3 = 0$$

$$\Rightarrow x_1 = 0$$

↖

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: $P \stackrel{?}{=} NP$.

$$P \subseteq \underline{NP}$$

$$P \supseteq NP \ ?$$

verbreitete Vermutung:

$$P \neq NP$$

$3\text{-SAT} \rightsquigarrow 2^{\# \text{variables}}$ Zertifikate

"schwerste Probleme in NP"

- 3-SAT
- Longest path
- 3-coloring
- TSP
-

} det. poly-zeit
 $\Rightarrow P = NP$

P versus NP

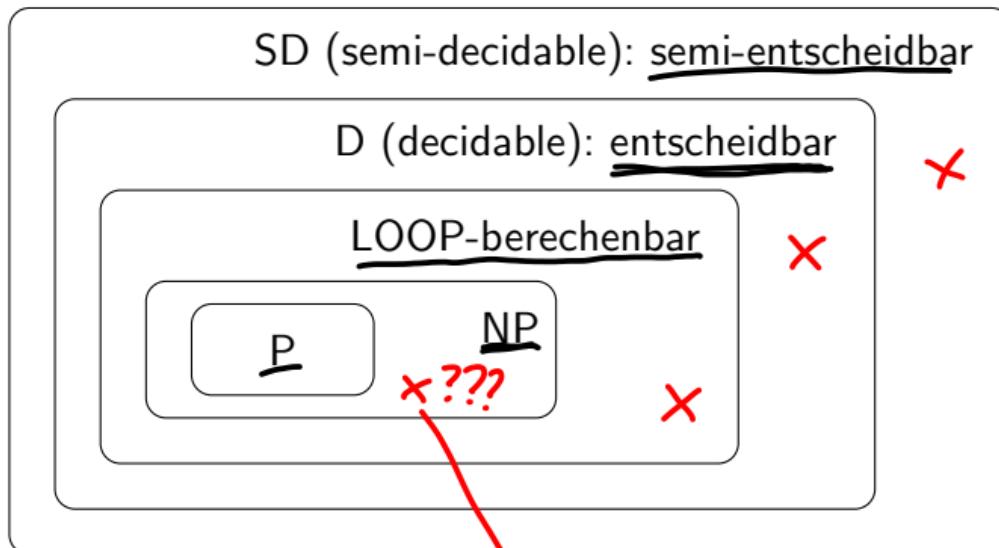
Die bekannteste offene Frage der (Theoretischen) Informatik ist: $P \stackrel{?}{=} NP$.

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

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. PSPACE

Polynomzeitreduktion I

Vielleicht das wichtigste Konzept der Komplexitätstheorie!

Definition

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

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

Σ^*
"

„Reduktionseigenschaft“

Wir nennen f eine

Reduktion von A auf B

(**Beachte:** f muss weder surjektiv noch injektiv sein).

Polynomzeitreduktion I

Vielleicht das wichtigste Konzept der Komplexitätstheorie!

\leq_m^P „many one“

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).

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“.

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 $A \leq B$

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) $A \leq_m^p B \Rightarrow A \leq B$

$$f_{AB}$$

$$f_{BC}$$

$$f_{BC} \circ f_{AB}$$

(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)

$$\frac{\begin{array}{c} f_{BC}(f_{AB}(x)) \\ \leq \text{poly}(x) \end{array}}{\text{poly}(\text{poly}(x))}$$

$$\frac{x \in A \Leftrightarrow f_{AB}(x) \in B \Leftrightarrow f_{BC}(f_{AB}(x)) \in C}{\text{NP-Vollständigkeit}}$$

Polynomzeitreduktion II

Lemma

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

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. $A \in NP$) .

„leicht“

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. $A \in NP$) .

①

②

Beweis

1. $A \leq_m^p B \rightsquigarrow$ „Reduktionsfunktion“ f in $p(n)$ Schritten berechenbar durch TM M_f

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. $\underline{B \in P}$ (bzw. $\underline{B \in NP}$) $\rightsquigarrow B$ in $q(n)$ Schritten entscheidbar durch TM $\underline{M_B}$
(wobei p und q Polynome)

Polynomzeitreduktion II

Lemma

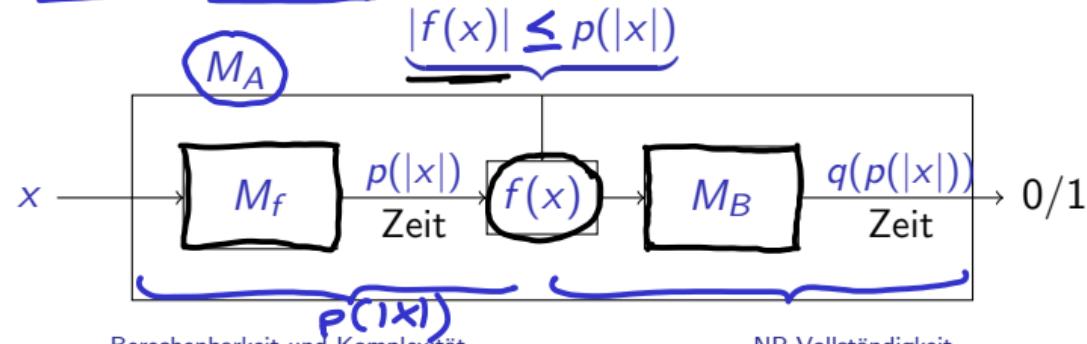
Gilt $A \leq_m^p B$ und ist $B \in P$ (bzw. $B \in NP$), so ist auch $\underline{A \in P}$ (bzw. $\underline{A \in NP}$) .

Beweis

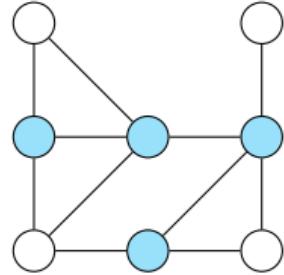
1. $A \leq_m^p B \sim$ „Reduktionsfunktion“ f in $p(n)$ Schritten berechenbar durch TM M_f
2. $B \in P$ (bzw. $B \in NP$) $\sim B$ in $q(n)$ Schritten entscheidbar durch TM M_B
(wobei p und q Polynome)

Wie zuvor gilt $\underline{\chi_A} = \underline{\chi_B \circ f}$

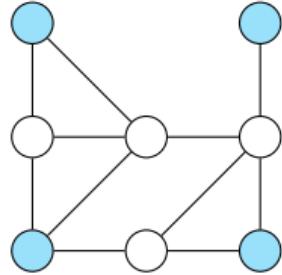
$\sim \underline{\chi_A}$ berechnet in $\underline{p(|x|)} + \underline{q(p(|x|))}$ (also polynomiell viele) Schritten.



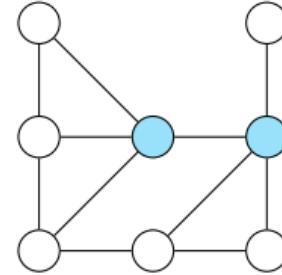
INDEPENDENT SET, VERTEX COVER und DOMINATING SET



VERTEX COVER



INDEPENDENT SET

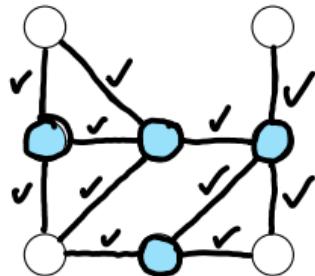


DOMINATING SET

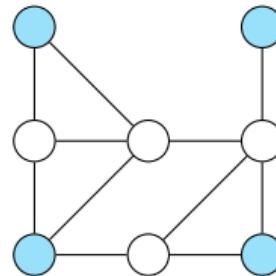
Eingabe: ungerichteter Graph G , Zahl $k > 0$

$\langle G, k \rangle$

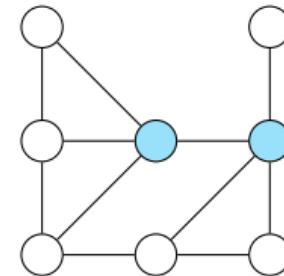
INDEPENDENT SET, VERTEX COVER und DOMINATING SET



VERTEX COVER



INDEPENDENT SET



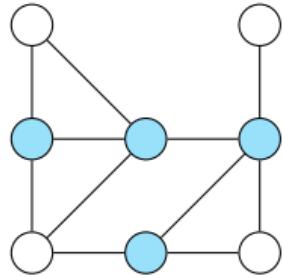
DOMINATING SET

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

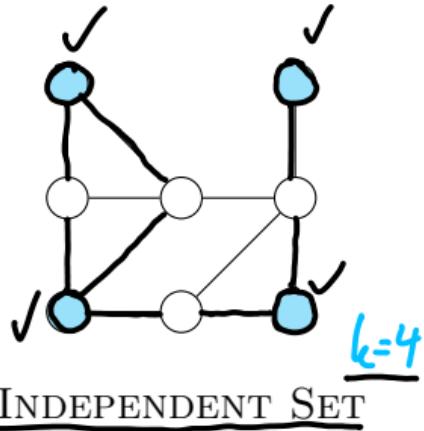
Frage: gibt es k Knoten in G , sodass ...

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

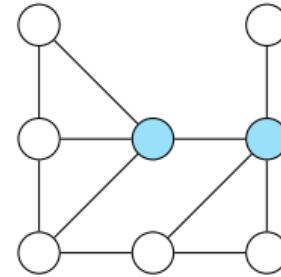
INDEPENDENT SET, VERTEX COVER und DOMINATING SET



VERTEX COVER



INDEPENDENT SET



DOMINATING SET

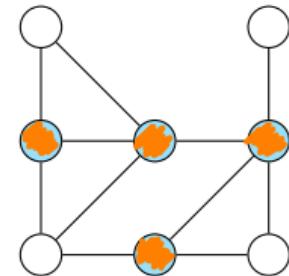
Eingabe: ungerichteter Graph G , Zahl $k > 0$

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?

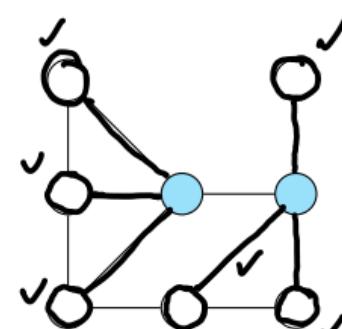
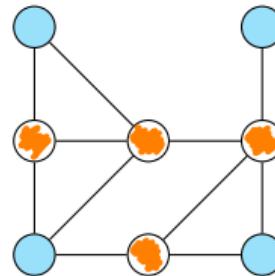
INDEPENDENT SET, VERTEX COVER und DOMINATING SET



VERTEX COVER



INDEPENDENT SET



DOMINATING SET

$k=2$

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.

VERTEX COVER und INDEPENDENT SET

Theorem

VERTEX COVER \leq_m^p INDEPENDENT SET.

$V(G) = \text{Knotenmenge von } G_f$

Beweis

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

VERTEX COVER und INDEPENDENT SET

Theorem

VERTEX COVER \leq_m^p INDEPENDENT SET.

$\exists e \forall u \in X \exists v \in V(G) \setminus X : e \in \{u, v\}$

$\exists e \forall u \in X \exists v \in V(G) \setminus X : e \in \{u, v\}$

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 $\underline{A \subseteq \Sigma^*}$ heißt...

a) ... **NP-schwer**, falls $\forall_{L \in \text{NP}} \underline{L \leq_m^p A}$.

problem

Halteproblem?

frage:

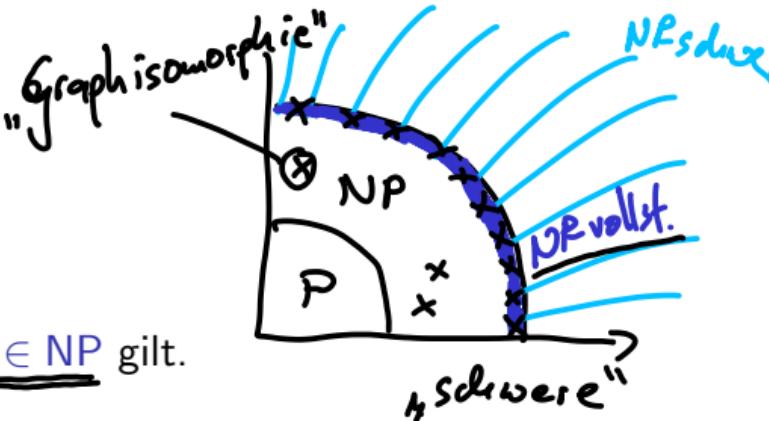
können Sie zeigen, dass das
Halteproblem NP-schwer ist?

NP-Vollständigkeit

Definition

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

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



NP-Vollständigkeit

$$\underline{SAT} \leq_p^{\textcircled{P}} \underline{VC} \leq_m^{\textcircled{P}} \underline{SAT}$$

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“)

1. NP-schwere Sprachen sind „mindestens so schwer“ zu entscheiden wie jede Sprache in NP
- ② NP-vollständige Sprachen sind „genau so schwer“ wie jede NP-vollständige Sprache

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: $A \in P \Leftrightarrow P = NP$.

NP-Vollständigkeit II

" \leq_m^P " ≡ "leichter"

Theorem

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

①

②

③

$NP \supseteq P$

$NP \subseteq P$

Beweis

„ \Rightarrow “: $(\forall L \in NP \ L \leq_m^P A) \wedge (A \in P)$ $\xrightarrow{\text{Lemma}}$ $\forall L \in NP \ L \in P \Rightarrow NP = P$

①

②

„ \Leftarrow “: $(A \in NP) \wedge (P = NP) \Rightarrow A \in P$

③

NP-Vollständigkeit II

Theorem

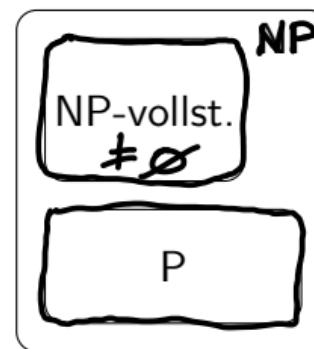
Für jede NP-vollständige Sprache A gilt: $\underline{A \in P \Leftrightarrow P = NP}$.

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 I

Cook & Levin

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?

Erfüllbarkeitsproblem I

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, \bar{x}_3 ,

$(x_1 \wedge \bar{x}_2)$,

$((\bar{x}_1 \wedge \bar{x}_2) \vee x_2 \vee \bar{x}_3)$

Erfüllbarkeitsproblem I

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.

Erfüllbarkeitsproblem I

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.

guess & check

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 $x \in L \Leftrightarrow \varphi(x) \in \text{SAT}$.

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. PSPACE

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?

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, \bar{x}_3 ,

$(x_1 \wedge \bar{x}_2)$,

$((\bar{x}_1 \wedge \bar{x}_2) \vee x_2 \vee \bar{x}_3)$

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.

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.

"guess & check"

Beweis (Idee, Details später)

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

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

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?

Konjunktion v. Disjunktionen
 $(x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge \dots$

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)

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: φ \rightsquigarrow erfüllbarkeits-äquivalente Formel ψ :

CNF-SAT ist NP-vollständig

$$x \rightarrow y = \bar{x} \vee y$$

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 (\leadsto CNF-SAT NP-vollständig)

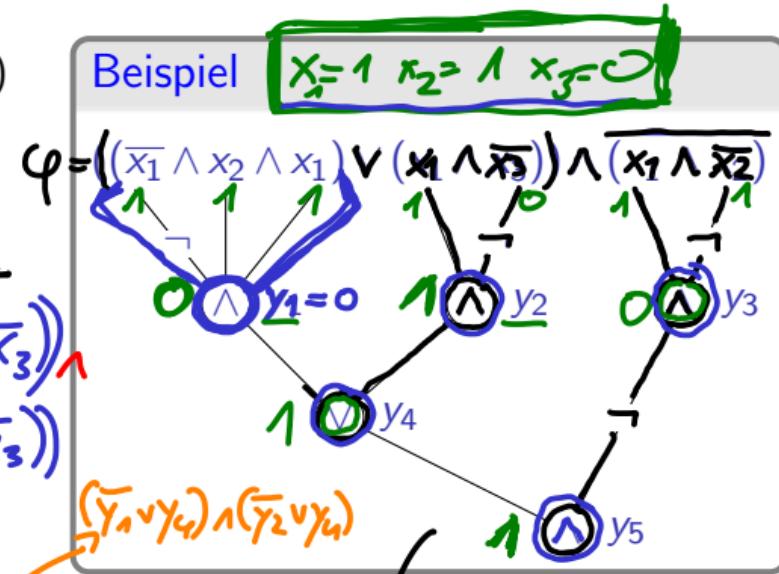
Beweis (Skizze)

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

$$\psi = (y_1 \leftrightarrow (\bar{x}_1 \wedge x_2 \wedge x_3)) \wedge (y_2 \leftrightarrow (x_1 \wedge \bar{x}_3)) \wedge \dots$$

$$\wedge y_5 \wedge (y_3 \leftrightarrow \dots) \wedge (y_4 \leftrightarrow (y_1 \vee y_2)) \wedge (y_5 \leftrightarrow (y_4 \wedge \bar{y}_3))$$

$$y_4 \leftrightarrow (y_1 \vee y_2) \equiv (y_4 \rightarrow (y_1 \vee y_2)) \wedge ((y_1 \vee y_2) \rightarrow y_4)$$
$$(\bar{y}_4 \vee y_1 \vee y_2) \wedge ((\bar{y}_1 \wedge \bar{y}_2) \vee y_4)$$



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 (\leadsto CNF-SAT NP-vollständig)

Beweis (Skizze)

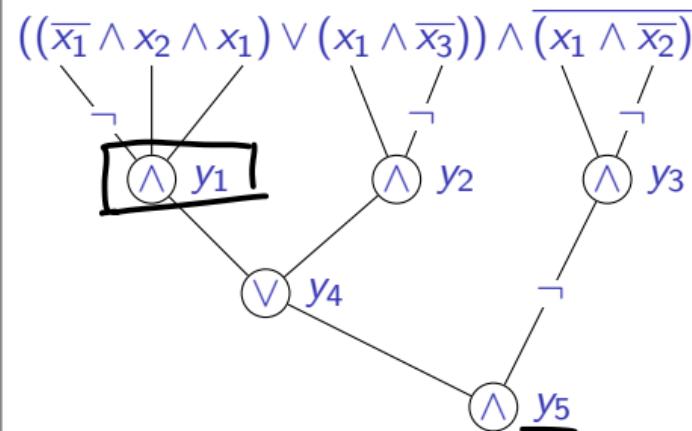
Reduktion: $\varphi \leadsto$ erfüllbarkeits-äquivalente Formel ψ :

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

mit “äquivalentem Wahrheitswert”

(2) neue Klausel für die Wurzel

Beispiel



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 (\leadsto CNF-SAT NP-vollständig)

Beweis (Skizze)

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

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

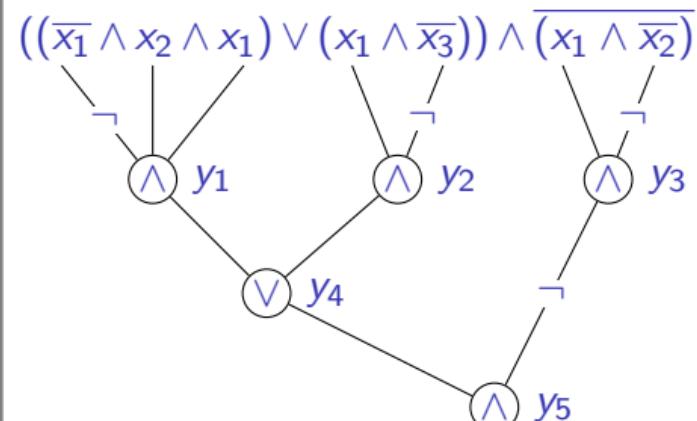
mit “äquivalentem Wahrheitswert”

(2) neue Klausel für die Wurzel

ψ erfüllbar \Leftrightarrow φ erfüllbar ✓

\hookrightarrow CNF

Beispiel



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 (\leadsto CNF-SAT NP-vollständig)

Beweis (Skizze)

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

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

mit “äquivalentem Wahrheitswert”

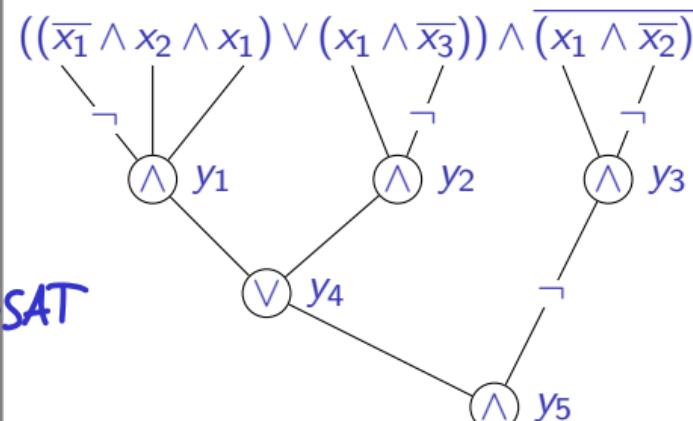
(2) neue Klausel für die Wurzel

ψ erfüllbar $\Leftrightarrow \varphi$ erfüllbar ✓

$\varphi \in \text{SAT} \Leftrightarrow \psi \in \text{CNF-SAT}$

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).

Formel in 3-KNF

$(x_1 \vee \overline{x}_2 \vee x_3) \wedge (\dots) \wedge (\dots)$
 ≤ 3 Literale

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

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;

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 $c_j = (\underline{\ell_1} \vee \underline{\ell_2} \vee \dots \vee \underline{\ell_r}) \in \varphi$,

► falls $r \leq 3$, dann füge c_j zu ψ hinzu;

► sonst füge c'_j hinzu mit

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

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

$$\rightarrow \text{Länge } r-2+1 = \underline{r-1}$$

„ \Rightarrow “ c_j erfüllt $\rightarrow \exists$ Literal ℓ_i s.d. $\beta(\ell_i) = 1$. Fall 1: $i \leq 2 \rightarrow \beta \cup \{(\underline{y_1}, 0)\}$ erfüllt
von β Fall 2: $i > 2 \rightarrow \beta \cup \{(\overline{y_{i-1}}, 1)\}$ erfüllt

„ \Leftarrow “ c'_j erfüllt v. β . Fall 1: $\beta(y_1) = 0 \rightarrow \beta(\ell_1) = 1$ oder $\beta(\ell_2) = 1 \rightarrow \beta$ erfüllt c_j
Fall 2: $\beta(y_1) = 1 \rightarrow \exists i \geq 2 \beta(\ell_i) = 1 \rightarrow \beta$ erfüllt c_j

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

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 φ erfüllbar ✓

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: $|\psi| \leq \underline{3} |\varphi|$

VERTEX COVER ist NP-vollständig



Theorem

3-SAT \leq_m^p VERTEX COVER.

VERTEX COVER ist NP-vollständig

Theorem

3-SAT \leq_m^P VERTEX COVER.

Beweis (Skizze)

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

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

VERTEX COVER ist NP-vollständig

Theorem

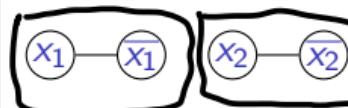
3-SAT \leq_m^P VERTEX COVER.

Beweis (Skizze)

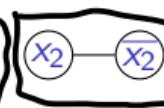
Formel $\varphi \rightsquigarrow (G, k = \#\text{Var} + 2\#\text{Klauseln})$

1. Variablen-Gadget: Variable $x_i \rightsquigarrow$ 2 benachbarte Knoten mit Beschriftungen x_i und \bar{x}_i

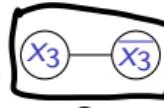
Beispiel: $(x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$



VG v. x_1



VG v. x_2



} Variablen-Gadgets

VG v. x_3

VERTEX COVER ist NP-vollständig

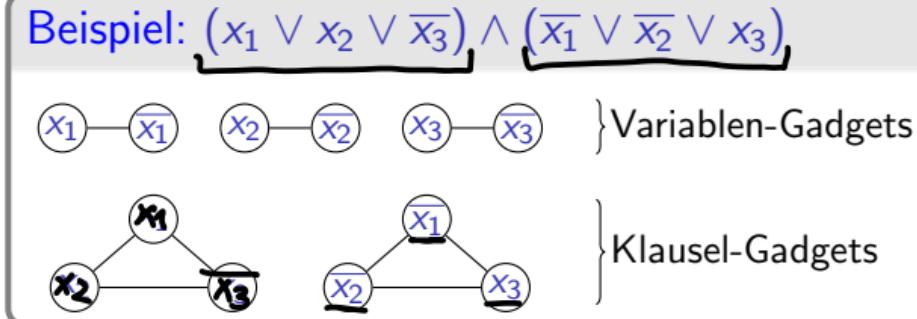
Theorem

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 $(\ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3}) \rightsquigarrow$ Dreieck mit Beschriftungen $\underline{\ell_{i_1}, \ell_{i_2}, \ell_{i_3}}$



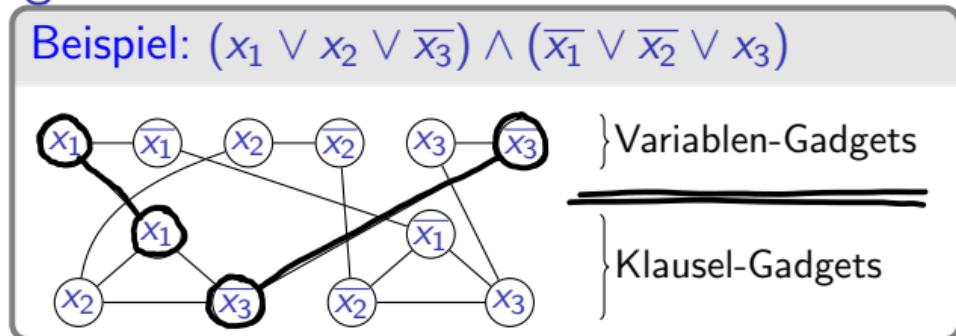
VERTEX COVER ist NP-vollständig

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 $(\ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3}) \rightsquigarrow$ Dreieck mit Beschriftungen $\ell_{i_1}, \ell_{i_2}, \ell_{i_3}$

3. Verbinde Knoten mit gleicher Beschriftung zwischen Var-Gadgets & Klausel-Gadgets

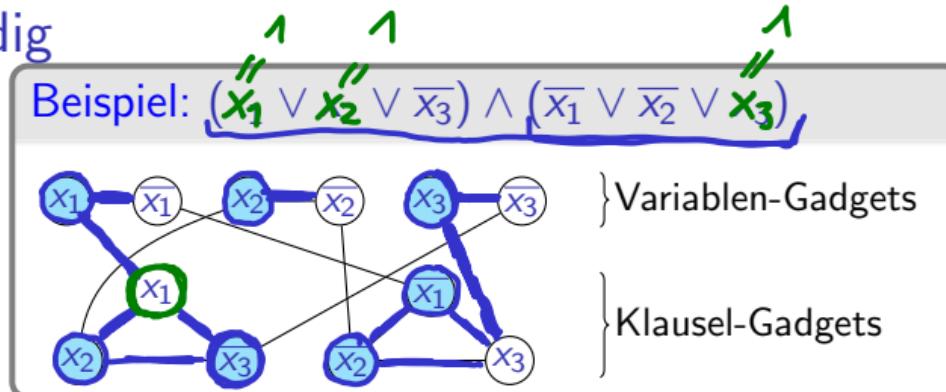
VERTEX COVER ist NP-vollständig

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 Knoten mit 2 Knoten aus jedem Klausel-Gadget überdeckt

- ① Kanten in Var.-Gadgets ✓
- ② Kanten in Klausel-Gadgets ✓
- ③ Kanten zwischen VGS & KGS ✓

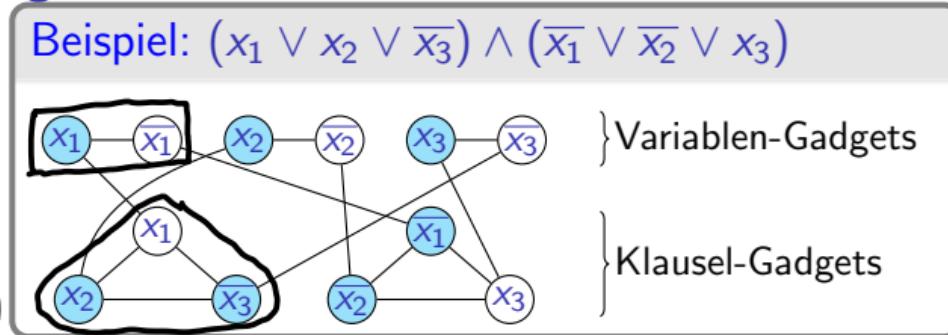
VERTEX COVER ist NP-vollständig

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 $(\ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3}) \rightsquigarrow$ Dreieck mit Beschriftungen $\ell_{i_1}, \ell_{i_2}, \ell_{i_3}$
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.

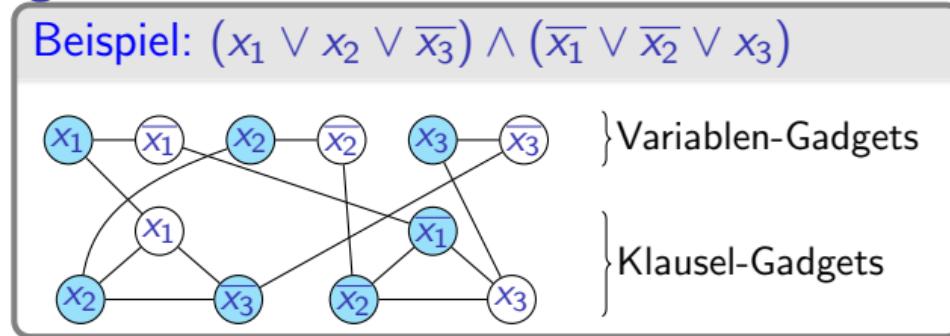
VERTEX COVER ist NP-vollständig

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 $(\ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3}) \rightsquigarrow$ Dreieck mit Beschriftungen $\ell_{i_1}, \ell_{i_2}, \ell_{i_3}$
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.

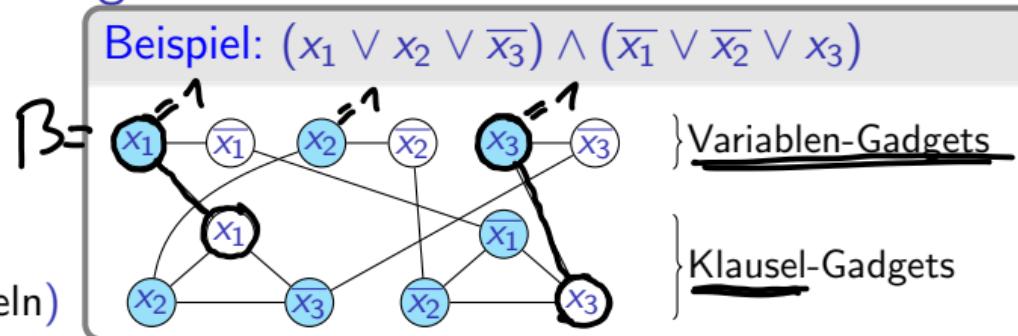
VERTEX COVER ist NP-vollständig

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 $(\ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3}) \rightsquigarrow$ Dreieck mit Beschriftungen $\ell_{i_1}, \ell_{i_2}, \ell_{i_3}$
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

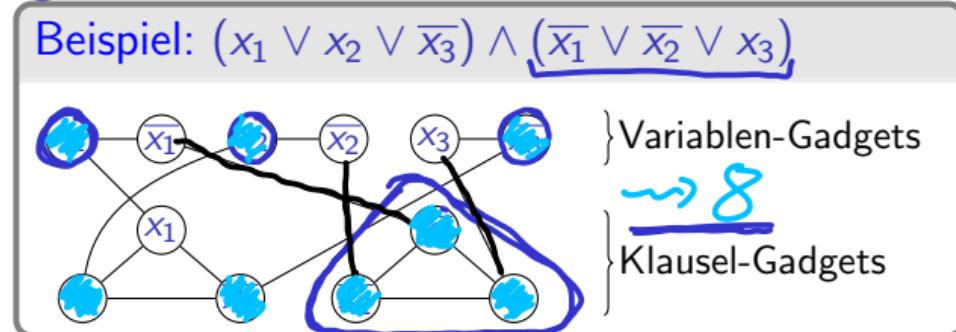
VERTEX COVER ist NP-vollständig

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 $(\ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3}) \rightsquigarrow$ Dreieck mit Beschriftungen $\ell_{i_1}, \ell_{i_2}, \ell_{i_3}$
3. Verbinde Knoten mit gleicher Beschriftung

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

\rightsquigarrow alle anderen Knoten 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.

DOMINATING SET ist NP-vollständig

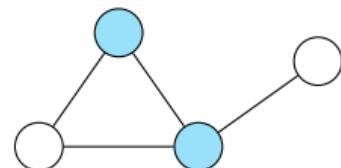
Theorem

VERTEX COVER \leq_m^p DOMINATING SET.

Beweis (Skizze)

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

Beispiel



$k = 2$

VERTEX COVER

$k = 2$

DOMINATING SET

DOMINATING SET ist NP-vollständig

Theorem

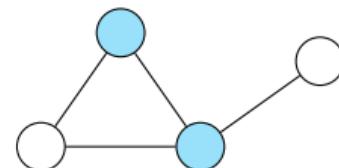
VERTEX COVER \leq_m^p DOMINATING SET.

Beweis (Skizze)

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

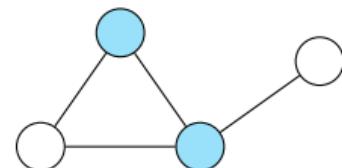
1. setze initial $G' = G$

Beispiel



$$k = 2$$

VERTEX COVER



$$k = 2$$

DOMINATING SET

DOMINATING SET ist NP-vollständig

Theorem

VERTEX COVER \leq_m^p DOMINATING SET.

Beweis (Skizze)

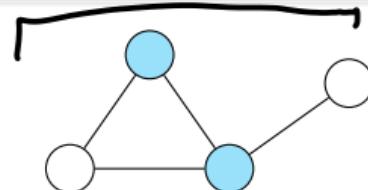
$$(G, k) \sim (G', k)$$

1. setze initial $G' = G$

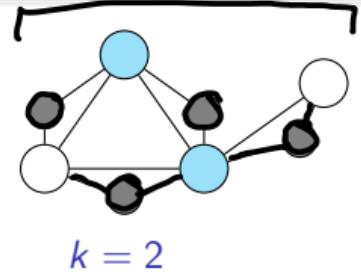
2. für jede Kante $e = \{u, v\}$ in G :

erzeuge einen neuen (grauen) Knoten in G' und verbinde ihn mit u und v

Beispiel



VERTEX COVER



DOMINATING SET

DOMINATING SET ist NP-vollständig

Theorem

VERTEX COVER \leq_m^p DOMINATING SET.

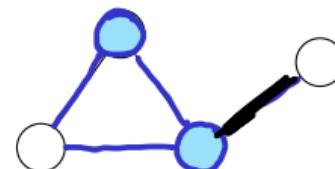
Beweis (Skizze)

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

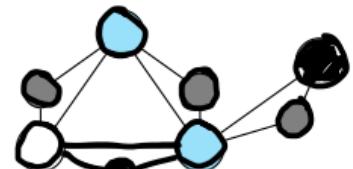
1. setze initial $G' = G$

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

Beispiel



VERTEX COVER



DOMINATING SET

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

DOMINATING SET ist NP-vollständig

Theorem

VERTEX COVER \leq_m^p DOMINATING SET.

Beweis (Skizze)

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

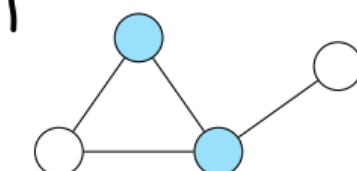
1. setze initial $G' = G$

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

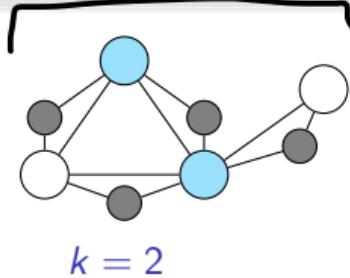
Korrektheit: „ \Rightarrow “: VC-Lösung in G ist auch DS-Lösung in G'

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

Beispiel



VERTEX COVER



DOMINATING SET

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

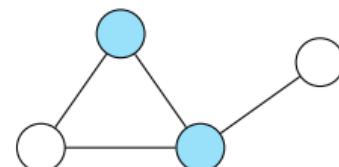
Korrektheit: „ \Rightarrow “: VC-Lösung in G ist auch DS-Lösung in G'

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

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

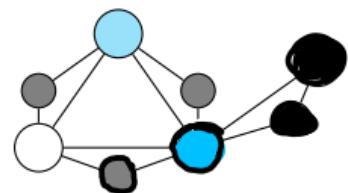
Lösung ohne graue Knoten

Beispiel



$$k = 2$$

VERTEX COVER



$$k = 2$$

DOMINATING SET

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 $|X| \leq k$

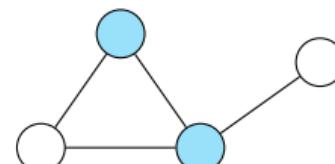
(a) neuer (grauer) Knoten \in DS-Lösung \rightsquigarrow mit weißem Nachbarn tauschen

\rightsquigarrow Lösung ohne graue Knoten

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

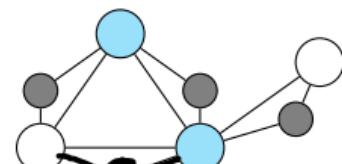
$\rightsquigarrow X$ ist vertex cover in G

Beispiel



$$k = 2$$

VERTEX COVER



$$k = 2$$

DOMINATING SET

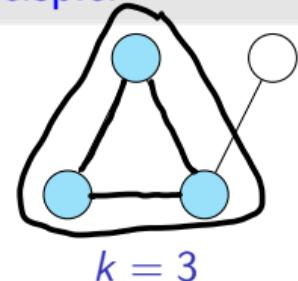
CLIQUE ist NP-vollständig

Clique

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

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

Beispiel



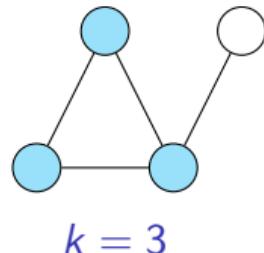
CLIQUE ist NP-vollständig

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.

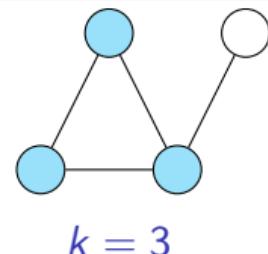
CLIQUE ist NP-vollständig

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.

ungeordnete
Knotenpaare

Beweis (Skizze)

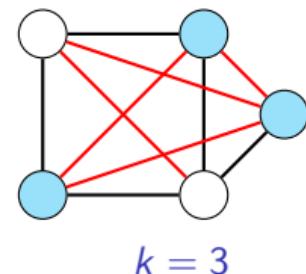
$$(G = (V, E), k) \sim (\bar{G} = (V, \binom{V}{2} \setminus E), k)$$

Korrektheit:

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

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

Beispiel



keine Kante zwischen
u & v in \bar{G}
Kante zwischen
u & v in \bar{G}

INDEPENDENT SET
CLIQUE

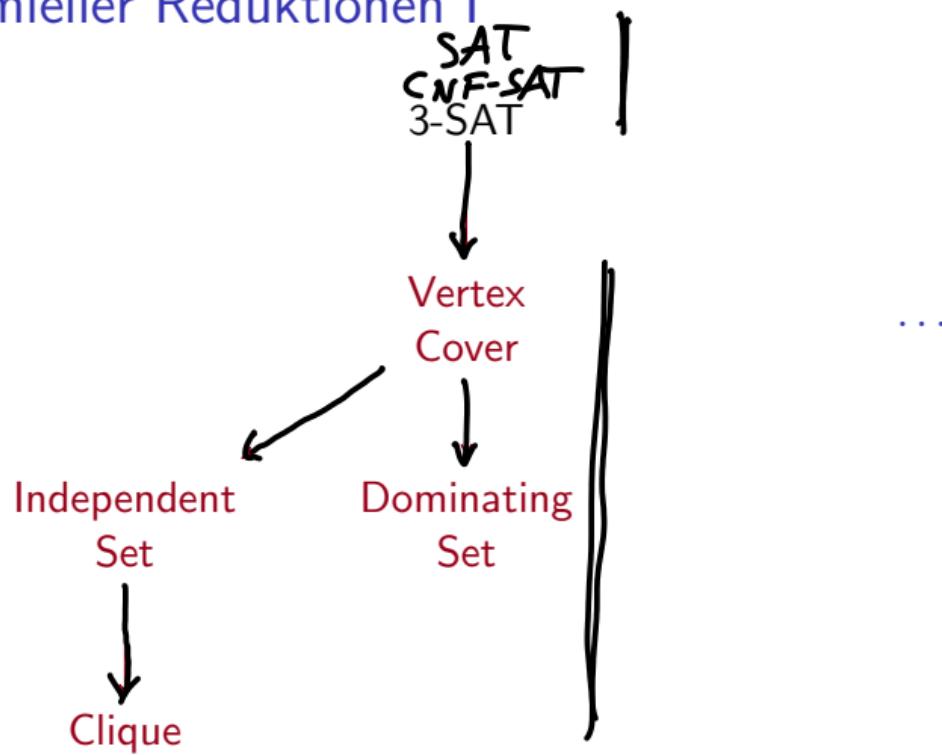
Wenn ein "Dominostein" fiele...

$SAT \leq_m^P 3-SAT \leq_m^P VC \leq_m^P IS \leq_m^P CLIQUE \leq_m^P SAT$

\downarrow
 ϵP ?
?



Netzwerk polynomieller Reduktionen I



HITTING SET und SET COVER

Eingabe:

- (1) Grundmenge („Universum“) $\underline{U} := \{x_1, x_2, \dots, x_n\}$,
- (2) eine Teilmengenfamilie $\mathcal{F} := \{\underline{S}_1, \underline{S}_2, \dots, \underline{S}_m\}$ mit $\underline{S}_i \subseteq \underline{U}$ für $1 \leq i \leq m$ und
- (3) ein $\underline{k} \in \mathbb{N}$

HITTING SET und SET COVER

Eingabe:

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

Hitting Set

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

HITTING SET und SET COVER

Eingabe:

- (1) Grundmenge („Universum“) $\underline{U} := \{x_1, x_2, \dots, x_n\}$,
- (2) eine Teilmengenfamilie $\underline{\mathcal{F}} := \{\underline{S}_1, \underline{S}_2, \dots, \underline{S}_m\}$ 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 $\underline{\mathcal{Z}} \subseteq \underline{\mathcal{F}}$ mit $\underline{|\mathcal{Z}| \leq k}$ und $\underline{\bigcup_{S \in \mathcal{Z}} S = U}$?

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 $1 \leq i \leq m$ 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) $U = \{1, 2, 3, 4, 5, 6\}$, $1, 2, 3, 4, 5, 6$
- (2) $S_1 = \{1, 3\}$, $S_2 = \{3, 4\}$, $S_3 = \{1, 5\}$, $S_4 = \{2, 4, 6\}$, $S_5 = \{1, 3, 5\}$
- (3) $k = 2$

$$S_4 \cup S_5 = U$$

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.

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“.

HITTING SET ist NP-vollständig

Theorem

VERTEX COVER \leq_m^p HITTING SET.

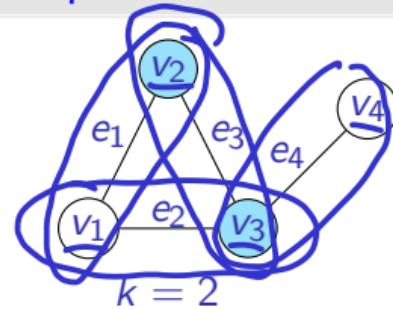
Beweis (Skizze)

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

Korrektheit: klar

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

Beispiel



$$\begin{aligned} U &= \{v_1, v_2, v_3, v_4\} \\ \mathcal{F} &= \{\{v_1, v_2\}, \{v_1, v_3\}, \\ &\quad \{v_2, v_3\}, \{v_3, v_4\}\} \end{aligned}$$

SET COVER ist NP-vollständig

Theorem

HITTING SET \leq_m^p SET COVER.

SET COVER ist NP-vollständig

Theorem

HITTING SET \leq_m^p SET COVER.

		HS/SC Dualität			
		1	2	3	4
$\mathcal{F} \setminus U$	1	○			
	{1, 2}	■	■	■	■
$\{1, 4\}$	■	■	■	■	■
	{2, 3}	■	■	■	■
$SC(\downarrow) \Leftarrow \equiv HS(\uparrow)$					
		1	2	3	4
$U \setminus \mathcal{F}$	1	{1, 2}	{1, 4}	{2, 3}	{2, 3, 4}
	2				
3	3				
	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 $F_x := \{S_i \in \mathcal{F} \mid x \in S_i\}$

HS/SC Dualität

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



$U \setminus \mathcal{F}$	1	2	3	4
{1, 2}				
{1, 4}				
{2, 3, 4}				
1				
2				
3				
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 $F_x := \{S_i \in \mathcal{F} \mid x \in S_i\}$

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}
$\{F_1, F_2\}$			
$= F_1$			
$= F_2$			
F_3			
F_4			

SET COVER ist NP-vollständig

Theorem

HITTING SET \leq_m^p SET COVER.

Beweis (Skizze)

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

mit $F_x := \{\underline{S_i \in \mathcal{F} \mid x \in S_i}\}$

Korrektheit:

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

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

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

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

$$\underline{\underline{x \in X}}$$

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

df. SC

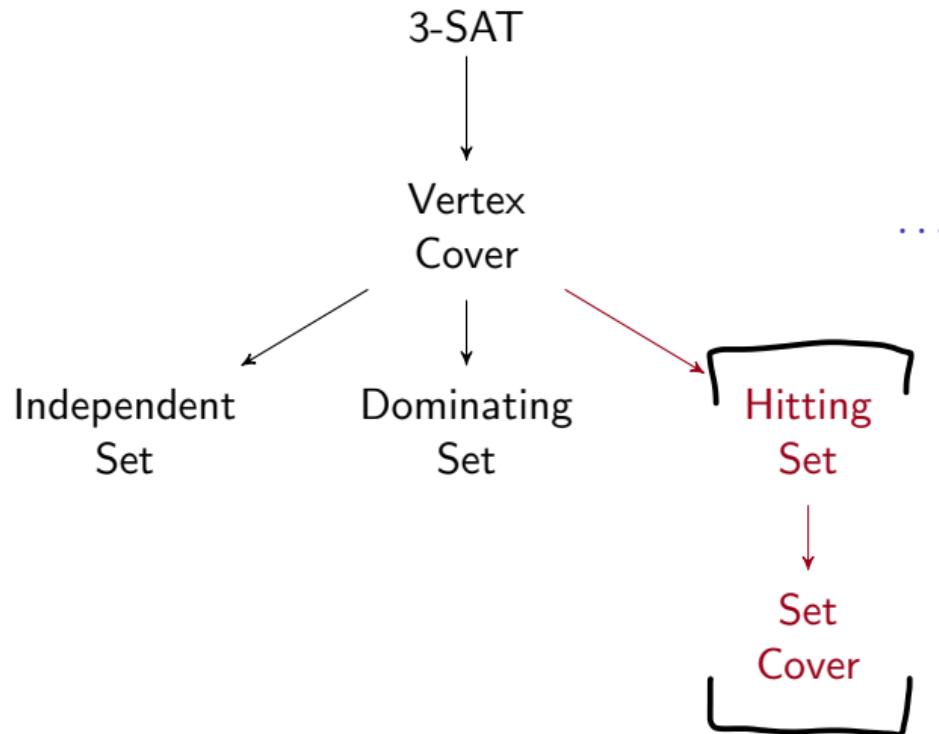
HS/SC Dualität

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

\downarrow poly Zeit

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

Netzwerk polynomieller Reduktionen II



SUBSET SUM

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

Subset Sum

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

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

Beispiel

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

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 $B = 29$.
 $\leadsto X = \{4, 4, 21\}$.

SUBSET SUM ist NP-vollständig

Theorem

3-SAT \leq_m^p SUBSET SUM.

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

Beispiel

$$c_1: x_1 \vee x_2 \vee \overline{x_3}$$

$$c_2: \overline{x_1} \vee x_2 \vee x_3$$

$$c_3: \overline{x_1} \vee \overline{x_2} \vee \overline{x_3}$$

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 $\overline{x_i} \in c_j$, und sonst 0.

Beispiel

$$c_1 : x_1 \vee x_2 \vee \overline{x_3}$$

$$c_2 : \overline{x_1} \vee x_2 \vee x_3$$

$$c_3 : \overline{x_1} \vee \overline{x_2} \vee \overline{x_3}$$

x_1	x_2	x_3	c_1	c_2	c_3
1	0	0	1	0	0
1	0	0	0	1	1

SUBSET SUM ist NP-vollständig

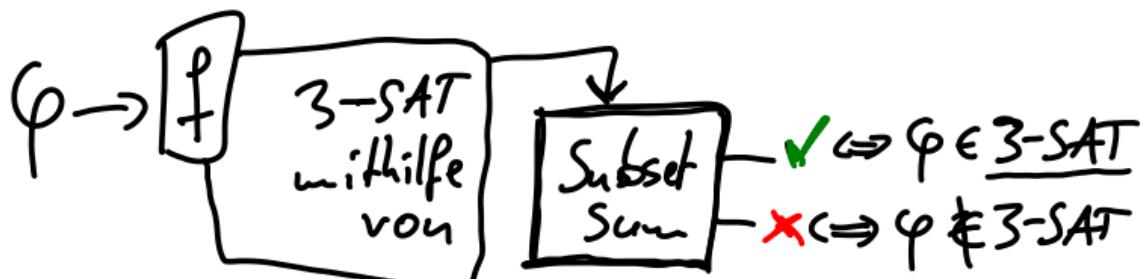
Theorem

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.



Beispiel

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

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

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

$$\begin{array}{ccccccc} x_1 & x_2 & x_3 & c_1 & c_2 & c_3 \\ \hline -y_1 & 1 & 0 & 0 & 0 & 0 \\ -z_1 & 1 & 0 & 0 & 1 & 1 \\ -y_2 & 0 & 1 & 0 & 1 & 0 \\ x_2 | z_2 & 0 & 1 & 0 & 0 & 1 \\ x_1 | y_3 & 0 & 0 & 1 & 0 & 1 \\ z_3 & 0 & 0 & 1 & 1 & 0 \end{array}$$

B: $\begin{array}{ccc|cc} 1 & 1 & 1 & 1 & 1 \\ \hline & & & 1 & 1 \\ & & & & 1 \end{array}$

2

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

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

B

3

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

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	1	0
g_3 :	0	0	0	0	0	1
h_3 :	0	0	0	0	0	1

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 $B := \underbrace{1 \dots 1}_{n} \underbrace{3 \dots 3}_{m}$.

Beispiel

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

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

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

$$\begin{array}{ccccccc} x_1 & x_2 & x_3 & c_1 & c_2 & c_3 \\ \hline y_1: & 1 & 0 & 0 & 1 & 0 & 0 \\ \rightarrow z_1: & 1 & 0 & 0 & 0 & 1 & 1 \\ y_2: & 0 & 1 & 0 & 1 & 1 & 0 \\ \rightarrow z_2: & 0 & 1 & 0 & 0 & 0 & 1 \\ y_3: & 0 & 0 & 1 & 0 & 1 & 0 \\ z_3: & 0 & 0 & 1 & 1 & 0 & 1 \\ g_1: & 0 & 0 & 0 & 1 & 0 & 0 \\ h_1: & 0 & 0 & 0 & 1 & 0 & 0 \\ g_2: & 0 & 0 & 0 & 0 & 1 & 0 \\ h_2: & 0 & 0 & 0 & 0 & 0 & 1 \\ g_3: & 0 & 0 & 0 & 0 & 0 & 1 \\ h_3: & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

$$B : \underbrace{1 \quad 1 \quad 1}_{(1)} \quad \underbrace{3 \quad 3 \quad 3}_{(0)}$$

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

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	1	0
g_3	0	0	0	0	0	1
h_3	0	0	0	0	0	1
B	1	1	1	<u>3</u>	<u>3</u>	<u>3</u>

1 1 1 3 3 3

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

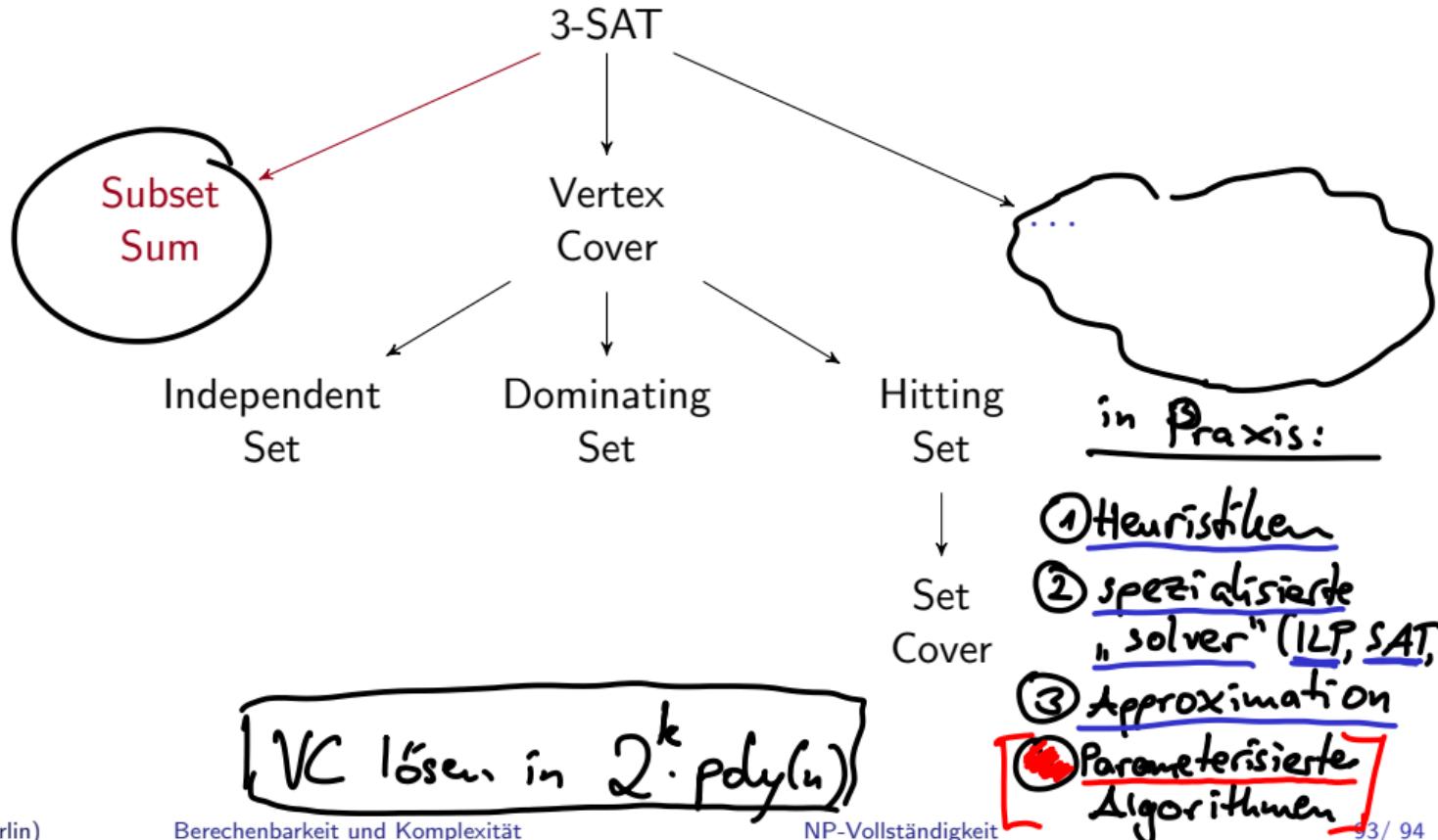
$$c_1: x_1 \vee x_2 \vee \bar{x}_3$$

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

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

	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
	2	2	2	1	1	1

Netzwerk polynomieller Reduktionen III



Gliederung

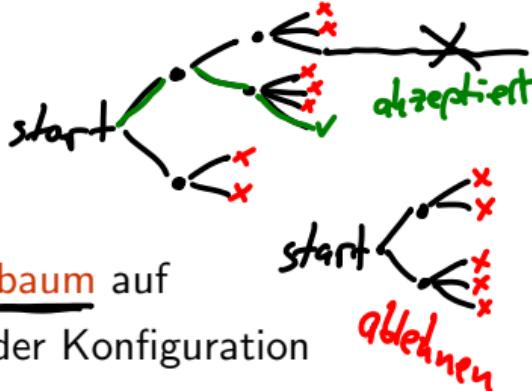
1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem
7. Aufzählbarkeit & (Semi-)Entscheidbarkeit
8. Reduzierbarkeit
9. Satz von Rice
10. Das Postsche Korrespondenzproblem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

Co-Nichtdeterministische Turing-Maschinen

Definition (Nichtdeterministische Turing-Maschine)

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

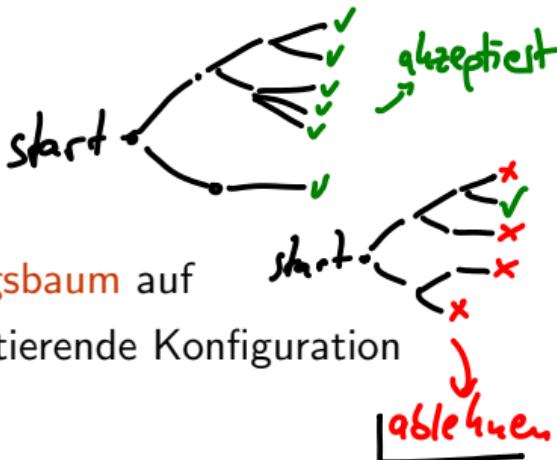
- ▶ “Folgekonfiguration”-Relation \vdash_M^1 von M spannt Berechnungsbaum auf
- ▶ NTM akzeptiert \Leftrightarrow es gibt Berechnungspfad zu akzeptierender Konfiguration



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



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

$\text{time}_{\text{co}N}$ und $\text{coNTIME}(f(n))$ analog zu time_N und $\text{NTIME}(f(n))$

Länge des längsten
Berechnungspfades

Probleme die von coNTM
in $f(n)$ Zeit entscheiden
werden können

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)

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

“co-nichtdeterministisch, in Polynomzeit”

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 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. $\underline{\text{time}_M(n)} \in O(n^c)$) existieren, sodass für jedes $x \in \Sigma^*$ gilt

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

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

beziehungsweise

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

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

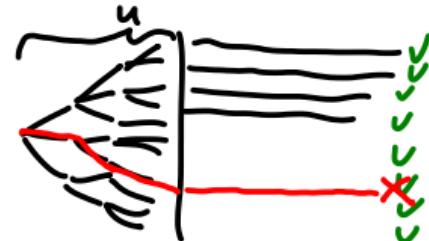
$$x \in L \Leftrightarrow \exists_{u \in \Sigma^{p(|x|)}} \langle x, u \rangle \in T(M).$$

beziehungsweise

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

beziehungsweise

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

*zertifikat
⇒ „Gegenbeispiel“*

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

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

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 $L \subseteq \Sigma^*$.

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

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

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.

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

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

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

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



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 $H \notin \text{coNP}$)

Die Komplexitätsklasse coNP II

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

Theorem

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

Bemerkungen:

- coNP ist nicht das Komplement von NP (z.B. $H \notin \text{NP}$ und $H \notin \text{coNP}$)
- $P \subseteq \underline{\text{NP}} \cap \underline{\text{coNP}}$ (da $\underline{L} \in P \Leftrightarrow \bar{L} \in P$)

$$\begin{aligned} P = P \cap \underline{\text{NP}} &= P \cap \underline{\text{coNP}} = \{L \mid \bar{L} \in \text{NP} \wedge L \in P\} \\ &= \{L \mid \underline{\bar{L} \in \text{NP}} \wedge \underline{L \in P}\} \\ &= \{L \mid \bar{L} \in P\} = \{L \mid L \in P\} = P \end{aligned}$$

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 $\Leftrightarrow \underbrace{\forall_{L \in \text{coNP}} L \leq_m^P A}_{A \text{ coNP-schwer}} \text{ und } \underline{A \in \text{coNP}}$

Die Komplexitätsklasse coNP II

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

Theorem

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

Bemerkungen:

- ▶ coNP ist **nicht** das Komplement von NP (z.B. $H \notin \text{NP}$ und $H \notin \text{coNP}$)
- ▶ $P \subseteq NP \cap \text{coNP}$ (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 $A \in \text{coNP}$
- ▶ $\overline{\text{SAT}} := \{\varphi \mid \varphi \text{ ist unerfüllbar}\} \in \underline{\text{coNP}}$ (sogar **coNP-vollständig**)

Die Komplexitätsklasse coNP II

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

Theorem

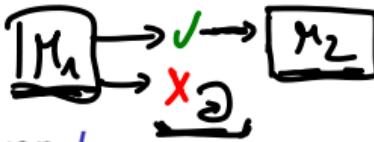
$$\text{coNP} = \{L \subseteq \Sigma^* \mid \bar{L} \in \text{NP}\}.$$



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 $\underline{L \in P \Leftrightarrow \bar{L} \in P}$)
- ▶ 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)
- ▶ $(P = NP) \Rightarrow (NP = \text{coNP} = P)$
für alle $\underline{L \in \text{coNP}}$ gilt: $\underline{\bar{L} \in \text{NP}} \Rightarrow \underline{\bar{L} \in P} \Rightarrow \underline{L \in P}$ und somit $\underline{\underline{L \in NP}}$

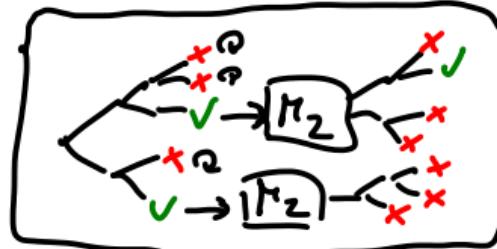
Die Komplexitätsklasse coNP II



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:** $(\text{NP} = \text{coNP}) \Rightarrow (P = \text{NP})?$
- ▶ **Vorsicht:** können nicht ohne Weiteres NTM & coNTM „zusammenstecken“

Ein coNP-vollständiges Problem

$$(x \vee \overline{x})$$

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?

F ist Tautologie
 \Leftrightarrow
 $\neg F$ unerfüllbar

$F \in \text{TAUT}$
 $\neg F \notin \text{SAT}$

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.

(1) $TAUT \in \text{coNP}$ ←
(2) $TAUT$ coNP-schwer $L \in \text{coNP} \leq^P TAUT$

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)

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. $\forall_{L \in \text{coNP}} L \leq_m^p \text{TAUT}$):
Da $\underline{\text{L}} \in \text{NP}$, gilt $\underline{\text{L}} \leq_m^p \text{SAT}$ vermöge einer Polynomzeitreduktion $f: x \mapsto F_x$. Also

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

$$\underline{x \in L} \Leftrightarrow \underline{x \notin \bar{L}} \Leftrightarrow \underline{F_x \notin SAT} \Leftrightarrow \underline{\neg F_x \in TAUT}.$$

Reduktions-eigen-schaft v. f

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. $\forall_{L \in \text{coNP}} L \leq_m^p$ TAUT):

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

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**



Quelle: http://commons.wikimedia.org/wiki/File:13_by_13_game_finished.jpg

Platzkomplexität: PSPACE und LOGSPACE

Bisher: Klassifikation der Berechnungsschwere von Problemen anhand von Zeitbedarf

Platzkomplexität: PSPACE und LOGSPACE

Bisher: Klassifikation der Berechnungsschwere von Problemen anhand von Zeitbedarf

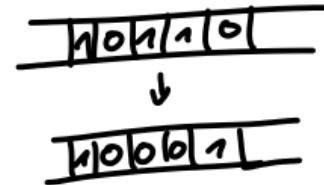
Jetzt: Zweites wichtiges Kriterium – Speicherplatzbedarf

Platzkomplexität: PSPACE und LOGSPACE

Bisher: Klassifikation der Berechnungsschwere von Problemen anhand von Zeitbedarf

Jetzt: Zweites wichtiges Kriterium – Speicherplatzbedarf

Zentraler Unterschied: Speicherplatz ist wiederverwendbar!



Platzkomplexität: PSPACE und LOGSPACE

Bisher: Klassifikation der Berechnungsschwere von Problemen anhand von Zeitbedarf

Jetzt: Zweites wichtiges Kriterium – Speicherplatzbedarf

Zentraler Unterschied: Speicherplatz ist wiederverwendbar!

Definition (PSPACE, LOGSPACE (informell))

Eine Sprache $L \subseteq \Sigma^*$ liegt in...

... PSPACE gdw. eine DTM M existiert mit $T(M) = L$ und M modifiziert höchstens
polynomiell viele Speicherzellen.

Platzkomplexität: PSPACE und LOGSPACE

Bisher: Klassifikation der Berechnungsschwere von Problemen anhand von Zeitbedarf

Jetzt: Zweites wichtiges Kriterium – Speicherplatzbedarf

Zentraler Unterschied: Speicherplatz ist wiederverwendbar!

Definition (PSPACE, LOGSPACE (informell))

Eine Sprache $L \subseteq \Sigma^*$ liegt in...

... **PSPACE** gdw. eine DTM M existiert mit $T(M) = L$ und M modifiziert höchstens **polynomiell** viele Speicherzellen.

... **LOGSPACE** gdw. eine DTM M existiert mit $T(M) = L$ und M modifiziert höchstens **logarithmisch** viele Speicherzellen (Eingabe darf nur gelesen werden).

$$\begin{array}{r} 17 \\ + 201 \\ \hline \end{array}$$

$$n \quad \log n$$

Eingabeband 11111
Arbeitsband 1111
 $\Theta(\log n)$

Platzkomplexität: PSPACE und LOGSPACE

Bisher: Klassifikation der Berechnungsschwere von Problemen anhand von Zeitbedarf

Jetzt: Zweites wichtiges Kriterium – Speicherplatzbedarf

Zentraler Unterschied: Speicherplatz ist wiederverwendbar!

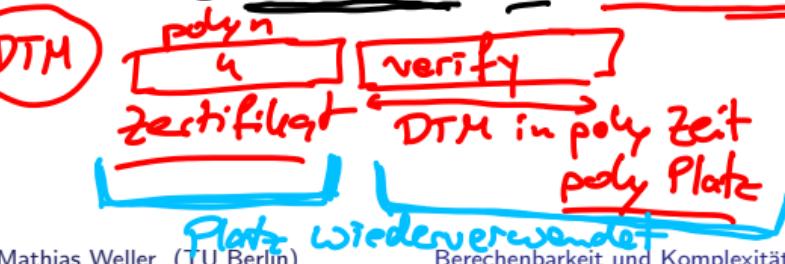
Definition (PSPACE, LOGSPACE (informell))

Eine Sprache $L \subseteq \Sigma^*$ liegt in...

... **PSPACE** gdw. eine DTM M existiert mit $T(M) = L$ und M modifiziert höchstens **polynomiell** viele Speicherzellen.

... **LOGSPACE** gdw. eine DTM M existiert mit $T(M) = L$ und M modifiziert höchstens **logarithmisch** viele Speicherzellen (Eingabe darf nur gelesen werden).

Mitteilung: LOGSPACE \subseteq P \subseteq NP \subseteq PSPACE.



κ benutzt $O(1)$ Platz
 $\frac{2^{O(\kappa)}}{h = \log(n)} \rightarrow 2^{O(\log n)} = \underline{\text{poly}(n)}$
PSPACE

Platzkomplexität: PSPACE und LOGSPACE

Bisher: Klassifikation der Berechnungsschwere von Problemen anhand von Zeitbedarf

Jetzt: Zweites wichtiges Kriterium – Speicherplatzbedarf

Zentraler Unterschied: Speicherplatz ist wiederverwendbar!

Definition (PSPACE, LOGSPACE (informell))

Eine Sprache $L \subseteq \Sigma^*$ liegt in...

... **PSPACE** gdw. eine DTM M existiert mit $T(M) = L$ und M modifiziert höchstens **polynomiell** viele Speicherzellen.

... **LOGSPACE** gdw. eine DTM M existiert mit $T(M) = L$ und M modifiziert höchstens **logarithmisch** viele Speicherzellen (Eingabe darf nur gelesen werden).

Mitteilung: $\text{LOGSPACE} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE}$.

Definition

- Eine Sprache A heißt **PSPACE-schwer**, falls $\forall_{L \in \text{PSPACE}} L \leq_m^p A$.
- A heißt **PSPACE-vollständig**, wenn A **PSPACE-schwer** ist und $A \in \text{PSPACE}$.

Ein PSPACE-vollständiges Problem

Erfüllbarkeit einer **quantifizierten** aussagenlogischen Formel.

Definition

Eine **quantifizierte** aussagenlogische Formel (in Pränexform) hat die Gestalt

$Q_1 x_1 Q_2 x_2 \dots Q_n x_n$ $F(x_1, \dots, x_n)$, wobei $Q_i \in \{\exists, \forall\}$ und F eine aussagenlogische Formel mit ungebundenen ("freien") Variablen x_i ist.

Ein PSPACE-vollständiges Problem

Erfüllbarkeit einer **quantifizierten** aussagenlogischen Formel.

Definition

Eine quantifizierte aussagenlogische Formel (in Pränexform) hat die Gestalt $Q_1 x_1 Q_2 x_2 \dots Q_n x_n F(x_1, \dots, x_n)$, wobei $Q_i \in \{\exists, \forall\}$ und F eine aussagenlogische Formel mit ungebundenen ("freien") Variablen x_i ist.

TQBF (True Quantified Boolean Formula) ist die Sprache aller wahren quantifizierten aussagenlogischen Formeln.

Ein PSPACE-vollständiges Problem

Erfüllbarkeit einer **quantifizierten** aussagenlogischen Formel.

Definition

Eine **quantifizierte** aussagenlogische Formel (in Pränexform) hat die Gestalt

$Q_1 x_1 Q_2 x_2 \dots Q_n x_n F(x_1, \dots, x_n)$, wobei $Q_i \in \{\exists, \forall\}$ und F eine aussagenlogische Formel mit ungebundenen ("freien") Variablen x_i ist.

TQBF (True Quantified Boolean Formula) ist die Sprache aller wahren quantifizierten aussagenlogischen Formeln.

Beispiele:

$$\underline{\exists x \forall y} (\underline{x \wedge y}) \vee (\underline{\bar{x} \wedge \bar{y}})$$

$$x=0 \rightarrow \exists y (\cancel{(0 \wedge y)} \vee \cancel{(1 \wedge \bar{y})}) \rightarrow \exists y \bar{y}$$

~~0~~ ~~y~~ ~~1~~ ~~¬y~~

wahr

$$x=1 \rightarrow \exists y (\cancel{(1 \wedge y)} \vee \cancel{(0 \wedge \bar{y})}) \rightarrow \exists y y$$

~~y~~ ~~0~~

wahr

Ein PSPACE-vollständiges Problem

Erfüllbarkeit einer **quantifizierten** aussagenlogischen Formel.

Definition

Eine **quantifizierte** aussagenlogische Formel (in Pränexform) hat die Gestalt $Q_1 x_1 Q_2 x_2 \dots Q_n x_n F(x_1, \dots, x_n)$, wobei $Q_i \in \{\exists, \forall\}$ und F eine aussagenlogische Formel mit ungebundenen ("freien") Variablen x_i ist.

TQBF (True Quantified Boolean Formula) ist die Sprache aller wahren quantifizierten aussagenlogischen Formeln.

Beispiele:

$$\forall_x \exists_y (x \wedge y) \vee (\bar{x} \wedge \bar{y}) \in \underline{\text{TQBF}}$$

Ein PSPACE-vollständiges Problem

Erfüllbarkeit einer **quantifizierten** aussagenlogischen Formel.

Definition

Eine **quantifizierte** aussagenlogische Formel (in Pränexform) hat die Gestalt

$Q_1 x_1 Q_2 x_2 \dots Q_n x_n F(x_1, \dots, x_n)$, wobei $Q_i \in \{\exists, \forall\}$ und F eine aussagenlogische Formel mit ungebundenen ("freien") Variablen x_i ist.

TQBF (True Quantified Boolean Formula) ist die Sprache aller wahren quantifizierten aussagenlogischen Formeln.

Beispiele:

$$\forall x \exists y (x \wedge y) \vee (\bar{x} \wedge \bar{y}) \in \text{TQBF}$$

$$\boxed{\forall x \forall y (x \wedge y) \vee (\bar{x} \wedge \bar{y})}$$

$$\boxed{(1 \wedge 0) \vee (0 \wedge 1)}$$

$$\underline{x=1, y=0}$$

Ein PSPACE-vollständiges Problem

Erfüllbarkeit einer **quantifizierten** aussagenlogischen Formel.

Definition

Eine **quantifizierte** aussagenlogische Formel (in Pränexform) hat die Gestalt

$Q_1 x_1 Q_2 x_2 \dots Q_n x_n F(x_1, \dots, x_n)$, wobei $Q_i \in \{\exists, \forall\}$ und F eine aussagenlogische Formel mit ungebundenen ("freien") Variablen x_i ist.

TQBF (True Quantified Boolean Formula) ist die Sprache aller wahren quantifizierten aussagenlogischen Formeln.

Beispiele:

$$\forall x \exists y (x \wedge y) \vee (\bar{x} \wedge \bar{y}) \in \text{TQBF}$$

$$\forall x \forall y (x \wedge y) \vee (\bar{x} \wedge \bar{y}) \notin \text{TQBF}$$

Ein PSPACE-vollständiges Problem

Erfüllbarkeit einer **quantifizierten** aussagenlogischen Formel.

$NP \subseteq PSPACE$
NP
 \cap
 $\text{co}NP$

Definition

Eine **quantifizierte** aussagenlogische Formel (in Pränexform) hat die Gestalt

$Q_1 x_1 Q_2 x_2 \dots Q_n x_n F(x_1, \dots, x_n)$, wobei $Q_i \in \{\exists, \forall\}$ und F eine aussagenlogische Formel mit ungebundenen ("freien") Variablen x_i ist.

TQBF (True Quantified Boolean Formula) ist die Sprache aller wahren quantifizierten aussagenlogischen Formeln.

Beispiele:

$$\forall_x \exists_y (x \wedge y) \vee (\bar{x} \wedge \bar{y}) \in \text{TQBF}$$

$$\forall_x \forall_y (x \wedge y) \vee (\bar{x} \wedge \bar{y}) \notin \text{TQBF}$$

Spezialfälle:

$$\underline{\text{SAT}} \rightsquigarrow \exists_{x_1} \dots \exists_{x_n} F(x_1, \dots, x_n) \in \text{TQBF}$$

$$\underline{\text{TAUT}} \rightsquigarrow \forall_{x_1} \dots \forall_{x_n} F(x_1, \dots, x_n) \in \text{TQBF}$$

Ein PSPACE-vollständiges Problem

Erfüllbarkeit einer **quantifizierten** aussagenlogischen Formel.

Definition

Eine **quantifizierte** aussagenlogische Formel (in Pränexform) hat die Gestalt

$Q_1 x_1 Q_2 x_2 \dots Q_n x_n F(x_1, \dots, x_n)$, wobei $Q_i \in \{\exists, \forall\}$ und F eine aussagenlogische Formel mit ungebundenen ("freien") Variablen x_i ist.

TQBF (True Quantified Boolean Formula) ist die Sprache aller wahren quantifizierten aussagenlogischen Formeln.

$$\overline{\exists} \overline{x}_1 \overline{\forall} \overline{x}_2 \overline{\exists} \overline{x}_3 \overline{\forall} \overline{x}_4 \varphi(x_1, x_2, x_3, x_4)$$

Beispiele:

$$\boxed{\forall x \exists y (x \wedge y) \vee (\bar{x} \wedge \bar{y})} \in \text{TQBF}$$
$$\forall x \forall y (x \wedge y) \vee (\bar{x} \wedge \bar{y}) \notin \text{TQBF}$$

Spezialfälle:

$$\text{SAT} \rightsquigarrow \exists_{x_1} \dots \exists_{x_n} F(x_1, \dots, x_n) \in \text{TQBF}$$

$$\text{TAUT} \rightsquigarrow \forall_{x_1} \dots \forall_{x_n} F(x_1, \dots, x_n) \in \text{TQBF}$$

Beachte: Ähnlichkeit zu "Spielproblemen" (kann Spielerin X gewinnen?)

TQBF ist PSPACE-vollständig

Theorem

TQBF ist PSPACE-vollständig.

TQBF ist PSPACE-vollständig

Theorem

TQBF ist PSPACE-vollständig.

Beweis (Skizze)

1. TQBF ∈ PSPACE:

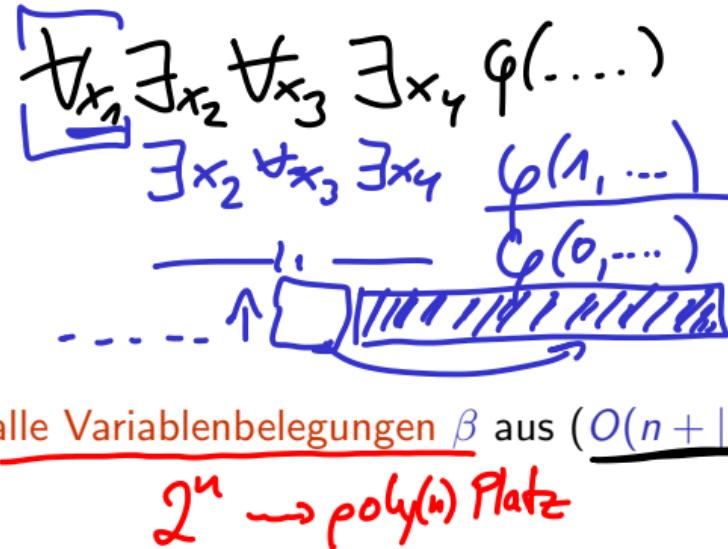
Werte die Formel $F(x_1, \dots, x_n)$ rekursiv für alle Variablenbelegungen β aus ($O(n + |F|)$ Platz)

TQBF(i):

IF $i \leq n$ THEN

$\beta(x_i) := 0; \quad a := \text{TQBF}(i+1);$

global



TQBF ist PSPACE-vollständig

Theorem

TQBF ist PSPACE-vollständig.



Beweis (Skizze)

1. $\text{TQBF} \in \text{PSPACE}$:

Werte die Formel $F(x_1, \dots, x_n)$ rekursiv für alle Variablenbelegungen β aus ($O(n + |F|)$ Platz)

$\text{TQBF}(i)$:

IF $i \leq n$ **THEN**

$\beta(x_i) := 0; \quad a := \text{TQBF}(i+1);$

$\beta(x_i) := 1; \quad b := \text{TQBF}(i+1);$

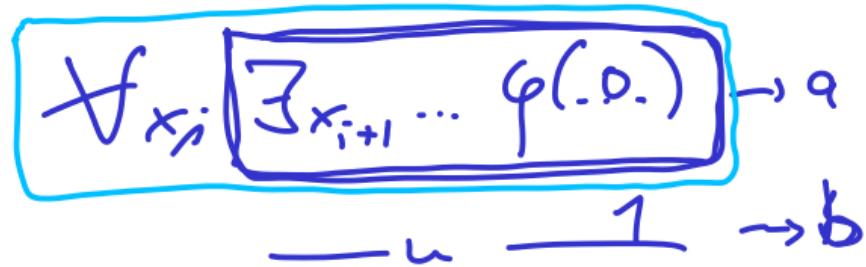
IF $Q_i = \exists$ **THEN**

RETURN $a \vee b;$

TQBF ist PSPACE-vollständig

Theorem

TQBF ist PSPACE-vollständig.



Beweis (Skizze)

1. $\text{TQBF} \in \text{PSPACE}$:

Werte die Formel $F(x_1, \dots, x_n)$ rekursiv für alle Variablenbelegungen β aus ($O(n + |F|)$ Platz)

$\text{TQBF}(i)$:

IF $i \leq n$ **THEN**

$\beta(x_i) := 0$; $a := \text{TQBF}(i + 1)$;

$\beta(x_i) := 1$; $b := \text{TQBF}(i + 1)$;

IF $Q_i = \exists$ **THEN**

RETURN $a \vee b$;

ELSE RETURN $a \wedge b$;

TQBF ist PSPACE-vollständig

Theorem

TQBF ist PSPACE-vollständig.

Beweis (Skizze)

1. TQBF \in PSPACE:

Werte die Formel $F(x_1, \dots, x_n)$ rekursiv für alle Variablenbelegungen β aus $(O(n + |F|))$ Platz

TQBF(i):

IF $i \leq n$ THEN

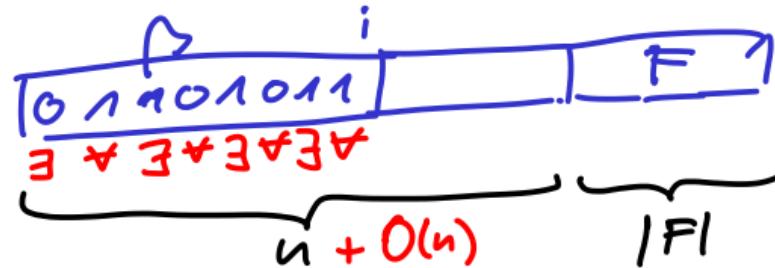
$\beta(x_i) := 0$; $a := \text{TQBF}(i+1)$;
 $\beta(x_i) := 1$; $b := \text{TQBF}(i+1)$;

IF $Q_i = \exists$ THEN

RETURN $a \vee b$;

ELSE RETURN $a \wedge b$;

ELSE RETURN $\beta(F)$



$$\text{platz}(i) \in O(\log n)$$

$$\text{poly}(n + |F|)$$

$$\text{platz}(a) = 1$$

$$\text{platz}(b) = 1$$

$$\text{platz}(\beta) = O(n)$$

$$\text{platz}(F) = O(|F|)$$

$$\sum: \text{poly}(n + |F|)$$

TQBF ist PSPACE-vollständig

Theorem

TQBF ist PSPACE-vollständig.

Beweis (Skizze)

1. TQBF \in PSPACE:

Werte die Formel $F(x_1, \dots, x_n)$ rekursiv für alle Variablenbelegungen β aus ($O(n + |F|)$ Platz)

TQBF(i):

IF $i \leq n$ **THEN**

$\beta(x_i) := 0$; $a := \text{TQBF}(i+1)$;

$\beta(x_i) := 1$; $b := \text{TQBF}(i+1)$;

IF $Q_i = \exists$ **THEN**

RETURN $a \vee b$;

ELSE RETURN $a \wedge b$;

ELSE RETURN $\beta(F)$

2. TQBF ist PSPACE-schwer: Ähnlich wie im Satz von Cook/Levin.



PSPACE und Spiele

PSPACE-vollständige Probleme haben oft Bezug zur Frage nach der Existenz von Gewinnstrategien für Spiele mit zwei Spielerinnen (z.B. Schach oder Go).

PSPACE und Spiele

PSPACE-vollständige Probleme haben oft Bezug zur Frage nach der Existenz von Gewinnstrategien für Spiele mit zwei Spielerinnen (z.B. Schach oder Go).

Szenario:

- ▶ „ \forall -Spielerin“ wählt Belegung für \forall -quantifizierte Variablen.

PSPACE und Spiele

PSPACE-vollständige Probleme haben oft Bezug zur Frage nach der Existenz von Gewinnstrategien für Spiele mit zwei Spielerinnen (z.B. Schach oder Go).

Szenario:

- ▶ „ \forall -Spielerin“ wählt Belegung für \forall -quantifizierte Variablen.
- ▶ „ \exists -Spielerin“ wählt Belegung für \exists -quantifizierte Variablen.

PSPACE und Spiele

PSPACE-vollständige Probleme haben oft Bezug zur Frage nach der Existenz von Gewinnstrategien für Spiele mit zwei Spielerinnen (z.B. Schach oder Go).

Szenario:

- ▶ „ \forall -Spielerin“ wählt Belegung für \forall -quantifizierte Variablen.
- ▶ „ \exists -Spielerin“ wählt Belegung für \exists -quantifizierte Variablen.
- ▶ \exists/\forall -Spielerin gewinnt, wenn die Formel F wahr/falsch wird.

PSPACE und Spiele

PSPACE-vollständige Probleme haben oft Bezug zur Frage nach der Existenz von Gewinnstrategien für Spiele mit zwei Spielerinnen (z.B. Schach oder Go).

Szenario:

- ▶ „ \forall -Spielerin“ wählt Belegung für \forall -quantifizierte Variablen.
- ▶ „ \exists -Spielerin“ wählt Belegung für \exists -quantifizierte Variablen.
- ▶ \exists/\forall -Spielerin gewinnt, wenn die Formel F wahr/falsch wird.

Frage: Existiert eine Gewinnstrategie für \exists -Spielerin?

„existiert ein Zug so dass egal was die Andere tut ich einen
„Zug habe des Gewinnt“ $(\exists x_1 \forall x_2 \exists x_3 \varphi(x_1, x_2, x_3))$

Ein Geographiespiel

Eingabe: Menge von Hauptstadtnamen. , längster Buchstabe

Spielregeln:

1. zwei Spielerinnen nennen abwechselnd eine Hauptstadt
2. jede genannte Hauptstadt muss mit dem letzten Buchstaben der Zuvorgenannten beginnen
3. wer als erstes keine Hauptstadt mehr nennen kann, verliert
4. keine Mehrfachnennungen

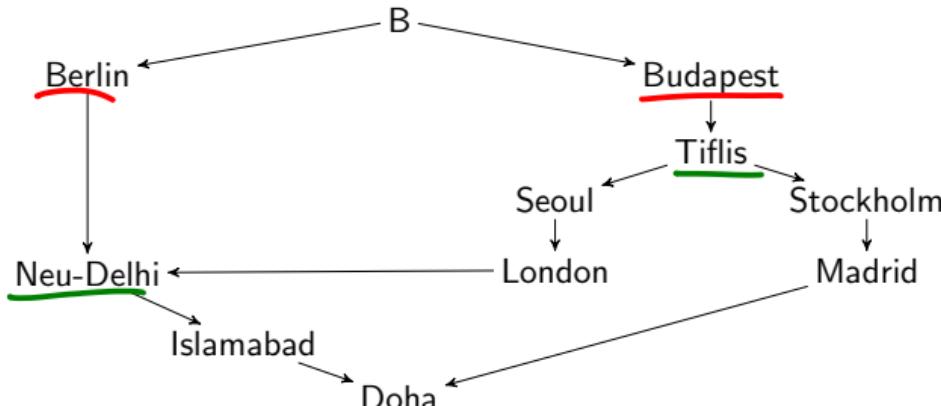
Ein Geographiespiel

Eingabe: Menge von Hauptstadtnamen.

Spielregeln:

1. zwei Spielerinnen nennen abwechselnd eine Hauptstadt
2. jede genannte Hauptstadt muss mit dem letzten Buchstaben der Zuvorgenannten beginnen
3. wer als erstes keine Hauptstadt mehr nennen kann, verliert
4. keine Mehrfachnennungen

Beispiel

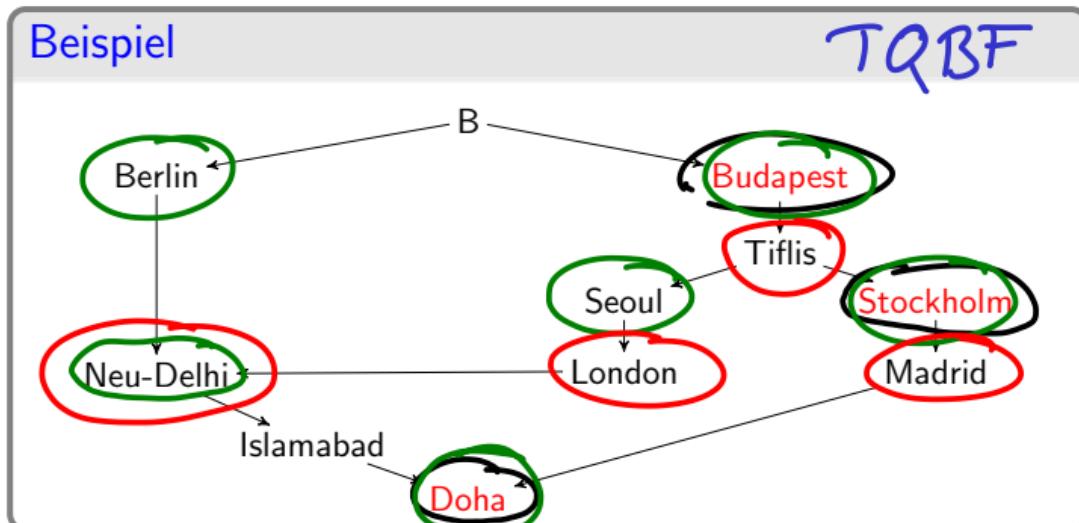


Ein Geographiespiel

Eingabe: Menge von Hauptstadtnamen.

Spielregeln:

1. zwei Spielerinnen nennen abwechselnd eine Hauptstadt
2. jede genannte Hauptstadt muss mit dem letzten Buchstaben der Zuvorgenannten beginnen
3. wer als erstes keine Hauptstadt mehr nennen kann, verliert
4. keine Mehrfachnennungen



Generalisiertes Geographiespiel

Eingabe: Ein gerichteter Graph $\underline{G} = (\underline{V}, \underline{E})$ und ein Startknoten $\underline{v} \in \underline{V}$.

Generalisiertes Geographiespiel

Eingabe: Ein gerichteter Graph $G = (V, E)$ und ein Startknoten $v \in V$.

Spielregeln:

1. Spielerinnen wählen abwechselnd den “nächsten Knoten” unter den Nachfolgern des aktuellen Knotens
2. wer keinen Nachbarknoten mehr auswählen kann verliert

Generalisiertes Geographiespiel

Eingabe: Ein gerichteter Graph $G = (V, E)$ und ein Startknoten $v \in V$.

Spielregeln:

1. Spielerinnen wählen abwechselnd den “nächsten Knoten” unter den Nachfolgern des aktuellen Knotens
2. wer keinen Nachbarknoten mehr auswählen kann verliert

Spielerin 1 hat “Gewinnstrategie” \rightsquigarrow Ähnlichkeit zu TQBF

Generalisiertes Geographiespiel

Eingabe: Ein gerichteter Graph $G = (V, E)$ und ein Startknoten $v \in V$.

Spielregeln:

1. Spielerinnen wählen abwechselnd den “nächsten Knoten” unter den Nachfolgern des aktuellen Knotens
2. wer keinen Nachbarknoten mehr auswählen kann verliert

Spielerin 1 hat “Gewinnstrategie” \rightsquigarrow Ähnlichkeit zu TQBF

Generalized Geography

Eingabe: gerichteter Graph $G = (V, E)$ und ein Knoten $v \in V$

Frage: Hat Spielerin 1 eine Gewinnstrategie, die mit einem Nachbarknoten von v startet?

Theorem

GENERALIZED GEOGRAPHY ist PSPACE-vollständig.

Generalisiertes Geographiespiel

Eingabe: Ein gerichteter Graph $G = (V, E)$ und ein Startknoten $v \in V$.

Spielregeln:

1. Spielerinnen wählen abwechselnd den “nächsten Knoten” unter den Nachfolgern des aktuellen Knotens
2. wer keinen Nachbarknoten mehr auswählen kann verliert

Spielerin 1 hat “Gewinnstrategie” \rightsquigarrow Ähnlichkeit zu TQBF

Generalized Geography

Eingabe: gerichteter Graph $G = (V, E)$ und ein Knoten $v \in V$

Frage: Hat Spielerin 1 eine Gewinnstrategie, die mit einem Nachbarknoten von v startet?

Theorem

GENERALIZED GEOGRAPHY ist PSPACE-vollständig.

1. GENERALIZED GEOGRAPHY \in PSPACE: Einfach.

Generalisiertes Geographiespiel

Eingabe: Ein gerichteter Graph $G = (V, E)$ und ein Startknoten $v \in V$.

Spielregeln:

1. Spielerinnen wählen abwechselnd den “nächsten Knoten” unter den Nachfolgern des aktuellen Knotens
2. wer keinen Nachbarknoten mehr auswählen kann verliert

Spielerin 1 hat “Gewinnstrategie” \rightsquigarrow Ähnlichkeit zu TQBF

Generalized Geography

Eingabe: gerichteter Graph $G = (V, E)$ und ein Knoten $v \in V$

Frage: Hat Spielerin 1 eine Gewinnstrategie, die mit einem Nachbarknoten von v startet?

Theorem

GENERALIZED GEOGRAPHY ist PSPACE-vollständig.

1. GENERALIZED GEOGRAPHY \in PSPACE: Einfach.
2. GENERALIZED GEOGRAPHY ist PSPACE-vollständig:
zeige TQBF \leq_m^p GENERALIZED GEOGRAPHY.

Abschließende Bemerkungen zu PSPACE

↳ Lemming

Mitteilung: Für viele Spiele, wie z.B. Schach, Go oder Dame, existieren verallgemeinerte Versionen (auf $n \times n$ -Spielbrettern), die PSPACE-schwer sind.

Abschließende Bemerkungen zu PSPACE

Mitteilung: Für viele Spiele, wie z.B. Schach, Go oder Dame, existieren verallgemeinerte Versionen (auf $n \times n$ -Spielbrettern), die PSPACE-schwer sind.

Typische Anwendungsgebiete wo PSPACE-vollständige Probleme auftreten:

► Robotik

(Roboter spielen gegen ihre Umwelt; „Motion Planning“, „Games against Nature“)



Abschließende Bemerkungen zu PSPACE

Mitteilung: Für viele Spiele, wie z.B. Schach, Go oder Dame, existieren verallgemeinerte Versionen (auf $n \times n$ -Spielbrettern), die PSPACE-schwer sind.

Typische Anwendungsgebiete wo PSPACE-vollständige Probleme auftreten:

- ▶ Robotik
(Roboter spielen gegen ihre Umwelt; „Motion Planning“, „Games against Nature“)
- ▶ Wortproblem für kontextsensitive Sprachen
(d.h. für Typ 1-Sprachen in der Chomsky-Hierarchie)

→ parser bauen
P & PSPACE

Abschließende Bemerkungen zu PSPACE

Mitteilung: Für viele Spiele, wie z.B. Schach, Go oder Dame, existieren verallgemeinerte Versionen (auf $n \times n$ -Spielbrettern), die PSPACE-schwer sind.

Typische Anwendungsgebiete wo PSPACE-vollständige Probleme auftreten:

- ▶ Robotik
(Roboter spielen gegen ihre Umwelt; „Motion Planning“, „Games against Nature“)
- ▶ Wortproblem für kontextsensitive Sprachen
(d.h. für Typ 1-Sprachen in der Chomsky-Hierarchie)

Bemerkungen zu NP und PSPACE:

- ▶ NP: kurze Zertifikate
- ▶ PSPACE: Zertifikat (Gewinnstrategie) kann exponentiell lang sein



Abschließende Bemerkungen zu PSPACE

Mitteilung: Für viele Spiele, wie z.B. Schach, Go oder Dame, existieren verallgemeinerte Versionen (auf $n \times n$ -Spielbrettern), die PSPACE-schwer sind.

Typische Anwendungsgebiete wo PSPACE-vollständige Probleme auftreten:

- ▶ Robotik
(Roboter spielen gegen ihre Umwelt; „Motion Planning“, „Games against Nature“)
- ▶ Wortproblem für kontextsensitive Sprachen
(d.h. für Typ 1-Sprachen in der Chomsky-Hierarchie)

poly (n)
0 1 0 1 1 0 1 0 0

Bemerkungen zu NP und PSPACE:

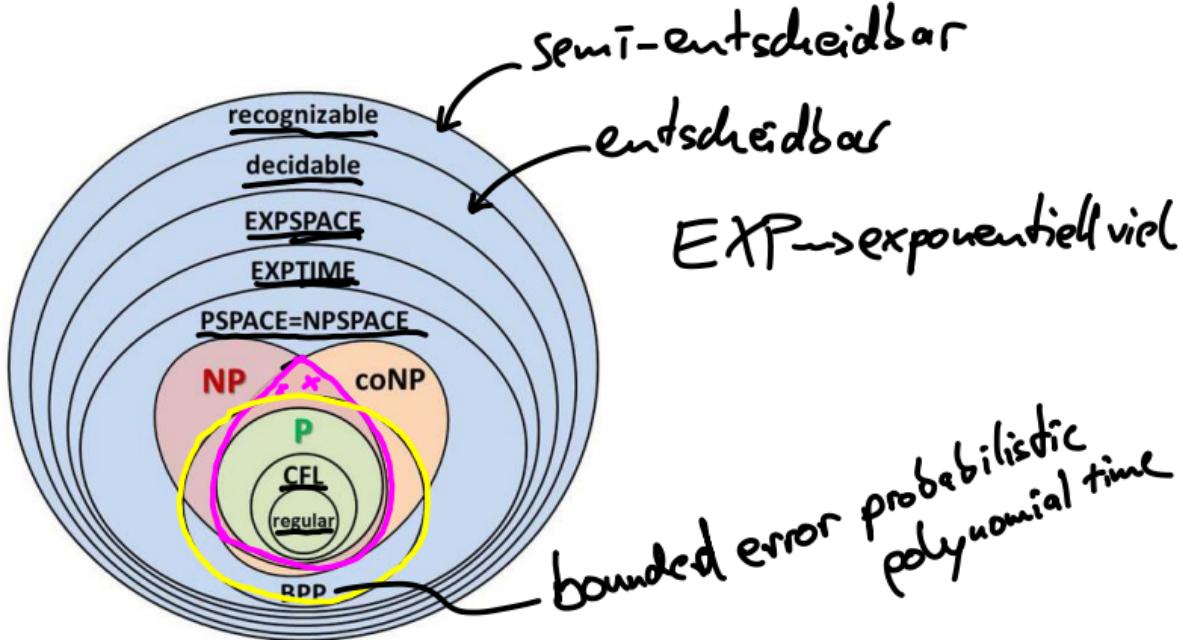
- ▶ NP: kurze Zertifikate
- ▶ PSPACE: Zertifikat (Gewinnstrategie) kann exponentiell lang sein
- ▶ PSPACE = NPSPACE

DTM

NTH

mehr als poly(n) viele
nichtdeterministische
Schritte nicht sinnvoll
→ kann alle nicht-
deterministischen
Entscheidungen in
poly(n) Platz durch-
probieren

Eine komplexe Welt



Quelle: <http://cse.psu.edu/~sxr48/cmpsc464/Complexity-classes-diagram.jpg>

vielleicht Faktorisierung?

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

SAT

I_y: aussagenlogische Formel φ
Q: \exists Belagung β unter der φ zu
1 (wahr) ausgewertet wird

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

Beweis (Skizze: SAT ist NP-schwer)

zu zeigen: $\forall_{L \in \text{NP}} L \leq_m^p \text{SAT}$.

Sei $L \in \text{NP}$. Dann existiert NTM M mit $L = T(M)$ & Polynom p beschränkt Laufzeit von M .

bei Eingabe x macht
 $M \leq_p(x)$ Schritte

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

Beweis (Skizze: SAT ist NP-schwer)

$$\ell, \ell' \in \mathbb{N}$$

zu zeigen: $\forall_{L \in \text{NP}} L \leq_m^p \text{SAT}$.

Sei $L \in \text{NP}$. Dann existiert NTM M mit $L = T(M)$ & Polynom p beschränkt Laufzeit von M .

Sei $M = (\underline{Z}, \underline{\Sigma}, \underline{\Gamma}, \underline{\delta}, \underline{z_1}, \underline{\square}, \underline{E})$ mit $\underline{\Gamma} = \{\underline{a_1} = \square, \dots, \underline{a_\ell}\}$ und $\underline{Z} = \{\underline{z_1}, \dots, \underline{z_k}\}$.

Satz von Cook und Levin



Theorem

SAT ist NP-vollständig.

Beweis (Skizze: SAT ist NP-schwer)

zu zeigen: $\forall_{L \in \text{NP}} L \leq_m^p \text{SAT}$.

Sei $L \in \text{NP}$. Dann existiert NTM M mit $L = T(M)$ & Polynom p beschränkt Laufzeit von M .

Sei $M = (Z, \Sigma, \Gamma, \delta, z_1, \square, E)$ mit $\Gamma = \{a_1 = \square, \dots, a_\ell\}$ und $Z = \{z_1, \dots, z_k\}$.

Annahme: M hält bei Eingabe $x = x_1 x_2 \dots x_n \in \Sigma^n$ nach genau $p(n)$ Schritten.

Wir konstruieren eine Polynomzeitreduktion f , sodass gilt $x \in L \Leftrightarrow f(x) := F_M(x) \in \text{SAT}$.

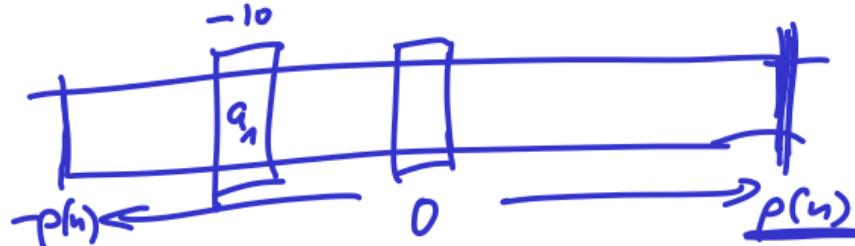
Zu konstruierende Formel $F_M(x)$ besitzt folgende boolesche Variablen:

beschreibt Arbeitsweise von $M(x)$

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.



$$\delta_{t_i-10, q_1} = 1$$

Beweis (Skizze: SAT ist NP-schwer)

zu zeigen: $\forall L \in \text{NP} \ L \leq_m^p \text{SAT}$.

Sei $L \in \text{NP}$. Dann existiert NTM M mit $L = T(M)$ & Polynom p beschränkt Laufzeit von M .

Sei $M = (Z, \Sigma, \Gamma, \delta, z_1, \square, E)$ mit $\Gamma = \{a_1 = \square, \dots, a_\ell\}$ und $Z = \{z_1, \dots, z_k\}$.

Annahme: M hält bei Eingabe $x = x_1 x_2 \dots x_n \in \Sigma^n$ nach genau $p(n)$ Schritten.

Wir konstruieren eine Polynomzeitreduktion f , sodass gilt $x \in L \Leftrightarrow f(x) := F_M(x) \in \text{SAT}$.

Zu konstruierende Formel $F_M(x)$ besitzt folgende boolesche Variablen:

Var.	Indizes	Bedeutung
* <u>$z_{t,j}$</u>	$0 \leq t \leq p(n) \quad 1 \leq j \leq k$	$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten ist M im Zustand z_j
* <u>$p_{t,i}$</u>	$0 \leq t \leq p(n) \quad -p(n) \leq i \leq p(n)$	$p_{t,i} = 1 \Leftrightarrow$ nach t Schritten ist Kopf auf Pos. i
* <u>$b_{t,i,a}$</u>	$0 \leq t \leq p(n) \quad -p(n) \leq i \leq p(n)$ $a \in \Gamma$	$b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten befindet sich auf Bandposition i das Zeichen a

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten: Zustand z_j
 $p_{t,i} = 1 \Leftrightarrow$ nach t Schritten: Kopfpos= i
 $b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten: Band[i]= a



Beweis (Skizze: SAT ist NP-schwer)

zu zeigen: $\forall L \in \text{NP} \ L \leq_m^p \text{SAT}$.

Sei $L \in \text{NP}$. Dann existiert NTM M mit $L = T(M)$ & Polynom p beschränkt Laufzeit von M .

Sei $M = (Z, \Sigma, \Gamma, \delta, z_1, \square, E)$ mit $\Gamma = \{a_1 = \square, \dots, a_\ell\}$ und $Z = \{z_1, \dots, z_k\}$.

Annahme: M hält bei Eingabe $x = x_1 x_2 \dots x_n \in \Sigma^n$ nach genau $p(n)$ Schritten.

Wir konstruieren eine Polynomzeitreduktion f , sodass gilt $x \in L \Leftrightarrow f(x) := \underline{F_M(x)} \in \text{SAT}$.

Zu konstruierende Formel $F_M(x)$ besitzt folgende boolesche Variablen:

Var.	Indizes	Bedeutung
$z_{t,j}$	$0 \leq t \leq p(n) \quad 1 \leq j \leq k$	$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten ist M im Zustand z_j
$p_{t,i}$	$0 \leq t \leq p(n) \quad -p(n) \leq i \leq p(n)$	$p_{t,i} = 1 \Leftrightarrow$ nach t Schritten ist Kopf auf Pos. i
$b_{t,i,a}$	$0 \leq t \leq p(n) \quad -p(n) \leq i \leq p(n)$ $a \in \Gamma$	$b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten befindet sich auf Bandposition i das Zeichen a

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten: Zustand z_j
 $p_{t,i} = 1 \Leftrightarrow$ nach t Schritten: Kopfpos= i
 $b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten: Band[i]= a

Beweis (Skizze: SAT ist NP-schwer)



Satz von Cook und Levin

Theorem

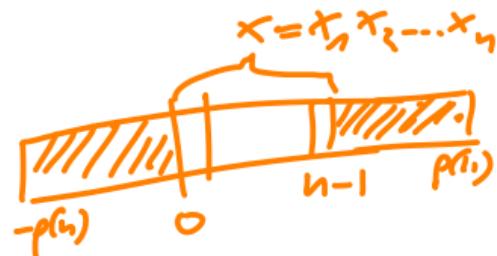
SAT ist NP-vollständig.

$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten: Zustand z_j
 $p_{t,i} = 1 \Leftrightarrow$ nach t Schritten: Kopfpos= i
 $b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten: Band[i]= a

Beweis (Skizze: SAT ist NP-schwer)

$$F_M(x) := A \wedge T_1 \wedge T_2 \wedge F \wedge R$$

Anfang $A := z_{0,1} \wedge p_{0,0} \wedge \bigwedge_{0 \leq i < n} b_{0,i,x_i} \wedge \bigwedge_{-p(n) \leq i < 0} b_{0,i,\square} \wedge \bigwedge_{n \leq i \leq p(n)} b_{0,i,\square}$



Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten: Zustand z_j
 $p_{t,i} = 1 \Leftrightarrow$ nach t Schritten: Kopfpos= i
 $b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten: Band[i]= a

Beweis (Skizze: SAT ist NP-schwer)

$$F_M(x) := A \wedge T_1 \wedge T_2 \wedge F \wedge R$$

Anfang $A := z_{0,1} \wedge p_{0,0} \wedge \bigwedge_{0 \leq i < n} b_{0,i,x_i} \wedge \bigwedge_{-p(n) \leq i < 0} b_{0,i,\square} \wedge \bigwedge_{n \leq i \leq p(n)} b_{0,i,\square}$

Ende $F := \bigvee_{\substack{z_j \in E \\ \underline{\underline{z_j \in E}}}} z_{p(n),j}$

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten: Zustand z_j
 $p_{t,i} = 1 \Leftrightarrow$ nach t Schritten: Kopfpos= i
 $b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten: Band[i]= a

Beweis (Skizze: SAT ist NP-schwer)

$$F_M(x) := A \wedge T_1 \wedge T_2 \wedge F \wedge R$$

Anfang $A := z_{0,1} \wedge p_{0,0} \wedge \bigwedge_{0 \leq i < n} b_{0,i,x_i} \wedge \bigwedge_{-p(n) \leq i < 0} b_{0,i,\square} \wedge \bigwedge_{n \leq i \leq p(n)} b_{0,i,\square}$

Ende $F := \bigvee_{z_j \in E} z_{p(n),j}$

Übergänge $T_1 := \bigwedge \left(\begin{array}{l} \text{wenn } (z_{t,j} \wedge p_{t,i} \wedge b_{t,i,a}) \\ \text{dann } (z_{t+1,j^*} \wedge p_{t+1,i+\gamma} \wedge b_{t+1,i,a^*}) \end{array} \right)$

$\rightarrow 0 \leq t < p(n)$

$\rightarrow -p(n) \leq i \leq p(n)$

$\rightarrow 1 \leq j \leq k$

$\rightarrow a \in \Gamma$

mit $\gamma \in \{-1, 0, 1\}$ (das heißt $L = -1, N = 0, R = 1$)

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten: Zustand z_j
 $p_{t,i} = 1 \Leftrightarrow$ nach t Schritten: Kopfpos= i
 $b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten: Band[i]= a

Beweis (Skizze: SAT ist NP-schwer)

$$F_M(x) := A \wedge T_1 \wedge T_2 \wedge F \wedge R$$

O(1)

Anfang $A := z_{0,1} \wedge p_{0,0} \wedge \bigwedge_{0 \leq i < n} b_{0,i,x_i} \wedge \bigwedge_{-p(n) \leq i < 0} b_{0,i,\square} \wedge \bigwedge_{n \leq i \leq p(n)} b_{0,i,\square}$

O(p(n))

Ende $F := \bigvee_{z_j \in E} z_{p(n),j}$

Übergänge $T_1 := \bigwedge (z_{t,j} \wedge p_{t,i} \wedge b_{t,i,a}) \rightarrow \bigvee (z_{t+1,j^*} \wedge p_{t+1,i+\gamma} \wedge b_{t+1,i,a^*})$

- $0 \leq t < p(n)$
- $-p(n) \leq i \leq p(n)$
- $1 \leq j \leq k$
- $a \in \Gamma$

$(z_{j^*}, a^*, \gamma) \in \delta(z_j, a)$

$$\begin{aligned} & O(p(n) \cdot p(n) \cdot 1 \cdot 1 \cdot 1) \\ & = O(p(n)^2) \end{aligned}$$

mit $\gamma \in \{-1, 0, 1\}$ (das heißt $L = -1, N = 0, R = 1$)

$$T_2 := \bigwedge (\overline{p_{t,i}} \wedge b_{t,i,a}) \rightarrow \underline{b_{t+1,i,a}}$$

- $0 \leq t < p(n)$
- $-p(n) \leq i \leq p(n)$

→ O(p(n)²)

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten: Zustand z_j
 $p_{t,i} = 1 \Leftrightarrow$ nach t Schritten: Kopfpos= i
 $b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten: Band[i]= a

Beweis (Skizze: SAT ist NP-schwer)

$$F_M(x) := A \wedge T_1 \wedge T_2 \wedge F \wedge R$$

Randbedingungen $R := \underline{R_z} \wedge \underline{R_p} \wedge \underline{R_b}$:

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten: Zustand z_j
 $p_{t,i} = 1 \Leftrightarrow$ nach t Schritten: Kopfpos= i
 $b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten: Band[i]= a

Beweis (Skizze: SAT ist NP-schwer)

$$F_M(x) := A \wedge T_1 \wedge T_2 \wedge F \wedge R$$

Randbedingungen $R := R_z \wedge R_p \wedge R_b$:

Zustände $R_z := \bigwedge_{0 \leq t \leq p(n)} \underline{\text{genau_eins}}(z_{t,1}, \dots, z_{t,k})$

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten: Zustand z_j
 $p_{t,i} = 1 \Leftrightarrow$ nach t Schritten: Kopfpos= i
 $b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten: Band[i]= a

Beweis (Skizze: SAT ist NP-schwer)

$$F_M(x) := A \wedge T_1 \wedge T_2 \wedge F \wedge R$$

Randbedingungen $R := R_z \wedge R_p \wedge R_b$:

Zustände $R_z := \bigwedge_{0 \leq t \leq p(n)} \text{genau_eins}(z_{t,1}, \dots, z_{t,k})$

Kopfpositionen $R_p := \bigwedge_{0 \leq t \leq p(n)} \text{genau_eins}(p_{t,-p(n)}, \dots, p_{t,p(n)})$
→ $0 \leq t \leq p(n)$

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten: Zustand z_j
 $p_{t,i} = 1 \Leftrightarrow$ nach t Schritten: Kopfpos= i
 $b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten: Band[i]= a

Beweis (Skizze: SAT ist NP-schwer)

$$F_M(x) := A \wedge T_1 \wedge T_2 \wedge F \wedge R$$

Randbedingungen $R := R_z \wedge R_p \wedge R_b$:

$$\text{Zustände } R_z := \bigwedge_{0 \leq t \leq p(n)} \text{genau_eins}(z_{t,1}, \dots, z_{t,k})$$

$$\text{Kopfpositionen } R_p := \bigwedge_{0 \leq t \leq p(n)} \text{genau_eins}(p_{t,-p(n)}, \dots, p_{t,p(n)})$$

$$\text{Bandinhalte } R_b := \bigwedge_{\substack{0 \leq t \leq p(n) \\ -p(n) \leq i \leq p(n)}} \text{genau_eins}(b_{t,i,a_1}, \dots, b_{t,i,a_\ell})$$

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten: Zustand z_j
 $p_{t,i} = 1 \Leftrightarrow$ nach t Schritten: Kopfpos= i
 $b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten: Band[i]= a

Beweis (Skizze: SAT ist NP-schwer)

$$F_M(x) := A \wedge T_1 \wedge T_2 \wedge F \wedge R$$

Randbedingungen $R := R_z \wedge R_p \wedge R_b$:

$$\text{Zustände } R_z := \bigwedge_{0 \leq t \leq p(n)} \text{genau_eins}(z_{t,1}, \dots, z_{t,k}) \rightsquigarrow O(p(n)^k) = O(1)$$

$$\text{Kopfpositionen } R_p := \bigwedge_{0 \leq t \leq p(n)} \text{genau_eins}(p_{t,-p(n)}, \dots, p_{t,p(n)}) \rightsquigarrow O(p(n)^2)$$

$$\text{Bandinhalte } R_b := \bigwedge_{\substack{0 \leq t \leq p(n) \\ -p(n) \leq i \leq p(n)}} \text{genau_eins}(b_{t,i,a_1}, \dots, b_{t,i,a_\ell}) \rightsquigarrow O(p(n)^2)$$

$$\text{genau_eins}(y_1, \dots, y_q) := \bigvee_{1 \leq i \leq q} y_i \wedge \bigwedge_{1 \leq i < j \leq q} \overline{y_i} \vee \overline{y_j} \rightsquigarrow O(q+q^2) = O(q^2)$$

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten: Zustand z_j
 $p_{t,i} = 1 \Leftrightarrow$ nach t Schritten: Kopfpos= i
 $b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten: Band[i]= a

Beweis (Skizze: SAT ist NP-schwer)

$$F_M(x) := A \wedge T_1 \wedge T_2 \wedge F \wedge R$$

Formelgröße:

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten: Zustand z_j
 $p_{t,i} = 1 \Leftrightarrow$ nach t Schritten: Kopfpos= i
 $b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten: Band[i]= a

Beweis (Skizze: SAT ist NP-schwer)

$$F_M(x) := A \wedge T_1 \wedge T_2 \wedge F \wedge R$$

Formelgröße:

$$|A| \in O(p(n))$$

$$|F| \in O(1)$$

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten: Zustand z_j
 $p_{t,i} = 1 \Leftrightarrow$ nach t Schritten: Kopfpos= i
 $b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten: Band[i]= a

Beweis (Skizze: SAT ist NP-schwer)

$$F_M(x) := A \wedge T_1 \wedge T_2 \wedge F \wedge R$$

Formelgröße:

$$|A| \in O(p(n))$$

$$|F| \in O(1)$$

$$|T_1| \in O((\underline{p(n)})^2)$$

$$|T_2| \in O((\underline{p(n)})^2)$$

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten: Zustand z_j
 $p_{t,i} = 1 \Leftrightarrow$ nach t Schritten: Kopfpos= i
 $b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten: Band[i]= a

Beweis (Skizze: SAT ist NP-schwer)

$$F_M(x) := A \wedge T_1 \wedge T_2 \wedge F \wedge R$$

Formelgröße:

$$|A| \in O(p(n))$$

$$|F| \in O(1)$$

$$|T_1| \in O((p(n))^2)$$

$$|T_2| \in O((p(n))^2)$$

$$|\text{genau_eins}(y_1, \dots, y_q)| \in O(q^2)$$

$$|R| \in O(\underline{(p(n))^3})$$

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten: Zustand z_j
 $p_{t,i} = 1 \Leftrightarrow$ nach t Schritten: Kopfpos= i
 $b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten: Band[i]= a

Beweis (Skizze: SAT ist NP-schwer)

$$F_M(x) := \underline{\quad} \wedge \underline{\underline{T_1}} \wedge \underline{\underline{T_2}} \wedge \underline{\quad F \quad} \wedge \underline{\quad}$$

Formelgröße:

$$|A| \in O(p(n))$$

$$|F| \in O(1)$$

$$|T_1| \in O((p(n))^2)$$

$$|T_2| \in O((p(n))^2)$$

$$|\text{genau_eins}(y_1, \dots, y_q)| \in O(q^2)$$

$$|R| \in O((p(n))^3)$$

Korrektheit:

Beobachtung: $F_M(x)$ modelliert akzeptierenden Berechnungspfad im Zustandsgraphen von $M(x)$

$F_M(x)$ erfüllbar

\exists Belegung, die $F_M(x)$ erfüllt \rightarrow \exists Folgekonfigurationsfolge zu akz. Kof.

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten: Zustand z_j
 $p_{t,i} = 1 \Leftrightarrow$ nach t Schritten: Kopfpos= i
 $b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten: Band[i]= a

Beweis (Skizze: SAT ist NP-schwer)

$$F_M(x) := A \wedge T_1 \wedge T_2 \wedge F \wedge R$$

Formelgröße:

$$|A| \in O(p(n))$$

$$|F| \in O(1)$$

$$|T_1| \in O((p(n))^2)$$

$$|T_2| \in O((p(n))^2)$$

$$|\text{genau_eins}(y_1, \dots, y_q)| \in O(q^2)$$

$$|R| \in O((p(n))^3)$$

Korrektheit:

Beobachtung: $F_M(x)$ modelliert akzeptierenden Berechnungspfad im Zustandsgraphen von $M(x)$

$x \in L \Leftrightarrow$ es gibt akzeptierenden Berechnungspfad im Zustandsgraphen von $M(x)$

Satz von Cook und Levin

Theorem

SAT ist NP-vollständig.

$z_{t,j} = 1 \Leftrightarrow$ nach t Schritten: Zustand z_j
 $p_{t,i} = 1 \Leftrightarrow$ nach t Schritten: Kopfpos= i
 $b_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten: Band[i]= a

Beweis (Skizze: SAT ist NP-schwer)

$$F_M(x) := A \wedge T_1 \wedge T_2 \wedge F \wedge R$$

Formelgröße:

$$|A| \in O(p(n))$$

$$|F| \in O(1)$$

$$|T_1| \in O((p(n))^2)$$

$$|T_2| \in O((p(n))^2)$$

$$|\text{genau_eins}(y_1, \dots, y_q)| \in O(q^2)$$

$$|R| \in O((p(n))^3)$$

Korrektheit:

Beobachtung: $F_M(x)$ modelliert akzeptierenden Berechnungspfad im Zustandsgraphen von $M(x)$

$x \in L \Leftrightarrow$ es gibt akzeptierenden Berechnungspfad im Zustandsgraphen von $M(x)$

$\Leftrightarrow F_M(x)$ erfüllbar

TQBF & PSPACE Theorem

TQBF ist PSPACE-vollständig.

Satz v. Savitch
Satz v. Cook

$$\text{TQBF \& PSPACE} = \bigcup_{k \in \mathbb{N}} \text{DSPACE}(n^k)$$

Theorem

TQBF ist PSPACE-vollständig.

Beweis (Skizze: TQBF ist PSPACE-schwer)

zu zeigen: $\forall L \in \text{PSPACE} \ L \leq_m^p \text{TQBF}$.

Sei $L \in \text{PSPACE}$. Dann existiert DTM M mit $L = T(M)$, platzbeschränkt durch Polynom p .

TQBF & PSPACE Theorem

TQBF ist PSPACE-vollständig.

Beweis (Skizze: TQBF ist PSPACE-schwer)

zu zeigen: $\forall_{L \in \text{PSPACE}} L \leq_m^p \text{TQBF}$.

Sei $L \in \text{PSPACE}$. Dann existiert DTM M mit $L = T(M)$, platzbeschränkt durch Polynom p .

Sei $M = (Z, \Sigma, \Gamma, \delta, z_1, \square, E)$ mit $\Gamma = \{a_1 = \square, \dots, a_\ell\}$ und $Z = \{z_1, \dots, z_k\}$.

TQBF & PSPACE

Theorem

TQBF ist PSPACE-vollständig.

Beweis (Skizze: TQBF ist PSPACE-schwer)

zu zeigen: $\forall L \in \text{PSPACE} \ L \leq_m^p \text{TQBF}$.

Sei $L \in \text{PSPACE}$. Dann existiert DTM M mit $L = T(M)$, platzbeschränkt durch Polynom p .

Sei $M = (Z, \Sigma, \Gamma, \delta, z_1, \square, E)$ mit $\Gamma = \{a_1 = \square, \dots, a_\ell\}$ und $Z = \{z_1, \dots, z_k\}$.

Sei K_x die Menge aller möglichen Konfigurationen von M bei Eingabe x

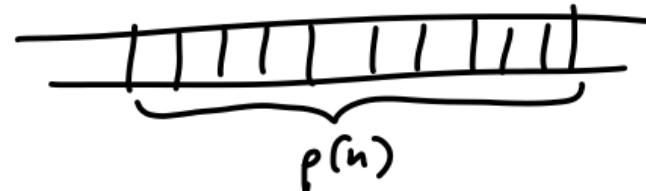
Sei $S \in K_x$ die Startkonfiguration von M bei Eingabe x . Argument ähnlich zu Satz v. Savitch:

M akzeptiert $x \Leftrightarrow \exists_{T \in K_x} T$ akzeptierend $\wedge \text{reach}_x(S, T, k \cdot p(n) \cdot |\Gamma|^{p(n)})$

$$\# \text{Konfigurationen} = |K_x|$$



$$\text{reach}(S, T, 3!) = 1$$



TQBF & PSPACE

Theorem

TQBF ist PSPACE-vollständig.

Beweis (Skizze: TQBF ist PSPACE-schwer)

zu zeigen: $\forall L \in \text{PSPACE} \ L \leq_m^p \text{TQBF}$.

Sei $L \in \text{PSPACE}$. Dann existiert DTM M mit $L = T(M)$, platzbeschränkt durch Polynom p .

Sei $M = (Z, \Sigma, \Gamma, \delta, z_1, \square, E)$ mit $\Gamma = \{a_1 = \square, \dots, a_\ell\}$ und $Z = \{z_1, \dots, z_k\}$.

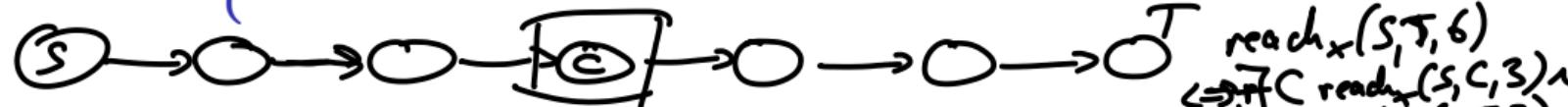
Sei \mathcal{K}_x die Menge aller möglichen Konfigurationen von M bei Eingabe x

Sei $S \in \mathcal{K}_x$ die Startkonfiguration von M bei Eingabe x . Argument ähnlich zu Satz v. Savitch:

M akzeptiert $x \Leftrightarrow \exists_{T \in \mathcal{K}_x} T$ akzeptierend $\wedge \text{reach}_x(S, T, k \cdot p(n) \cdot |\Gamma|^{p(n)})$

($\text{reach}_x(Q, R, j) \hat{=} \text{es gibt einen } Q-R\text{-Pfad der Länge } \leq j$ im Konfigurationsgraph von $M(x)$)

$$\boxed{\text{reach}_x(Q, R, j) := \begin{cases} R \text{ ist Folgekonfiguration von } Q \vee (R = Q) & \text{falls } j = 1 \\ \exists_{C \in \mathcal{K}_x} \text{reach}_x(Q, C, \lfloor j/2 \rfloor) \wedge \text{reach}_x(C, R, \lceil j/2 \rceil) & \text{falls } j > 1 \end{cases}}$$



TQBF & PSPACE

Theorem

TQBF ist PSPACE-vollständig.

$$S(j) = 2 \cdot S(j/2) \quad S(1) = O(n)$$

$$\rightsquigarrow S(j) = j$$

Beweis (Skizze: TQBF ist PSPACE-schwer)

zu zeigen: $\forall L \in \text{PSPACE} \ L \leq_m^P \text{TQBF}$.

Sei $L \in \text{PSPACE}$. Dann existiert DTM M mit $L = T(M)$, platzbeschränkt durch Polynom p .

Sei $M = (Z, \Sigma, \Gamma, \delta, z_1, \square, E)$ mit $\Gamma = \{a_1 = \square, \dots, a_\ell\}$ und $Z = \{z_1, \dots, z_k\}$.

Sei \mathcal{K}_x die Menge aller möglichen Konfigurationen von M bei Eingabe x

Sei $S \in \mathcal{K}_x$ die Startkonfiguration von M bei Eingabe x . Argument ähnlich zu Satz v. Savitch:

M akzeptiert $x \Leftrightarrow \exists_{T \in \mathcal{K}_x} T$ akzeptierend $\wedge \text{reach}_x(S, T, k \cdot p(n) \cdot |\Gamma|^{p(n)})$ \rightarrow exponentiell in n !!!

$\text{reach}_x(Q, R, j) \hat{=} \text{es gibt einen } Q-R\text{-Pfad der Länge } \leq j \text{ im Konfigurationsgraph von } M(x)$

$$\text{reach}_x(Q, R, j) := \begin{cases} R \text{ ist Folgekonfiguration von } Q \vee (R = Q) & \text{falls } j = 1 \\ \exists_{C \in \mathcal{K}_x} \text{reach}_x(Q, C, \lfloor j/2 \rfloor) \wedge \text{reach}_x(C, R, \lceil j/2 \rceil) & \text{falls } j > 1 \end{cases}$$

$\xrightarrow{\text{D}, D' \in \mathcal{K}_x} \text{reach}(D, D', \lfloor j/2 \rfloor) \text{ für } D = Q \wedge D' = C \quad \xrightarrow{\text{D} = C \wedge D' = R} \text{reach}(D, D', \lceil j/2 \rceil) \text{ für }$

$\sim |\text{reach}_x(Q, R, j)| \approx 2 \cdot |\text{reach}_x(Q, R, \lfloor j/2 \rfloor)| \approx j \sim \times$

TQBF & PSPACE

Theorem

TQBF ist PSPACE-vollständig.

Beweis (Skizze: TQBF ist PSPACE-schwer)

zu zeigen: $\forall L \in \text{PSPACE} \ L \leq_m^p \text{TQBF}$.

Sei $L \in \text{PSPACE}$. Dann existiert DTM M mit $L = T(M)$, platzbeschränkt durch Polynom p .

Sei $M = (Z, \Sigma, \Gamma, \delta, z_1, \square, E)$ mit $\Gamma = \{a_1 = \square, \dots, a_\ell\}$ und $Z = \{z_1, \dots, z_k\}$.

Sei \mathcal{K}_x die Menge aller möglichen Konfigurationen von M bei Eingabe x

Sei $S \in \mathcal{K}_x$ die Startkonfiguration von M bei Eingabe x . Argument ähnlich zu Satz v. Savitch:

M akzeptiert $x \Leftrightarrow \exists_{T \in \mathcal{K}_x} T$ akzeptierend $\wedge \text{reach}_x(S, T, k \cdot p(n) \cdot |\Gamma|^{p(n)})$

$\text{reach}_x(Q, R, j) \hat{=} \text{es gibt einen } Q\text{-}R\text{-Pfad der Länge } \leq j \text{ im Konfigurationsgraph von } M(x)$

$$\text{reach}_x(Q, R, j) := \begin{cases} R \text{ ist Folgekonfiguration von } Q \vee (R = Q) & \text{falls } j = 1 \\ \exists C \in \mathcal{K}_x \forall_{D, D' \in \mathcal{K}_x} ((D = Q \wedge D' = C) \vee (D = C \wedge D' = R)) \\ (D, D') \in \{(Q, C), (C, R)\} & \rightarrow \text{reach}_x(D, D', [j/2]) \quad \text{falls } j > 1 \end{cases}$$

$\sim |\text{reach}_x(Q, R, j)| \approx 2 \cdot |\text{reach}_x(Q, R, [j/2])| \approx j \sim x$

TQBF & PSPACE Theorem

Zusammen-
bruch

PH falls nur konstant
viele Quantorenwechseln

TQBF ist **PSPACE**-vollständig.

Beweis (Skizze: TQBF ist PSPACE-schwer)

zu zeigen: $\forall L \in \text{PSPACE} \ L \leq_m^p \text{TQBF}.$

Sei $L \in \text{PSPACE}$. Dann existiert DTM M mit $L = T(M)$, platzbeschränkt durch Polynom p .

Sei $M = (Z, \Sigma, \Gamma, \delta, z_1, \square, E)$ mit $\Gamma = \{a_1 = \square, \dots, a_\ell\}$ und $Z = \{z_1, \dots, z_k\}$.

Sei \mathcal{K}_x die Menge aller möglichen Konfigurationen von M bei Eingabe x .

Sei $S \in \mathcal{K}_x$ die Startkonfiguration von M bei Eingabe x . Argument ähnlich zu Satz v. Savitch:

\boxed{M} akzeptiert $x \Leftrightarrow \exists_{T \in \mathcal{K}_x} T$ akzeptierend $\wedge \text{reach}_x(S, T, k \cdot p(n) \cdot |\Gamma|^{p(n)})$, $\log(x) \in P(\underline{u})^{O(1)}$

reach_x(Q, R, j) $\hat{=}$ es gibt einen Q-R-Pfad der Länge $\leq j$ im Konfigurationsgraph von M(x)

$$\underbrace{\text{reach}_x(Q, R, j)} := \begin{cases} R \text{ ist Folgekonfiguration von } Q \vee \underline{(R = Q)} & \text{falls } j = 1 \\ \exists c \in \mathcal{K}_x \forall D, D' \in \mathcal{K}_x ((D = Q \wedge D' = C) \vee \underline{(D = C \wedge D' = R)}) \\ \qquad\qquad\qquad \rightarrow \text{reach}_x(D, D', [j/2]) & \text{falls } j > 1 \end{cases}$$

$$\leadsto |\text{reach}_x(Q, R, j)| \approx O(\cancel{j}) + |\text{reach}_x(\cancel{Q}, \cancel{R}, [j/2])| \in O(\log j \cdot \cancel{p(j)}) \checkmark$$