user: theoric
pw: spass

Ausgewählte Kapitel der Theoretischen Informatik

1. Komplexitätstheorie

1.1. Einige Grundlagen

Def: Sei Σ ein Alphabet, d.h. eine endliche Menge von Buchstaben. Σ^* bezeichnet die Menge aller Worte über Σ. Eine Teilmenge $L \subseteq \Sigma^*$ heißt Sprache (über Σ).

Ein Problem ist eine Relation $\mathcal{P} \subseteq \mathcal{I} \times \mathcal{S}$, wobei $\mathcal{I}$ die Menge der Probleminstanzen und $\mathcal{S}$ die Menge der Problemlösungen ist.

Ist $\mathcal{P}$ rechtseindeutig, d.h. eine (partielle) Fkt, dann heißt $\mathcal{P}$ Funktionsproblem.

Ist $\mathcal{P}$ ein Funktionsproblem und $\mathcal{S} = \{0,1\}$, dann heißt $\mathcal{P}$ auch Entscheidungsproblem.

Bsp 1: Sei $\mathcal{P}$ ein Entscheidungsproblem, dann ist $L =_{def} \{ x \in \mathcal{I} \mid \mathcal{P}(x) = 1 \}$ eine Sprache und $\mathcal{P}$ wird auch als Wortproblem der Sprache L bezeichnet.

Bsp 2: Sei V eine Menge von aussagenlogischen Variablen, etwa $V = \{x_1, x_2, x_3, \ldots\}$ und sei $\mathcal{I}$ die Menge aller aussagenlogischen Formeln über (Teilmengen) von V.

a) Sei $\mathcal{G}$ die Menge aller Fkten von endlichen Teilmengen von V nach $\{0,1\}$, dann ist $sat \subseteq \mathcal{I} \times \mathcal{G}$ definiert durch $(H, f) \in sat$ gdw H ist aussagenlogische Formel über $V' \subseteq V$, $f: V' \to \{0,1\}$ und f ist erfüllende Belegung von H.

b) Sei $V' \subseteq V$. Nun wird eine Ordnung $<$ auf der Menge der Fkten $f: V' \to \{0,1\}$ wie folgt definiert:

$$f_1 < f_2 \text{ gdw } \exists x_i \in V' \left(f_1(x_i) < f_2(x_i) \wedge \forall x_j \in V' \left(j < i \Rightarrow f_1(x_j) = f_2(x_j) \right) \right)$$

„lexikographische Ordnung"

LEXMINSAT: $\mathcal{I} \to \mathcal{G}$, wobei

$$\text{LEXMINSAT}(H) =_{def} \begin{cases} \text{kleinste Fkt } f: V' \to \{0,1\}, \text{ sodass } H \text{ benutzt Variablen aus } V' \text{ und } (H, f) \in sat, \text{ falls solch ein } f \text{ existiert} \\ f_0 \in \mathcal{G} \text{ mit } f_0(x) = 0 \text{ f.a. } x \in V', \text{ sonst} \end{cases}$$

Bem: Eine Probleminstanz $x \in \mathcal{I}$ kann ein n-Tupel sein. D.h. die Def. deckt auch n-stellige Fktprobleme ab.

Ziele:
- Untersuchung der Komplexität v. Berechnungen (z.B. Anzahl der Schritte / Laufzeit oder Speicherbits)

Fokus ↗

- Untersuchung der Komplexität von Algorithmen (z.B. Anzahl der Befehle)

1.2. Komplexitätsmaße und -Klassen

Def: Ein Algorithmus A (TM, RAM, C) *berechnet* eine Fkt $f_A: \mathcal{I} \to \mathcal{S}$, wenn

$$f_A(x) = \begin{cases} \text{Ergebnis von A bei Eingabe } x, & \text{falls A stoppt} \\ \text{undef}, & \text{sonst} \end{cases}$$

Der Algorithmus A hat Φ-Komplexität (Laufzeit, Speicherplätze), wobei $\Phi_A: \mathcal{I} \to \mathbb{N}$ und

$$\Phi_A(x) =_{def} \begin{cases} \Phi\text{-Komplexität von A bei Eingabe } x, & \text{falls A stoppt} \\ \text{undef}, & \text{sonst} \end{cases}$$

Sei $\Phi_A: \mathbb{N} \to \mathbb{N}$ vermöge

$$\Phi_A(n) =_{def} \max_{|x|=n} \Phi_A(x), \text{ wobei } |x| \text{ Länge von } x.$$

Dann heißt Φ_A *worst-case Komplexität.*

D.h. die Komplexität wird <u>immer</u> (fast) über die Eingabelänge gemessen!

<u>Def</u>: „Komplexitätsklasse": Gegeben sei Φ- Komplexität, τ-Algorithmentyp (z.B. TM, RAM, C_1...) und eine Schrankenfkt $t: \mathbb{N} \to \mathbb{N}$, dann

$F_\tau \Phi(t) =_{def} \{ f \mid f$ total und es ex. ein Algorithmus A vom Typ τ der f berechnet und $\Phi_A \leq_{ae} t \}$

$\tau \Phi(t) =_{def} \{ L \mid L$ Sprache und es ex. ein Algorithmus A vom Typ τ der das Wortproblem von L entscheidet und $\Phi_A \leq_{ae} t \}$

$t_1 \leq_{ae} t_2$ gdw es ex. ein $n_0 \geq 0$ und $t_1(n) \leq t_1(n)$ f.a. $n \geq n_0$

„t_1 almost everywhere less or equal than t_2"

$F_\tau \Phi(O(t)) =_{def} \bigcup_{k \geq 0} F_\tau \Phi(k \cdot t)$ vgl.

„Funktionsklasse"

$\tau \Phi(O(t)) =_{def} \bigcup_{k \geq 0} \tau \Phi(k \cdot t)$

„Sprachklasse"

(4a)

Fachhochschule Wiesbaden
Prof. Dr. Steffen Reith

1 Random-Access-Maschinen

Die *Random-Access-Maschinen* (kurz: RAM) sind ein mathematisches Modell für reale Rechner. Eine RAM besteht aus einer Steuereinheit, aus unendlich vielen durchnumerierten Registern **R0**, **R1**, **R2**, ... und dem Befehlsregister **BR**. Jedes Register kann eine natürliche Zahl enthalten, wobei sich die n Eingaben am Anfang in den Registern **R0**, ..., **Rn** befinden. Alle nicht benutzten Register enthalten zu Beginn 0. Das Ergebnis wird am Ende in Register **R0** abgelegt.

Folgende Befehle sind zulässig:

Befehl	Wirkung	
Transportbefehle		
Ri ← Rj	<Ri> := <Rj>	 := + 1
Ri ← RRj	<Ri> := <R<Rj>>	 := + 1
RRi ← Rj	<R<Ri>> := <Rj>	 := + 1
Arithmetische Befehle		
Ri ← k	<Ri> := k	 := + 1
Ri ← Rj + Rk	<Ri> := <Rj> + <Rk>	 := + 1
Ri ← Rj - Rk	<Ri> := <Rj> - <Rk>	 := + 1
Sprungbefehle		
GOTO m		 := m
IF Ri = 0 GOTO m	 := m falls <Ri> = 0	 := m sonst
IF Ri > 0 GOTO m	 := m falls <Ri> > 0	 := m sonst

Dabei bezeichnet <Ri> den Inhalt des Registers i.

Gegeben sei die folgende RAM:

```
0  R3 ← 1
1  IF R1 = 0 GOTO 5   falls <R1> > 0 führe Schleife durch
2  R2 ← R2 + R0       addiere x zu <R2>
3  R1 ← R1 - R3       dekrementiere R1
4  GOTO 1
5  R0 ← R2            Ergebnis nach R0
6  STOP
```

2 Aufgaben

1. Simulieren Sie die Wirkungsweise des oben angegebenen RAM-Programms für die Eingaben $x = 5$ und $y = 3$. Geben Sie an, welche Funktion durch diese RAM berechnet wird.

(4b)

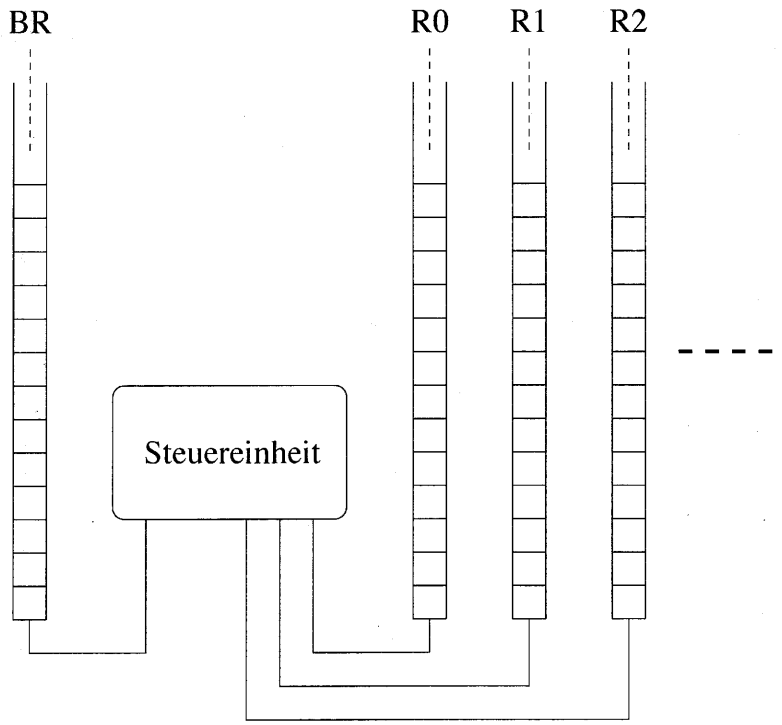


Abbildung 1: Bestandteile einer Random-Access-Maschine

2. Entwickeln Sie ein RAM-Programm, das die Funktion `ggT(x,y)` berechnet (`ggT` = größter gemeinsamer Teiler).

Fachhochschule Wiesbaden
Prof. Dr. Steffen Reith

1 Turingmaschinen

Eine Turingmaschine (kurz TM), eingeführt von Alan Turing 1936 in seiner für die Informatik zentralen Arbeit „On computable numbers with an application to the Entscheidungsproblem“, hat die folgenden Bestandteile:

- Mehrere ($k \geq 1$) beidseitig unendliche, in *Felder* unterteilte *Bänder*. In jedem Feld steht ein Symbol (Buchstabe) aus einem endlichen *Bandalphabet* Σ. Das Leerzeichen $\square \in \Sigma$ deutet an, dass in diesem Feld eigentlich nichts steht.
- Ein *Lese-* und *Schreibkopf* für jedes Band (kurz *Kopf* genannt), der sich von Feld zu Feld bewegen und den Inhalt des jeweils betrachteten Feldes lesen und ändern kann.
- Eine *Steuereinheit*, die sich in einem der *Zustände* aus einer endlichen Zustandsmenge Z befindet, Informationen über die von den Köpfen gelesenen Symbole bekommt und deren Aktivitäten steuert. Es gibt zwei besonders ausgezeichnete Zustände: den *Startzustand* und den *Endzustand*.

Eine Turingmaschine arbeitet taktweise und kann in Abhängigkeit

- vom gegenwärtigen Zustand und
- von den durch die Köpfe gelesenen k Bandsymbole

gleichzeitig

- einen neuen Zustand annehmen
- die k gelesenen Bandsymbole verändern und
- jeden der Köpfe um maximal ein Feld bewegen.

Das Verhalten einer Turingmaschine in einem Takt wird durch die (totale) Überführungfunktion

$$\delta\colon Z \times \Sigma^k \to Z \times \Sigma^k \times \{L, N, R\}^k$$

festgelegt. Für $z \in Z, a_1, \ldots a_k \in \Sigma$ beschreibt

$$\delta(z, a_1, \ldots, a_k) = (z', a'_1, \ldots, a'_k, \sigma_1, \ldots, \sigma_k)$$

das Verhalten der Turingmaschine im Zustand z, wenn sie auf dem Band i das Symbol a_i gelesen hat und dann in den Zustand z' übergegangen ist und auf Band i das Symbol a'_i geschrieben hat. Dannach wird der Kopf i nach links (L), nicht (N) oder rechts (R)

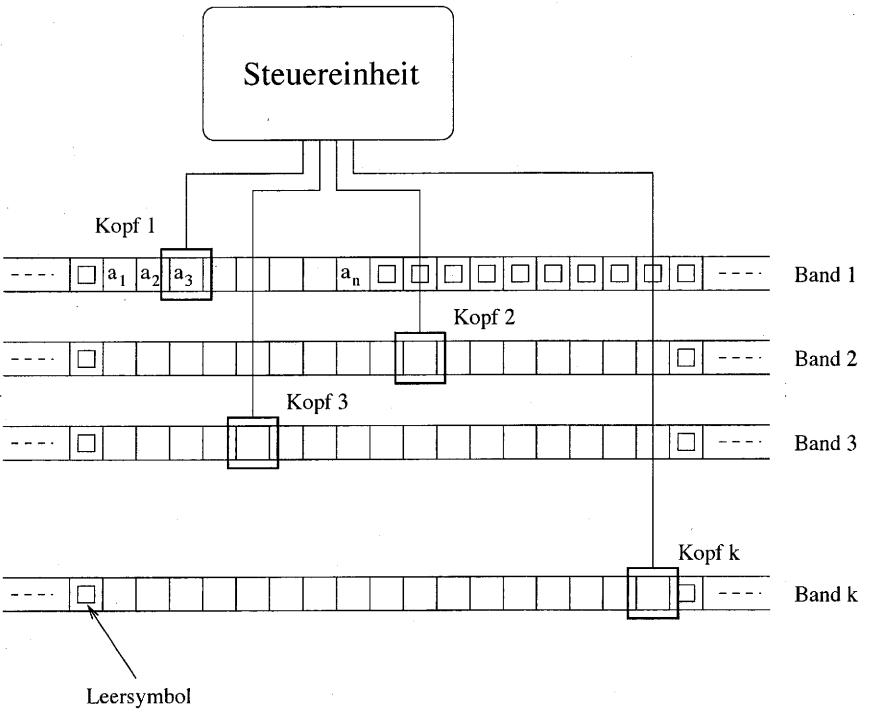


Abbildung 1: Bestandteile einer Turingmaschine

bewegt. Der Startzustand ist z_0 und der Endzustand ist z_1. Die Turingmaschine beginnt ihre Arbeit immer im Zustand z_0 und findet die Eingabe auf dem ersten Band. Dabei befindet sich der Kopf fur die Eingabe immer ganz links auf der Eingabe. Die TM hält (stoppt), falls sie in den Zustand z_1 gelangt und schreibt die Ausgabe auch auf Band 1.

Es ist folgende Aufgabe zu lösen:
Erkenne ob die Eingabe ein Palindrom ist, d.h. ob die Buchstaben von links nach rechts gelesen das gleiche Wort bilden, wie von rechts nach links gelesen. D.h. wir sollen ein Turing-Programm für die folgende Funktion angeben:

$$\text{palindrom}(x) =_{\text{def}} \begin{cases} \text{a}, & \text{falls } x \text{ ein Palindrom} \\ \text{b}, & \text{sonst} \end{cases}$$

Idee:

- Merke Buchstabe links im Zustand und lösche ihn
- Vergleich mit Buchstaben rechts:
  - Wenn die Buchstaben nicht übereinstimmen, dann alles löschen und b schreiben
  - Bei Übereinstimmung letztes Zeichen löschen und ganz nach links wandern. Dann diesen Prozess wiederholen.

Die Zustände haben die folgende Bedeutung:

z_a: Buchstabe a gemerkt

z_b: Buchstabe b gemerkt

4e

z'_a: Ein Schritt nach links und a testen

z'_b: Ein Schritt nach links und b testen

z_0: Startzustand, d.h. „Merkprozess“ starten

z_1: Endzustand

z_2: Test positiv, d.h. nach links laufen und Prozess wiederholen

z_3: Test negativ, alles löschen.

Nun geben wir eine Turingmaschine $M = (\Sigma, Z, f, z_0, z_1)$ an, die das Problem löst.

- $\Sigma = \{a, b, \square\}$
- $Z = \{z_a, z_b, z'_a, z'_b, z_0, z_1, z_2, z_3\}$
- f Siehe Bild 2
- z_0 Startzustand
- z_1 Endzustand

$$
\begin{array}{llllll}
z_0 & \square & \to & z_1 & a & N \quad // \text{ leeres Wort} \\
z_0 & a & \to & z_a & \square & R \quad // \text{ a merken nach rechts} \\
z_0 & b & \to & z_b & \square & R \quad // \text{ b merken nach rechts} \\
 & & & & & \\
z_a & a & \to & z_a & a & R \quad // \text{ nach rechts, a gemerkt} \\
z_a & b & \to & z_a & b & R \\
z_a & \square & \to & z'_a & \square & L \quad // \text{ rechtes Ende gefunden} \\
z'_a & a & \to & z_2 & \square & L \quad // \text{ Vergleich positiv} \\
z'_a & b & \to & z_3 & \square & L \quad // \text{ Vergleich negativ} \\
 & & & & & \\
z'_a & \square & \to & z_1 & a & N \quad // \text{ War Palindrom} \\
 & & & & & \\
z_b & a & \to & z_b & a & R \quad // \text{ Nach rechts, b gemerkt} \\
z_b & b & \to & z_b & b & R \\
z_b & \square & \to & z'_b & \square & L \quad // \text{ rechtes Ende gefunden} \\
z'_b & b & \to & z_2 & \square & L \quad // \text{ Vergleich positiv} \\
z'_b & a & \to & z_3 & \square & L \quad // \text{ Vergleich negativ} \\
 & & & & & \\
z'_b & \square & \to & z_1 & a & N \quad // \text{ War Palindrom} \\
 & & & & & \\
z_2 & a & \to & z_2 & a & L \quad // \text{ Nach links und} \\
z_2 & b & \to & z_2 & b & L \quad // \text{ neuen Vergleich starten} \\
z_2 & \square & \to & z_0 & \square & R \\
 & & & & & \\
z_3 & a & \to & z_3 & \square & L \quad // \text{ Band löschen und nach links} \\
z_3 & b & \to & z_3 & \square & L \\
z_3 & \square & \to & z_1 & b & N
\end{array}
$$

Abbildung 2: Ein Turingprogramm zum Erkennen von Palindromen

1.3. Deterministische Berechnungen

Für die Rechenzeit eines <u>deterministischen</u> Algorithmentyps schreiben wir DTIME (kurz: Zeit) und für den Speicherplatz DSPACE (kurz: Platz / Raum)

In der Theoretischen Informatik werden besonders die Algorithmentyp TM und RAM verwendet, da sie sich leicht analysieren lassen. Hier: Ein- und Mehrband-TMs evtl. mit (nur lesbarem) Eingabeband und (nur schreibbarem) Ausgabeband.
Wird eine mehrstellige Fkt berechnet, so sind die Argumente auf dem Eingabeband durch ein besonderes Trennsymbol (z.B. , oder *) voneinander getrennt.

Sei M eine (Mehrband) TM und x eine Eingabeinstanz, dann

$$DTIME_M(x) =_{def} \begin{cases} \text{Anz. der Takte von M bei Eingabe } x, \text{ falls M stoppt} \\ \text{undef., sonst} \end{cases}$$

$$DSPACE_M(x) =_{def} \begin{cases} \text{Anzahl der von M besuchten Felder bei Eingabe } x, \text{ falls M stoppt} \\ \text{undef., sonst} \end{cases}$$

Da wir nur TMen verwenden, wird τ ab jetzt nicht mehr verwendet.

Bsp: Sei $A = \{0^k 1^k \mid k \geq 0\} \subseteq \{0,1\}^*$, dann kann das Wortproblem von A durch folgende 1-Band TM entschieden werden:

i) Prüfe, ob eine 0 rechts von einer 1 steht
ii) Wieder hole Schritt iii), solange noch eine 0 und 1 auf dem Band stehen
iii) Streiche eine 0 und eine 1
iv) Falls 0en oder 1en übrig bleiben, dann schreibe 0 ($\hat{=}$ ablehnen) auf das Band, sonst 1 ($\hat{=}$ akzeptieren)

Zeitbedarf bei Eingabelänge n:

i) $O(n)$
ii) $O(n)$
iii) $O(n)$

} Schleife wird $O(n)$-mal durchlaufen

$\Rightarrow$ totale Rechenzeit: $O(n^2)$

$\Rightarrow$ $A \in DTIME(n^2)$

Eine 2-Band Maschine erledigt die Aufgabe in Zeit $O(n)$.

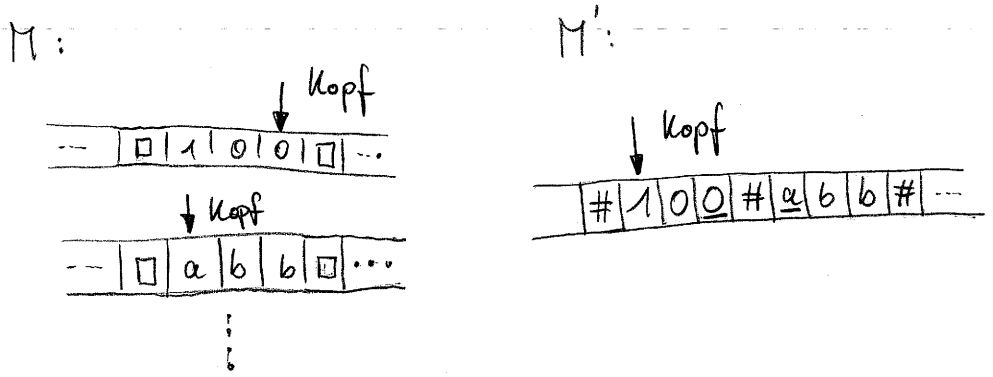
Platzbedarf: $O(n)$, d.h. $A \in DSPACE(n)$

Die Leistungsfähigkeit von 1-Band und Mehrbandmaschinen unterscheiden sich kaum:

Satz: Sei $t(n) \geq n$, dann kann jede Mehrband TM M mit $DTIME_M(t)$ von einer 1-Band TM M' in $DTIME_{M'}(\mathcal{O}(t^2))$ simuliert werden.

Beweis: M habe k Bänder, dann wird M' wie folgt konstruiert

M' speichert auf ihrem Band die Inhalte und Kopfpositionen hintereinander, getrennt durch ein Sonderzeichen (z.B. #).
Ein Schritt von M bedeutet nun, dass M' das gesamte Band durchlaufen muss, um alle Änderungen durchzuführen. (Bandinhalte, Kopfpositionen).



⇒ Ein Simulationsschritt benötigt $O(t(n))$ Zeit, also beträgt die gesamte Zeit $O(t^2(n))$. #

Bem: Der letzte Satz läßt sich auf $O(t \log t)$ verbessern.

1.4. Nichtdeterministische Berechnungen

Bei einer deterministischen TM ex. maximal ein Nachfolgeschritt in einer Berechnung, wie die Überführungsfkt zeigt:

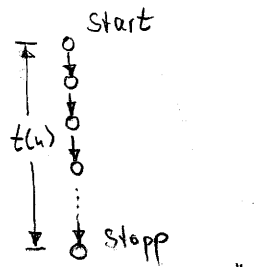
$$\delta : Z \times \Sigma \to Z \times \Sigma \times \{L, R, N\}$$

- Z: Menge d. Zustände
- Σ: (Arbeits)alphabet
- $\{L, R, N\}$: mögliche Kopfbewegungen

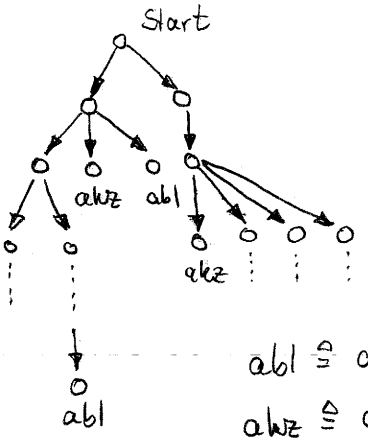
Bei nichtdeterministischen Berechnungen ex. oftl. mehrere Nachfolgeschritte, d.h. die TM verzweigt sich in mehrere Kopien.

⇒ Durch die nichtdeterministische Berechnung entsteht ein Baum.

„Berechnungsbaum"



„Deterministische TM"



abl ≙ ablehnen
akz ≙ akzeptieren

„Nichtdeterministische TM"

Die Möglichkeit zur (mehrfachen) Verzweigung ergibt die Überführungsfkt

$$\delta : Z \times \Sigma \to \mathcal{P}(Z \times \Sigma \times \{L, R, N\})$$

Potenzmenge

Sei nun M eine nicht det. TM und x eine Eingabe instanz,

dann

$$NTIME_M(x) =_{def} \begin{cases} \text{Maximale Anzahl von Takten} \\ \text{auf allen Berechnungspfaden, falls } M \text{ stoppt} \\ \text{mit Eingabe } x \\ \text{undef, sonst} \end{cases}$$

$$NSPACE_M(x) =_{def} \begin{cases} \text{Maximale Anzahl von } M \text{ besuchten} \\ \text{Feldern, falls } M \text{ stoppt mit Eingabe } x \\ \text{undef, sonst} \end{cases}$$

Def: Eine nicht det. Maschine M <u>akzeptiert</u> eine Sprache L, wenn

$$x \in L \text{ gdw } \exists z\,(M(x|z) = 1), \text{ wobei}$$

$$M(x|z) =_{def} \begin{cases} \text{Resultat von } M \text{ bei Eingabe } x \\ \text{auf dem Pfad } z, \text{ falls } M \text{ hält} \\ \text{undef, sonst} \end{cases}$$

Bem:

- Jede(r) deterministische(r) Algorithmus / Maschine ist nicht det. Algorithmus / Maschine (vom gleichen Typ).
- „Implementierung" durch
  - RAMs: mehrere Befehle mit gleicher Nummer
  - TMs: $sa \to s_1 a_1 \sigma_1$, $sa \to s_2 a_2 \sigma_2$ } mehrere Befehle mit gleicher linken Seite
  - C: neben $s_1; s_2; \ldots; s_n$ gibt es $s_1 | s_2 | \ldots | s_n$

Proposition: Sei $t: \mathbb{N} \to \mathbb{N}$, dann gilt
$DTIME(t) \subseteq NTIME(t)$ und
$DSPACE(t) \subseteq NSPACE(t)$

1.5 Grundlegende Beziehungen zwischen Komplexitätsklassen

Satz: Sei $t(n) \geq n$, dann gilt

$$NTIME(t(n)) \subseteq DTIME(2^{O(t(n))})$$

Beweis: Sei M eine nichtdet. TM, die in Zeit $t(n)$ arbeitet. Bei Eingabe w spannt M einen Berechnungsbaum der Tiefe $\leq t(|w|)$ auf, wobei $n = |w|$.

- Baumknoten $\hat{=}$ „Situation d. Berechnung", d.h. Zustand, Inhalte d. Bänder u. Kopfpositionen (so genannte Konfigurationen)
- Verzweigungen $\hat{=}$ nichtdet. Wahlmöglichkeiten

Sei M' eine det. TM, die den Berechnungsbaum Knoten für Knoten konstruiert und nach einer akz. Konfiguration sucht (vgl. Breitensuche)

Die Größe des Baums ist $\leq \sum_{i=0}^{t(n)} b^i \overset{\text{geo. Reihe}}{=} \frac{b^{t(n)+1} - 1}{b-1}$

$= 2^{O(t(n))}$)

wenn b der max. Verzweigungsgrad des Berechnungsbaums ist. Ein Schritt von einer Konfiguration zu einer nächsten dauert max. $O(t(n))$ (Zeitschranke d. nichtdet. Berechnung), d.h. die gesamte Rechenzeit beträgt $O(t(n)) \cdot 2^{O(t(n))}$

$= 2^{O(t(n))}$ #

Satz: Sei $t(n) \geq n$, dann gilt

$$NTIME(t(n)) \subseteq DSPACE(t(n))$$

Beweis: Verwende die Simulation aus letztem Beweis, aber der Berechnungsbaum wird nun in Tiefensuche durchmustert

Platzbedarf: Konfiguration benötigt $O(t(n))$
vollständiger Pfad zur aktuellen Konfig benötigt $O(t(n))$ (Pfad ist eine b-näre Zahl der Länge $\leq t(n)$)

$\Rightarrow$ Speicherbedarf gesamt: $O(t(n))$ #

Folgerung: Sei $t(n) \geq n$, dann gilt

$$DTIME(t(n)) \subseteq DSPACE(t(n))$$

Satz: Sei $s(n) \geq \log n$, dann gilt

$$DSPACE(s(n)) \subseteq DTIME(2^{O(s(n))})$$

Beweis: Sei M eine TM mit Platzbedarf s, Anzahl von Zuständen q, Anzahl von Bändern k und das Arbeitsalphabet habe r Buchstaben.

Klar: Durchläuft M eine bel. Konfiguration zweimal, dann befindet sich M in einer Endlosschleife. Da M eine Sprache entscheidet, kann diese Situation nicht vorkommen

$\Rightarrow$ Rechenzeit bei Eingabelänge n

$\leq$ # Konfigurationen bei Eingabelänge n

$\leq$ # Zustände $\cdot$
möglicher Kopfpositionen auf d. Eingabeband $\cdot$
mögliche Arbeitsbandinhalte $\cdot$
Kopfpos. auf den Arbeitsbändern

$= q \cdot (n+2) \cdot (r^{s(n)})^k \cdot (s(n))^k$

(Blanks)

$= 2^{O(s(n))}$, falls $\underbrace{s(n) \geq \log n}_{s(n) \neq o(\log n) \Rightarrow \text{sublineare Zeit}}$ #

Def: Eine Fkt $f: \mathbb{N} \to \mathbb{N}$ heißt raumkonstruierbar, falls es eine det. TM M gibt, die bei Eingabe x genau Platzbedarf $f(|x|)$ hat.

Bem: Alle „üblichen" Fkten wie n, n^k $(k \in \mathbb{N})$, $\log_2 n$, $n \cdot \log_2 n$ 2^n sind raumkonstruierbar.

Satz: Sei $s(n) \gg \log n$ raumkonstruierbar, dann gilt

$$NSPACE(s(n)) \subseteq DTIME(2^{O(s(n))})$$

Bem: Tiefensuche klappt hier nicht, da der Berechnungsbaum eine Tiefe von bis zu $2^{O(s(n))}$ hat
$\Rightarrow$ Zeitbedarf: $2^{2^{O(s(n))}}$

Beweis: Sei M eine nicht det. TM mit Platzbedarf s und M' eine TM die die Raumkonstruierbarkeit garantiert

Konstruiere ein det. TM N die auf Eingabe w, $n = |w|$ wie folgt arbeitet:

i) Markiere $s(n)$ Bandzellen auf Arbeitsband 1 via Simulation von M'

ii) Schreibe alle Konfigurationen von M auf Band 2

iii) Markiere Startkonfig rot

iv) solange es noch rote Konfigurationen gibt wiederhole v) und vi)

v) Sei K die erste rote Konfiguration. Markiere K grün

vi) Markiere alle unmarkierten Folgekonfigs von K (erreichbar in einem Schritt) rot

vii) Falls eine akz. Endkonfig markiert ist, so akzeptiere, sonst lehne ab.

Zeitbedarf von N:

Die Anzahl der Schleifendurchläufe iv) - vi) ist durch die Anzahl der Konfigs beschränkt, da in jedem Schritt eine Konfig grün gefärbt wird

$\Rightarrow$ # Schleifendurchläufe $\leq 2^{O(s(n))}$

Schritt	Zeitbedarf	Grund
i)	$2^{O(s(n))}$	$DSPACE(s) \subseteq DTIME(2^{O(s)})$
ii)	$2^{O(s(n))} \cdot O(s(n))$	# Konfigs · Schritte pro Konfig
iii)	$O(s(n))$	alle Zeichen
iv)	$2^{O(s(n))} \cdot O(s(n))$	# Konfigs · Länge Konfig
v)	$2^{O(s(n))} \cdot O(s(n))$	# Konfigs · Test auf rot
vi)	$2^{O(s(n))} \cdot O(s(n))$	Alle Konfigs durchlaufen · Umfärben
vii)	$2^{O(s(n))} \cdot O(s(n))$	Alle Konfigs durchlaufen · Test auf akz

$\Rightarrow$ Zeitbedarf $2^{O(s(n))} \cdot 2^{O(s(n))} \cdot O(s(n)) = 2^{O(s(n))}$,

falls $s(n) \geq \log_2(s(n))$ #

Satz (Satz von Savitch): Sei $s(n) \geq \log n$ raumkonstruierbar, dann gilt

$$NSPACE(s(n)) \subseteq DSPACE((s(n))^2).$$

Beweis: Sei M eine nichtdet. TM mit Platzbedarf s und w mit $|w| = n$ die Eingabe von M.

Die folgende rekursive Prozedur testet, ob Konfig k_2 von M mit w aus Konfig k_1 in $\leq 2^t$ Schritten erreichbar ist.

```
bool erreichbar (K1, K2, t)
{
    if ((K1 == K2) || (K2 ist direkter Nachfolger von K1)) {
        return true;
    } else if (t == 0) {
        return false;
    } else {
        for all (Konfigurationen K mit Platzbedarf s) {
            if (erreichbar(K1, K, t-1) && erreichbar(K, K2, t-1)) {
                return true;
            }
        }
    }
    return false.
```

„teile und herrsche"

Sei nun K_0 die Startkonfiguration von M und $d \in \mathbb{N}$, sodass $2^{d \cdot s(n)}$ der Raumbedarf von M.

Definiere det. TM N wie folgt:

Eingabe: w, $n = |w|$

```
        for all (akz. Konfigurationen K von M mit Eingabe w
                 mit Platzbedarf s(n)) {
            if (erreichbar(K0, K, d·s(n)) {
                return true;
            }
        }
        return false.
```

Analyse des Platzbedarfs von N:

- N muß die Variablen k, k_1, k_2 und t speichern mit Platzbedarf $O(s(n))$
- Rekursionstiefe beträgt $d \cdot s(n) = O(s(n))$

$\Rightarrow$ gesamter Platzbedarf: $O((s(n))^2)$ #

Nun sollen spezielle Komplexitätsklassen definiert werden:

Def:

- $L =_{def} DSPACE(\log n)$
- $NL =_{def} NSPACE(\log n)$
- $P =_{def} DTIME(n^{O(1)})$
- $NP =_{def} NTIME(n^{O(1)})$
- $PSPACE =_{def} DSPACE(n^{O(1)}) = NSPACE(n^{O(1)})$ ← Satz v. Savitch
- $EXP =_{def} DTIME(2^{n^{O(1)}})$
- $NEXP =_{def} NTIME(2^{n^{O(1)}})$

Damit ergibt sich das folgende Inklusionsdiagramm:

EXP
|
PSPACE
|
NP
|
P
|
NL
|
L

Ausgewählte Kapitel der Theoretischen Informatik
Sommersemester 2010

Fachhochschule Wiesbaden
Prof. Dr. Steffen Reith

1 Die Gödelisierung von Turingmaschinen

Aus technischen Gründen wollen wir Turingmaschinen in natürliche Zahlen umwandeln.

Sei $M = (\Sigma, Z, \delta, z_0, z_1)$ eine 1-Band TM mit $\Sigma = \{a_0, \ldots, a_k\}$ und $Z = \{z_0, \ldots, z_l\}$, dann kann man die Übergangsfunktion δ wie folgt kodieren:

$$\delta(z_i, a_j) \ni (z_{i'}, a_{j'}, \sigma)$$

wird zu

$$\#\#\text{bin}(i)\#\text{bin}(j)\#\text{bin}(i')\#\text{bin}(j')\#\text{bin}(m),$$

wobei

$$m = \begin{cases} 0, & \text{falls } \sigma = L \\ 1, & \text{falls } \sigma = R \\ 2, & \text{falls } \sigma = N. \end{cases}$$

Dabei ist $\text{bin}(x)$ wieder die Binärkodierung von x. Die Befehle unserer Turingmaschine schreiben wir in einer festgelegten Reihenfolge auf und bekommen so ein Wort über $\{\#, 0, 1\}$. Nun kodieren wir

$$\begin{array}{rcl} 0 & \mapsto & 00, \\ 1 & \mapsto & 01 \text{ und} \\ \# & \mapsto & 11. \end{array}$$

Die so berechnete Zahl nennen wir *Gödelnummer*[1] der Maschine M. Die Konstruktion verdeutlicht, dass für jede beliebige Turingmaschine leicht ihre Gödelnummer berechenbar ist. Ebenfalls einsichtig ist, dass man auch aus einer Gödelnummer leicht wieder die ursprüngliche Turingmaschine gewinnen kann. Weiterhin kann man sich leicht überlegen, dass nicht jede natürliche Zahl in binärer Darstellung eine Turingmaschine beschreibt. Dieses Problem können wir aber relativ leicht reparieren. Sei $\hat{M}$ eine beliebige aber festgelegte Turingmaschine, dann sei

$$M_w =_{\text{def}} \begin{cases} M, & \text{falls } w \text{ die Gödelnummer der Turingmaschine } M \text{ ist} \\ \hat{M}, & \text{sonst.} \end{cases}$$

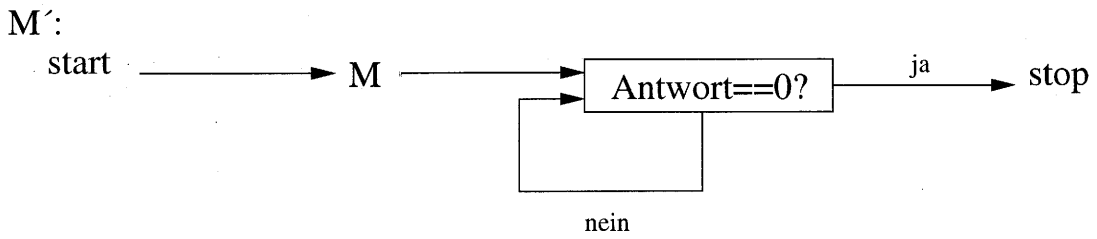
Mit der *Gödelisierung* ($\triangleq$ Kodierung einer Turingmaschine durch eine Gödelnummer) haben wir erreicht, dass eine Turingmaschine die Eingabe einer anderen Turingmaschine M sein kann, d.h. die Turingmaschine M kann Berechnungen durchführen, die bestimmte Eigenschaften dieser Turingmaschine testet. Mit Hilfe dieser Idee definieren wir

Definition 1.1 (Spezielles Halteproblem)

$$H =_{\text{def}} \{w \in \{0,1\}^* \mid M_w \textit{ hält mit der Eingabe } w \textit{ an}\}$$

[1] Gödelnummern sind nach dem österreichisch-ungarischen Mathematiker KURT GÖDEL (*1906 in Brno (Tschechien) - †1978 in Princeton (USA)) benannt, der Gödelnummern in der Logik einführte.

Beweis: Angenommen H wäre entscheidbar, dann wäre die charakteristische Funktion c_H mit Hilfe der TM M berechenbar. Wir bauen nun die Turingmaschine M in die Maschine M' wie folgt um:



Damit ergibt sich die folgende Situation: Die Maschine M' stoppt gdw. M die Antwort 0 ausgibt. Berechnet M die Antwort 1, dann gelangt M' in eine Endlosschleife und hält nicht an. Sei w' die Gödelnummer von M', dann gilt:

M' mit Eingabe w' hält	gdw.	M mit Eingabe w' gibt 0 aus
	gdw.	$c_H(w') = 0$
	gdw.	$w' \notin H$
	gdw.	$M_{w'}$ hält bei Eingabe w' nicht an
	gdw.	M' mit Eingabe w' hält nicht.

Dies ist ein Widerspruch zu der ursprünglichen Annahme, d.h. die Annahme war falsch und es kann keine Turingmaschine geben, die die charakteristische Funktion c_H berechnet. Damit ist das spezielle Halteproblem H nicht entscheidbar. #

1.6. Hierarchiesätze

Ganz natürlich stellen sich die Fragen:

- Gibt es s, s', so dass $\mathrm{DSPACE}(s) \subsetneq \mathrm{DSPACE}(s')$?
  analog: DTIME
- Gibt es eine Klasse $\mathrm{DSPACE}(s)$, so dass alle entscheidbaren Sprachen enthalten sind?

Bem: $f = o(g)$, falls $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ (f wächst wesentlich langsamer als g)

Satz: Sei s raumkonstruierbar, dann gibt es eine Sprache $L \in \mathrm{DSPACE}(s)$, die nicht mit Platzbedarf $o(s)$ entschieden werden kann.

Beweis: (Diagonalisierung)

Für eine TM M sei $\langle M\rangle$ eine Kodierung von M in einem geeigneten Alphabet (z.B. als Binärzahl). $\langle M\rangle$ heißt die Gödelisierung von M.

Sei nun $\hat{M}$ eine TM mit Eingabe ω, $n = |\omega|$, die wie folgt arbeitet:

i) markiere $s(n)$ Bandzellen (s raumkonstruierbar!). spätere Simulation verläßt markierten Bereich $\Rightarrow$ ablehnen
ii) falls ω nicht der Form $\langle M\rangle 1 0^*$ f. eine TM M hat, dann ablehnen
iii) simuliere M auf ω. Benötigt M mehr als $2^{s(n)}$ Schritte, dann ablehnen
iv) falls M akz, dann ablehnen, sonst akz

Sei A die Sprache, die $\hat{M}$ entscheidet. Offensichtlich gilt $A \in DSPACE(s)$

Zeigen: Wenn $s' = o(s)$, dann $A \notin DSPACE(s')$.

Annahme: $A \in DSPACE(s')$, dann ex. eine TM M', die A entscheidet und Platz s' benötigt.

Wähle w von der Form $\langle M' \rangle 10^*$ lang genug, so dass M' auf w Platz bedarf $\leq s(|w|)$ und Zeitbedarf $\leq 2^{s(|w|)}$ hat. „Padding"

Aufgrund dieser Wahl erreicht $\hat{M}$ auf w den Schritt iv, ihrer Rechnung

Also gilt

$w \in A$ gdw M' akz w (Annahme)
gdw $\hat{M}$ akz w nicht (Konstr. v. $\hat{M}$, Schritt iv))
gdw $w \notin A$ ↯

$\Rightarrow$ Annahme falsch $A \notin DSPACE(s')$ #

Folgerung: Sei $s_1 = o(s_2)$, s_2 raumkonstruierbar, dann gilt

$$DSPACE(s_1) \subsetneq DSPACE(s_2)$$

Satz: Sei $t_1(n) = o(t_2(n)/\log t_2(n))$, t_2 „zeitkonstruierbar", dann gilt

$$DTIME(t_1) \subsetneq DTIME(t_2)$$

Beweis: ähnliche Diagonalisierungsmethode #

Ausgewählte Kapitel der Theoretischen Informatik
Sommersemester 2010

Fachhochschule Wiesbaden
Prof. Dr. Steffen Reith

1 Die Klasse P

Es sollten unterschiedlich effiziente Algorithmen bezüglich ihrer Laufzeit verglichen werden. Dabei gibt die nächste Tabelle die reale Laufzeit in Abhängigkeit zur Anzahl der notwendigen Prozessortakte an. Angenommen wir ein 1 MIPS Rechner.

Takt-	Objekte n					
Anzahl	10	20	30	40	50	60
n	0.00001 Sekunden	0.00002 Sekunden	0.00003 Sekunden	0.00004 Sekunden	0.00005 Sekunden	0.00006 Sekunden
n^2	0.0001 Sekunden	0.0004 Sekunden	0.0009 Sekunden	0.0016 Sekunden	0.0025 Sekunden	0.0036 Sekunden
n^3	0.001 Sekunden	0.008 Sekunden	0.027 Sekunden	0.064 Sekunden	0.125 Sekunden	0.216 Sekunden
n^5	0.1 Sekunden	3,2 Sekunden	24.3 Sekunden	1.7 Minuten	5.2 Minuten	13.0 Minuten
2^n	0.001 Sekunden	1 Sekunde	17.9 Minuten	12.7 Tage	35.7 Jahre	366 Jahrhunderte
3^n	0.059 Sekunden	58 Minuten	6.5 Jahre	3855 Jahrhunderte	$2 \cdot 10^8$ Jahrhunderte	$1.3 \cdot 10^{13}$ Jahrhunderte

Es zeigt sich, dass Probleme, die in polynomieller Laufzeit lösbar sind, auch praktisch lösbar sind. Probleme, die lediglich Algorithmen mit exponentieller Laufzeit besitzen, sind hingegen praktisch von (derzeitigen) Computern nicht lösbar (auch bei mehr Hardwareeinsatz). Ähnlich kann man die Eingabelänge angeben, die eine schnellere Computergeneration gerade noch in einer Stunde bewältigen kann:

Laufzeit	mit heutigen Computer	mit 100mal schnelleren Computer	mit 1000mal schnellern Computer
n	n_1	$100n_1$	$1000n_1$
n^3	n_2	$4.64n_2$	$10n_2$
n^5	n_3	$2.5n_3$	$3.98n_3$
2^n	n_4	$n_4 + 6.64$	$n_4 + 9.97$
3^n	n_5	$n_5 + 4.19$	$n_5 + 6.29$

Folgerung: Es gilt $NL \subsetneq PSPACE$ und $P \subsetneq EXP$.

2. Das $P \stackrel{?}{=} NP$ Problem

2.1. Die Klasse P

Bekannte Algorithmen mit polynomieller Laufzeit sind „effizient", da die auftretenden Polynome (sehr oft) einen kleinen Grad haben.

Andererseits: Viele Probleme lassen sich (scheinbar) nur durch „Probieren" lösen.

Deshalb: $P = \bigcup_{k \geq 1} DTIME(n^k)$ ist die Klasse der effizient lösbaren Probleme.

Bem: Die Klasse P ist sogar maschinenunabhängig, d.h. Polynomialzeit auf 1-Band TMs, multi-Band TM, C, RAMs, etc führt immer zur gleichen Klasse von Problemen.

Sogar die Kodierung von Problemen spielt keine Rolle, da „vernünftige" Kodierungen leicht ineinander umgewandelt werden können. ⇒ grobe Algorithmenbeschreibungen reichen uns aus!

Bsp: $PATH = \{(G, s, t) \mid G$ ist ein gerichteter Graph mit Pfad von s nach $t\}$

$PATH \in P$, da

Eingabe: (G, s, t)

i) markiere s
ii) wiederhole iii) solange noch Knoten markiert werden
iii) für jede Kante (u,v), u markiert und v unmarkiert markiere v
iv) falls t markiert akz sonst ablehnen

Laufzeit: da ii) u. iii) max. sooft durchlaufen wird, wie die Anzahl von Knoten von G und i) bzw iv), nur einmal gibt es ein Polynom, das die Laufzeit beschränkt.

2.2. Die Klasse NP

Fakt: Für viele (praktisch relevante) kennt man nur Lösungsverfahren die alle potentiellen Möglichkeiten durchprobieren
⇒ exponentielle Zeit
Viele dieser Probleme können in nichtdet. polynomieller Zeit (vgl. NP) gelöst werden.
Es zeigt sich:

P = „effizient lösbare Probleme"
NP = „effizient überprüfbare Probleme"

Def: Eine Sprache A heißt effizient überprüfbar, falls es einen Algorithmus V gibt, sodass

$w \in A$ gdw es gibt ein y, sodass $\langle w, y \rangle$ von V akz wird. Die Laufzeit ist durch $p(|w|)$, p Polynom, beschränkt.

y heißt Zeuge, Zertifikat oder Beweis für $w \in A$, wenn V $\langle w, y \rangle$ akzeptiert.

Satz: Eine Sprache ist in NP gdw sie polynomiell überprüfbar ist.

Beweis:

„$\Rightarrow$": Sei $A = \underbrace{L(M)}_{\text{von M akz. Sprache}}$ für die $\underbrace{\text{NTM } M}_{\text{nichtdet. TM}}$.

Idee zur Konstruktion der Verifizierers V:

- Ein Zertifikat für $w \in A$ ist ein Pfad im Berechnungsbaum, der zu einer akz. Endkonfiguration führt.
- V simuliert die Arbeitsweise von M auf einem gegebenen Pfad.

Eingabe: $\langle w, y = y_1 y_2 y_3 \ldots \rangle$

Simuliere M bei Eingabe w, wobei y die Beschreibung der nichtdet. Wahlmöglichkeiten ist

D.h. $y_i = 0 \Rightarrow$ Im Schritt i wähle 1. nichtdet. Möglichkeit

$y_i = 1 \Rightarrow$ —"— 2. —"—

⋮

evtl. mehr Möglichkeiten bei höheren Verzweigungsgrad

falls die Simulation akzeptiert : akzeptieren

sonst : ablehnen

„$\Leftarrow$" Sei A polynomiell-überprüfbar via V, wobei V eine det. TM mit Laufzeit n^k, $k \gg 0$ bei Eingaben $\langle w, y \rangle$, $|w| = n$.

Nun wird eine nichtdet. TM M konstruiert, die A akz und in Polynomialzeit arbeitet, indem sie ein Zertifikat rät.

Eingabe: w, $|w| = n$

for $(i = 1$ to $n^k)$ {

/* nicht det. Wahl */

$x_i = 0$; | $x_i = 1$;

}

Simuliere V auf Eingabe $\langle w, x_1 x_2 \ldots \rangle$;

falls V akzeptiert : akzeptiere

sonst : lehne ab

#

Def: Der Graph $(V, V \times V)$ heißt vollständiger Graph mit k Knoten, wenn $\#V = k$.

Bsp 1: $CLIQUE = \{\langle G, k\rangle \mid G$ ist ein ungerichteter Graph, der als Teilgraph einen vollständigen Graph mit k Knoten enthält$\}$

$CLIQUE \in NP$, da ein polynomiell beschränkter Verifizierer V ex

Eingabe: $\langle\langle G, k\rangle, X\rangle$

überprüfe, ob X eine Menge v. Knoten in G;
überprüfe, ob $\#X \geq k$; /* Anzahl d. Knoten checken */
überprüfe, ob jedes Knotenpaar aus $X \times X$ in G durch eine Kante verbunden ist;
falls ja akz, sonst abl.

Def: Sei $G = (V, E)$ mit $V = \{v_1, v_2, \ldots, v_k\}$. Ein Hamiltonscher Pfad von s nach t ist eine Folge von Kanten $(v_{i_1}, v_{i_2}), (v_{i_2}, v_{i_3}), \ldots, (v_{i_{n-1}}, v_{i_n})$, wobei $v_{i_j} \neq v_{i_k}$, wenn $j \neq k$, $s = v_{i_1}$ und $t = v_{i_n}$. („jeder Knoten wird genau einmal besucht")

Bsp 2: $HAMPATH = \{\langle G, s, t\rangle \mid$ es gibt in G einen Hamiltonschen Pfad von s nach $t\}$

$HAMPATH \in NP$

Eingabe: $\langle\langle G, s, t\rangle, X\rangle$

überprüfe, ob X ein Pfad in G von s nach t modelt
und ob jeder Knoten genau einmal enthalten ist

falls ja akz, sonst abl.

Bsp 3: $SOS = \{ \langle \{x_1, \dots, x_n\}, t \rangle \mid x_1, \dots, x_n, t \in \mathbb{N}$ und es ex. $l \in \mathbb{N}$, so dass $\{y_1, \dots, y_l\} \subseteq \{x_1, \dots, x_n\}$ mit $\sum_{i=1}^{l} y_i = t \}$

„Sum of Subset"

SOS ∈ NP, da eine geeignete Teilmenge von Summanden ein Zertifikat ist

Bsp 4: $SAT = \{ \langle H \rangle \mid H$ ist eine erfüllbare aussagenlogische Formel$\}$

SAT ∈ NP, da eine erfüllende Belegung ist ein Zertifikat

Frage: Gilt für jedes A ∈ NP auch A ∈ P?

Es sind Probleme aus NP bekannt, von denen man überzeugt ist, dass sie nicht zu P gehören.

Es zeigt sich sogar, dass die folgenden Aussagen gleichwertig sind

i) CLIQUE ∈ P
ii) HAMPATH ∈ P
iii) SOS ∈ P
iv) SAT ∈ P
v) P = NP

D.h. CLIQUE, SOS, HAMPATH und SAT sind „schwierigste" Probleme in NP.

2.2.1 NP-Vollständigkeit

Def: Seien $A \subseteq \Sigma^*$ und $B \subseteq \Delta^*$ Sprachen. A heißt auf B <u>in Polynomialzeit reduzierbar</u>, falls es ein $f: \Sigma^* \to \Delta^*$ gibt mit

- es gibt eine TM die in Polynomialzeit arbeitet und f berechnet, d.h. $f \in FP$ (Funktionsklasse!)
- $f.a.$ $x \in \Sigma^*$ gilt $x \in A$ gdw $f(x) \in B$

In Zeichen: $A \leq_m^p B$

Die Reduktion wird auch Karp-Reduzierbarkeit oder many-one Reduzierbarkeit genannt.

Bem:
- one-one $\hat{=}$ injektiv
- many-one $\hat{=}$ nicht notwendigerweise injektiv

Lemma: Wenn $A \leq_m^p B$ gilt, dann

i) Ist $B \in NP$, dann $A \in NP$

ii) Ist $B \in P$, dann $A \in P$

„A ist nicht schwerer als B"

Beweis:

i) Sei $A \leq_m^P B$ via $f \in FP$ und nichtdet. TM M die B akzeptiert.

Eine nichtdet. Polynomialzeit TM für A:

Eingabe: x
berechne $y = f(x)$;
simuliere M mit Eingabe y;
akz wenn Simulation akz., sonst lehne ab.

ii) analog (deterministisch) #

Bem: Reduktionen sollen reflexiv und transitiv sein (Quasiordnung).

Proposition: $\leq_m^P$ ist reflexiv und transitiv

Beweis:
- reflexiv: vermöge der Identitätsfkt
- transitiv: Sei $A \leq_m^P B$ via f_1 und $B \leq_m^P C$ via f_2, dann gilt $A \leq_m^P C$ via $f_2 \circ f_1 \in FP$. #

Intuitiv: $A \leq_m^P B$ bedeutet, dass A nicht schwerer als B ist.
Ziel: Definiere und untersuche schwerste Probleme in NP

Def:
i) Eine Sprache B ist NP-hart, falls für alle $A \in NP$ gilt $A \leq_m^P B$
ii) Eine Sprache B ist NP-vollständig, falls B NP-hart ist und $B \in NP$ gilt.

Proposition: Sei B NP-vollständig, dann gilt

$$B \in P \text{ gdw } P = NP$$

Beweis:

„$\Rightarrow$" Sei $B \in P$ und $A \in NP$. Da $A \leq_m^p B$ gilt mit letzter Proposition $A \in P$ und damit $NP \subseteq P$. Sowieso $P \subseteq NP$, d.h. $P = NP$.

„$\Leftarrow$" Wenn $P = NP$, dann auch $B \in P$. #

Frage: Um die NP-Vollständigkeit eines Problems zu zeigen müssen unendlich viele Reduktionen untersucht werden. $\Rightarrow$ geht das eigentlich?

Proposition: Sei B NP-vollständig und $C \in NP$. Falls $B \leq_m^p C$, so ist auch C NP-vollständig.

Beweis: Da für alle $A \in NP$ gilt $A \leq_m^p B$ und $B \leq_m^p C$, ergibt sich mit der Transitivität auch $A \leq_m^p C$.
Da $C \in NP$ ist C NP-vollständig. #

Ziel: Wir müssen ein NP-vollständiges Problem finden, d.h. wir brauchen eine generische Reduktion die jedes Problem aus NP in dieses Problem überführt.

2.2.2 Der Satz von Cook / Levin

Satz: (S. Cook, 1973 / L. Levin, 1973): SAT ist NP-vollständig.

Beweis: Wissen schon: $SAT \in NP$

$\Rightarrow$ Zeige, dass für alle $A \in NP$ gilt $A \leq_m^p SAT$

Sei M eine nicht det. 1-Band TM, die A in Zeit n^k akzeptiert, wobei $M = (Q, \Sigma, \delta, q_0, F)$.
Sei $w = x_1 \dots x_n$, $|w| = n$.

Ein Tableau T von M mit Eingabe w ist eine Tabelle mit n^k Zeilen und $2n^k + 4$ Spalten, wobei

- jede Zeile beschreibt eine Konfiguration von M: Ist M im Zustand q mit Bandinhalt uv und steht der Kopf auf dem ersten Zeichen von v, dann wird dies durch den String $\# u q v \#$ notiert.
  u und v werden links bzw rechts mit Leersymbolen aufgefüllt, sodass immer $|uv| = 2n^k + 1$ gilt.

- Die erste Zeile entspricht der Startkonfiguration

$$\# \underbrace{\square\square\dots\square}_{n^k} q_0 \underbrace{x_1 x_2 \dots x_n \square\square\dots\square}_{n^k+1} \#$$

q_0 ist der Startzustand

- Jede Zeile $i+1$ von T geht aus der Zeile i durch einen Schritt von M hervor.
- Ein Tableau heißt <u>akzeptierend</u>, wenn es eine Zeile enthält, die einer akzeptierenden Konfiguration entspricht.

Also: $w \in A$ gdw es gibt ein akzeptierendes Tableau von M bei Eingabe M

Nun soll eine Formel H konstruiert werden, so dass $w \in A$ gdw $H \in SAT$ ($\Rightarrow$ Variablen mit Hilfe d. Tableaus kodieren)

Genauer: Für $1 \le i \le n^k$, $1 \le j \le 2n^k + 4$ und $s \in Q \cup \Sigma \cup \{\#\}$ gibt es eine Variable $x_{i,j,s}$.

$x_{i,j,s} = 1$ gdw in Zeile i und Spalte j steht s

Kurzschreibweise $Zelle[i,j] = s$

Die zu konstruierende Formel H ist erfüllbar durch eine Belegung I gdw das Tableau zu M bei Eingabe von w mit einer Zeile akzeptiert, die I entspricht.

H hat die Form $H = H_0 \wedge H_s \wedge H_a \wedge H_t$

Teilformel H_0: „Genau eine Variable $x_{i,j,s}$ ist auf 1 gesetzt"

$Zelle[i,j]$ ist eindeutig

$$H_0 = \bigwedge_{\substack{1 \le i \le n^k \\ 1 \le j \le 2n^k+4}} \left[\left(\bigvee_{s \in Q \cup \Sigma \cup \{\#\}} x_{i,j,s} \right) \wedge \left(\bigwedge_{s,t \in Q \cup \Sigma \cup \{\#\}} (\neg x_{i,j,s} \vee \neg x_{i,j,t}) \right) \right]$$

Teilformel H_s: „Die erste Zeile beschreibt die Startkonfiguration"

$$H_s = X_{1,1,\#} \wedge \bigwedge_{2 \le j \le n^k+1} X_{1,j,\square} \wedge X_{1,n^k+2,q_0} \wedge X_{1,n^k+3,x_1} \wedge \dots \wedge X_{1,n^k+2+n,x_n}$$

$$\wedge \bigwedge_{n^k+3+n \le j \le 2n^k+3} X_{1,j,\square} \wedge X_{1,2n^k+4,\#}$$

Teilformel H_a: „Irgendeine Zeile enthält einen akz. Zustand"

$$H_a = \bigvee_{\substack{1 \le i \le n^k \\ 1 \le j \le 2n^k+4 \\ s \in F}} X_{i,j,s}$$

Teilformel H_t: „Zwei aufeinanderfolgende Zeilen entsprechen einem Rechenschritt von M"

Dazu werden „Fenster" der Größe 2×3 im Tableau betrachtet. Ein Fenster heißt <u>legal</u>, wenn es der Überführungsfkt δ nicht widerspricht.

i) Ist $\delta(q,a) \ni (p,b,N)$, so ist

c	q	a
c	p	b

legal für $c \in \Sigma \cup \{\#\}$

ii) Ist $\delta(q,a) \ni (p,b,R)$, so ist

c	q	a
c	b	p

legal für $c \in \Sigma \cup \{\#\}$

iii) Ist $\delta(q,a) \ni (p,b,L)$, so ist

c	q	a
p	c	b

legal für $c \in \Sigma \cup \{\#\}$

Zusätzlich gibt es noch zu i) - iii) gehörige „Randfälle"

i) | d | c | q |
   | d | c | p |
   ,
   | q | a | c |
   | p | b | c |
   ,
   | a | c | d |
   | b | c | d |
   sind legal

für alle $c \in \Sigma$, $d \in \Sigma \cup \{\#\}$

ii) | d | c | q |
    | d | c | b |
    ,
    | q | a | c |
    | b | p | c |
    ,
    | a | c | d |
    | p | c | d |
    sind legal

für alle $c \in \Sigma$, $d \in \Sigma \cup \{\#\}$

iii) | d | c | q |
     | d | p | c |
     ,
     | q | a | d |
     | c | b | d |
     ,
     | e | d | c |
     | e | d | p |
     sind legal

für alle $c, d \in \Sigma$, $e \in \Sigma \cup \{\#\}$

Außerdem ist | a | b | c |
             | a | b | c |
legal für $a, b, c \in \Sigma \cup \{\#\}$

$H_t = \bigwedge_{\substack{1 \le i \le n^k - 1 \\ 1 \le j \le 2n^k + 2}}$ das Fenster, dessen linke obere Zelle die Koordinaten $[i,j]$ hat ist legal

$= \bigwedge_{\substack{1 \le i \le n^k - 1 \\ 1 \le j \le 2n^k + 2}} \bigvee_{a_1, \dots, a_6 \in \Sigma \cup Q \cup \{\#\}} (X_{i,j,a_1} \wedge X_{i,j+1,a_2} \wedge X_{i,j+2,a_3} \wedge$

| a_1 | a_2 | a_3 |
| a_4 | a_5 | a_6 |
ist legal

$\wedge X_{i+1,j,a_4} \wedge X_{i+1,j+1,a_5} \wedge X_{i+1,j+2,a_6})$

$\Rightarrow$ Tableau T besteht nur aus legalen Fenstern gdw alle Konfigurationen in T entsprechen einem möglichen Rechenweg von M.

Damit: $w \in A$ gdw $H \in SAT$

Die Tranformation kann in polynomieller Zeit berechnet werden (z.B. $|H| \leq p(|w|)$, p Polynom)

Also: $A \leq_m^p SAT$ und somit SAT NP-vollständig. #

2.2.3 Weitere NP-vollständige Probleme

Def: $3SAT =_{def} \{H \mid H$ ist erfüllbare KNF-Formel, wobei jede Klausel genau 3 Literale enthält$\}$

Satz: 3SAT ist NP-vollständig

Beweis: Da $3SAT \subset SAT \in NP$ ist nur noch die NP-Härte zu zeigen.

Beo: Die im letzten Beweis konstruierte Formel ist (fast) in KNF.

$\Rightarrow$ Alle Klauseln in solche mit 3 Literalen wandeln

Fall # Literale < 3:

„Auffüllen" mit $K \vee (y \wedge \neg y) \equiv (K \vee y) \wedge (K \vee \neg y)$

Fall # Literale > 3:

Klauseln wie folgt aufbrechen:

Sei $K = (L_1 \vee L_2 \vee \dots \vee L_m)$, $m > 3$ eine Klausel, dann werden neue Variablen $z_1, \dots, z_{m-3}$ eingeführt und K wird ersetzt durch $K' = (L_1 \vee L_2 \vee z_1) \wedge (\neg z_1 \vee L_3 \vee z_2) \wedge (\neg z_2 \vee L_4 \vee z_3) \wedge \dots \wedge (\neg z_{m-3} \vee L_{m-1} \vee L_m)$. Die so erzeugte Formel ist nicht unbedingt logisch äquivalent, aber die ursprüngliche Formel ist erfüllbar gdw. die neue Formel ist erfüllbar. $\Rightarrow$ 3SAT ist NP vollständig. #

Satz: CLIQUE ist NP-vollständig.

Beweis: Wissen schon: $CLIQUE \in NP$

Zeigen: $3SAT \leq_m^P CLIQUE$

Sei $H = (L_{1,1} \vee L_{1,2} \vee L_{1,3}) \wedge (L_{2,1} \vee L_{2,2} \vee L_{2,3}) \wedge \dots \wedge (L_{k,1} \vee L_{k,2} \vee L_{k,3})$

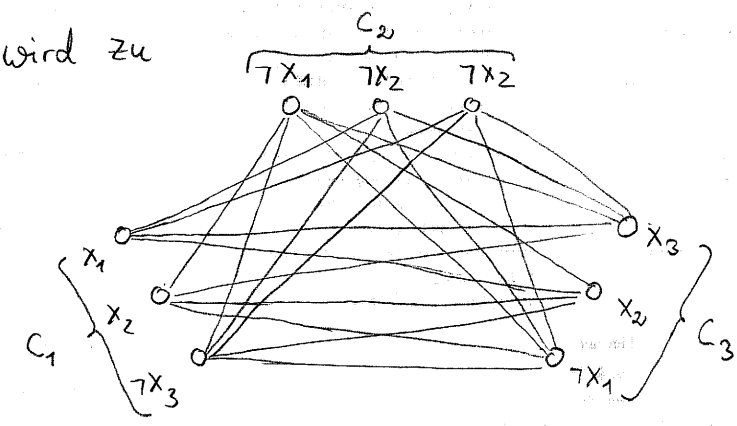
Nun wird ein ungerichteter Graph $G = (V, E)$ wie folgt definiert:

$$V = \{ L_{i,j} \mid 1 \leq i \leq k,\ 1 \leq j \leq 3 \}$$

$$E = \{ (L_{i,j}, L_{i',j'}) \mid i \neq i' \text{ und } L_{i,j} \neq \neg L_{i',j'} \}$$, d.h. $L_{i,j}$ und $L_{i',j'}$ sind nicht konträr.

Bsp: $H' = \overbrace{(x_1 \vee x_2 \vee \neg x_3)}^{C_1} \wedge \overbrace{(\neg x_1 \vee \neg x_2 \vee \neg x_2)}^{C_2} \wedge \overbrace{(\neg x_1 \vee x_2 \vee x_3)}^{C_3}$

wird zu



Beh: H erfüllbar gdw G hat eine Clique der Größe K.

„⇒" Sei H erfüllbar vermöge $I \models H$.

Definiere $V' \subseteq V$ durch: für $1 \leq i \leq K$ enthält V' einen Knoten $L_{i,j}$ mit $I \models L_{i,j}$

Ein solcher Knoten ex., da $I \models H$, also $I \models L_{i,1} \vee L_{i,2} \vee L_{i,3}$.
Falls für ein i mehrere passende j ex., so wähle genau ein beliebiges solches j aus.

Dann gilt

i) $\# V' = K$

ii) V' ist eine Clique, denn für $L_{i,j}, L_{i',j'} \in V'$ gilt $i \neq i'$ (da nur ein Literal pro Klausel gewählt wurde) und $L_{i,j}, L_{i',j'}$ sind nicht konträr, da I beide erfüllt
$\Rightarrow (L_{i,j}, L_{i',j'}) \in E$

Also ist $(G, k) \in$ CLIQUE

„⇐" Sei $V' \subseteq V$ eine Clique mit $|V'| \geq k$.

Falls $L_{i,j}, L_{i',j} \in V'$, dann

i) $i \neq i'$

ii) $L_{i,j}$ und $L_{i',j'}$ sind nicht konträr, da $(L_{i,j}, L_{i',j'}) \in E$.

Nun wird die Belegung I wie folgt definiert $I \models L_{i,j}$ gdw $L_{i,j} \in V'$. Es gilt

- I ist konsistent wegen ii)
- $I \models H$, da $I \models L_{i,1} \vee L_{i,2} \vee L_{i,3}$ für $1 \leq i \leq k$, wegen i)

Also $H \in 3SAT$

Die Abbildung $H \mapsto (G, k)$ ist polynomialzeit berechenbar, d.h. $3SAT \leq_m^P CLIQUE$. #

<u>Satz:</u> HAMPATH ist NP-vollständig

<u>Beweis:</u> siehe Sipser. #

<u>Satz:</u> SOS ist NP-vollständig

<u>Beweis:</u> siehe Sipser #

} „Selbststudium"

2.3. Approximationsalgorithmen

Frage: Wie geht man mit NP-vollständigen Problemen in der Praxis um?

i) Ein spezielleres Problem suchen, dass für die praktische Anwendung ausreicht und evtl. nicht NP-vollständig ist.

ii) Heuristiken (z.B. greedy, evolutionäre Algorithmen, simulated annealing, Ameisenalgorithmen, ...)
Problem: Güte der Lsgen kann nicht garantiert werden.
$\Rightarrow$ „Besser als nichts Ansatz"

iii) Approximationsalgorithmen
Vorteil: Die Güte der Lsgen ist garantiert!

2.3.1. Optimierungsprobleme

Def: Ein Optimierungsproblem $\mathcal{P}$ ist ein 5-Tupel

$$\mathcal{P} = (\mathcal{I}, \mathcal{Y}, Sol, m, goal)$$

wobei
- $\mathcal{I}$ ist die Menge der Instanzen von $\mathcal{P}$
- $\mathcal{Y}$ ist die Menge der möglichen Lösungen
- $Sol \subseteq \mathcal{I} \times \mathcal{Y}$
- $m: \mathcal{I} \times \mathcal{Y} \to \mathbb{N}$ ist die Maßfunktion, dabei ist $m(x,y)$ der Wert (Maß f. die Kosten) der Lsg y für die Instanz x. m ist total.

- goal $\in \{ \min, \max \}$

Bem: Manchmal schreiben wir auch $\mathcal{I}_\mathcal{B}$, $\mathcal{G}_\mathcal{B}$, $m_\mathcal{B}$ und $goal_\mathcal{B}$, um hervorzuheben, dass Komponenten des Optimierungsproblems $\mathcal{B}$ gemeint sind.

Bsp: Min TSP $= (\mathcal{I}, \mathcal{G}, Sol, m, goal)$

- $\mathcal{I} = \{ \langle (d_{i,j})_{1 \le i,j \le n} \rangle \mid n, d_{i,j} \in \mathbb{N} \}$

  „Menge der Distanzmatrizen"

- $\mathcal{G} = S_n$ „Gruppe der Permutationen von n Elementen"

- Sei $D \in \mathcal{I}$ und $\Pi \in S_n$, sodass $(D, \Pi) \in Sol$, dann ist Π eine der Rundreisen für die Distanzmatrix D.

- $m(D, \Pi)$ ist die Länge der Rundreise Π, d.h.
  $m(D, \Pi) = \sum_{i=1}^{n-1} d_{\Pi(i), \Pi(i+1)} + d_{\Pi(n), \Pi(1)}$

- goal = min

Def: Sei $\mathcal{B} = (\mathcal{I}, \mathcal{G}, Sol, m, goal)$ ein Optimierungsproblem. Die Lsg $y^* \in \mathcal{G}$ heißt optimale Lsg zu x, falls

$$m(x, y^*) = goal \{ m(x,y) \mid (x,y) \in Sol \}$$

Weiterhin

$$Sol^*(x) =_{def} \{ y^* \in \mathcal{G} \mid y^* \text{ ist optimale Lsg von } x \}$$

und

$$m^*(x) =_{def} m(x, y^*) \quad \text{für ein } y^* \in Sol^*(x)$$

Def: Sei $\mathcal{P} = (\mathcal{I}, \mathcal{Y}, Sol, m, goal)$ ein Optimierungsproblem. Das Entscheidungsproblem zu $\mathcal{P}$ ist

$$\mathcal{P}_D = \{(x, k) \mid x \in \mathcal{I}_\mathcal{P},\ k \in \mathbb{N} \text{ und } m^*(x) \leq k, \text{ falls } goal_\mathcal{P} = min,\ m^*(x) \geq k, \text{ falls } goal_\mathcal{P} = max\}$$

Bsp: Das Entscheidungsproblem zu MinTSP ist TSP.

Def: Ein Optimierungsproblem $\mathcal{P} = (\mathcal{I}, \mathcal{Y}, Sol, m, goal)$ gehört zur Klasse NPO, wenn gilt:

i) $\mathcal{I} \in \mathcal{P}$, d.h. die Menge der Instanzen ist in Polynomialzeit entscheidbar

ii) es gibt ein Polynom p, sodass f.a. $x \in \mathcal{I}$ und alle $y \in \mathcal{Y}$ mit $(x, y) \in Sol$ gilt $|y| \leq p(|x|)$

iii) $\{\langle x, y\rangle \mid (x, y) \in Sol\} \in \mathcal{P}$ „effizient überprüfbar"

iv) $m \in FP$, d.h. polynomialzeitberechenbar

Bsp MINTSP $\in$ NPO

Beo: Sei $\mathcal{P} \in$ NPO, dann gilt $\mathcal{P}_D \in NP$

Beweis: Sei $\mathcal{P} = (\mathcal{I}, \mathcal{G}, Sol, m, goal)$ und $\mathcal{P} \in NPO$.

Für den Fall eines Minimisierungsproblems gilt:

Eingabe: $x \in \mathcal{I}_{\mathcal{P}}$, $k \in \mathbb{N}$

rate nichtdet. alle y mit $|y| \leq p(|x|)$, wobei p das Polynom aus obiger Def;

if $(x,y) \in Sol$ then

if $m(x,y) \leq k$ then

akzeptieren;

endif;

endif;

lehne ab.

Analog für Maximierungsprobleme. #

Def: Ein Optimierungsproblem $\mathcal{P} \in NPO$ gehört zur Klasse PO, falls einen det. Polynomialzeitalgorithmus gibt, der zu $x \in \mathcal{I}_{\mathcal{P}}$ ein bel. y berechnet, sodass $y \in Sol^*(x)$.

Satz: $P = NP$ gdw $PO = NPO$

Beweis: „$\Leftarrow$" Ist $NPO = PO$, dann gilt $MinTSP \in PO$, also $TSP \in P$. Deshalb $P = NP$

„$\Rightarrow$" hier nicht. #

Sprechweise: Ein Optimierungsproblem $\mathcal{P}$ heißt NP-hart, wenn jedes Problem aus NP durch Anweisungen der Form „$y \in Sol^*_{\mathcal{P}}(x)$" und „$k = m^*_{\mathcal{P}}(x)$" gelöst werden kann.

(Formaler: Jedes Problem aus NP ist Turing-reduzibel auf $\mathcal{P}$)

Def: Sei $\mathcal{P} = (\mathcal{I}, \mathcal{S}, Sol, m, goal)$ ein Optimierungsproblem. Ein Algorithmus A heißt *Approximationsalgorithmus*, wenn er bei Eingabe $x \in \mathcal{I}$ eine Lösung $A(x) \in \mathcal{S}$ berechnet.

Schreibweise: $m_A(x) = m_{\mathcal{P}}(x, A(x))$

Ab jetzt:
- Approximationsalgorithmen arbeiten immer in Polynomialzeit
- Die Ausgabe eines Approximationsalgorithmus liegt nicht „weit" vom Optimum weg:

$$\left.\begin{array}{ll} \Rightarrow goal = min: & \dfrac{m_A(x)}{m^*(x)} \\ \Rightarrow goal = max: & \dfrac{m^*(x)}{m_A(x)} \end{array}\right\} \text{ nach oben beschränken}$$

Def: Sei $\mathcal{P} = (\mathcal{I}, \mathcal{S}, Sol, m, goal)$ ein Optimierungsproblem, $x \in \mathcal{I}$ und $y \in \mathcal{S}$, sodass $(x,y) \in Sol$.
Die *Performanz(-rate)* von y (bzgl. x) ist dann

$$R_{\mathcal{P}}(x,y) = \begin{cases} \dfrac{m(x,y)}{m^*(x)}, & \text{falls } goal = min \\ \dfrac{m^*(x)}{m(x,y)}, & \text{falls } goal = max \end{cases}$$

$$= \max\left\{ \frac{m(x,y)}{m^*(x)}, \frac{m^*(x)}{m(x,y)} \right\}$$

Klar: $R(x,y) = 1$ gdw $y \in Sol^*(x)$. Für $y \notin Sol^*(x)$ kann $R(x,y)$ beliebig groß werden.

Def: Sei $\mathcal{P} = (\mathcal{I}, \mathcal{S}, Sol, m, goal)$ ein Optimierungsproblem und A ein Approximationsalgorithmus für $\mathcal{P}$. Die Performanz(-rate) von A (bzgl. $\mathcal{P}$) ist

$$R_A(x) =_{df} R_{\mathcal{P}}(x, A(x))$$

Sei $r: \mathbb{N} \to \mathbb{Q}$. A heißt r-Approximationsalgorithmus für $\mathcal{P}$, falls

$$\max_{|x|=n} R_A(x) \leq r(n)$$

Bem: Natürlich kann r auch eine Konstante sein.

2.3.1. Das Problem des Handlungsreisenden

Satz: Falls $P \neq NP$, dann gibt es kein $c \geq 1$, sodass für MinTSP ein c-Approximationsalgorithmus existiert.

Beweis: Annahme: Es existiert ein c-Approximationsalgorithmus A für MinTSP.

Ziel: Benutzen HAMCIRC, das NP-vollständig ist

Hamiltonkreisproblem

$$HAMCIRC = \{ G \mid G = (\{v_1, \dots, v_n\}, E) \text{ Graph und es gibt einen Pfad } (v_{i_1}, v_{i_2}), \dots (v_{i_{n-1}}, v_{i_n}), (v_{i_n}, v_{i_1}), v_{i_j} \neq v_{i_k} \text{ wenn } j \neq k \text{ und } v_{i_1} = v_1 \} \text{ „Kreis der alle Knoten genau einmal enthält“}$$

und zeigen, dass dann $HAMCIRC \in P$

$\Rightarrow$ Widerspruch zur Annahme $P \neq NP$

Algorithmus für HAMCIRC:

Eingabe: $G = (V, E)$ mit $V = \{v_1, \dots, v_n\}$

sei $d_{ij} = \begin{cases} 1, & \text{falls } (v_i, v_j) \in E \\ 1 + n \cdot c, & \text{sonst} \end{cases}$

Simuliere A auf Eingabe $x = \langle d_{ij}, n \rangle$;

akz wenn $m_A(x) = n$.

$G \in$ HAMCIRC: Dann hat die Instanz x von MINTSP eine Lsg mit Maß n und diese ist optimal $m^*(x) = n$.

Jede nicht-optimale Lsg hat Maß

$\underbrace{n-1}_{\text{„gute Kanten“}} + \underbrace{1 + c \cdot n}_{\text{„schlechte Kanten“}} = (c+1) \cdot n$, d.h.

die Performanz einer nicht-optimalen Lsg ist $c+1$ $\Rightarrow$ A berechnet die optimale Lsg und obiger Algorithmus akzeptiert.

$G \notin$ HAMCIRC: $\Rightarrow m^*(x) \geq (c+1) \cdot n$

$\Rightarrow m_A(x) \geq (c+1) \cdot n$, d.h.

obiger Algorithmus lehnt ab.

Da angegebene Algorithmus entscheidet HAMCIRC in polynomieller Zeit $\lightning$ $\Rightarrow$ es gibt keinen c-Approximationsalgorithmus für MINTSP, wenn $P \neq NP$. #

Idee: Schränke MINTSP geeignet ein.

PROBLEM: Min MTSP (metric TSP)

EINGABE: $\langle (d_{i,j})_{1 \le i,j \le n} \rangle$ mit $n, d_{i,j} \in \mathbb{N}$ und f.a. $i,j,k \in \{1,\dots,n\}$ gilt

$d_{i,j} = d_{j,i}$ „Symmetrie"

$d_{i,j} + d_{j,k} \geq d_{i,k}$ „Dreiecksungleichung"

LÖSUNG: $\Pi \in S_n$

Maß: $\sum_{i=1}^{n-1} d_{\Pi(i),\Pi(i+1)} + d_{\Pi(n),\Pi(1)}$

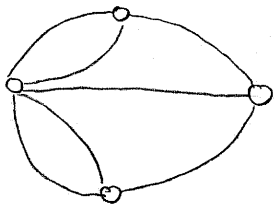
Bem:
- Eine nicht negative Funktion $g(x,y)$, die symmetrisch ist, $g(x,x)=0$ und die Dreiecksungleichung erfüllt, heißt Metrik ($\hat{=}$ „Abstände" zwischen Punkten messen).

  Eine Menge S mit einer dazugehörigen Metrik heißt metrischer Raum.

- Min MTSP$_D$ ist weiterhin NP-vollständig

Frage: Kann man Min MTSP besser approximieren?

Def: Ein Multigraph ist ein Paar $G=(V,E)$, wobei V eine Knotenmenge und E eine Kantenmultimenge, d.h. zwischen zwei Knoten kann mehr als eine Kante verlaufen



„Brücken problem"

Def: Ein <u>Eulerpfad</u> in G ist ein Pfad $v_1, v_2, \dots, v_m$ mit $\{v_i, v_{i+1}\} \in E$, $1 \le i \le m$, wobei jede Kante <u>genau einmal</u> besucht wird (Knoten können bel. oft besucht werden). Ein <u>Eulerkreis</u> ist ein Eulerpfad mit gleichem Start- und Endknoten.

Eulerpfade - und kreise können, wenn existent, in Polynomialzeit berechnet werden.

Def: Ein <u>gewichteter Graph</u> ist ein Graph mit Gewichten auf den Kanten, d.h. ein Tripel $G = (V, E, D)$, wobei (V, E) ein Graph ist und $D = (d_{u,v})_{\{u,v\} \in E} \in \mathbb{N}$ die Gewichte der Kanten.

Def: Ein <u>Spannbaum</u> in einem Graphen G ist ein Teilgraph $G' = (V, E')$ mit $E' \subseteq E$, der ein Baum ist. (ein Spannbaum enthält alle Knoten des ursprünglichen Graphens)

Ein <u>minimaler Spannbaum</u> in einem gewichteten Graphen $G = (V, E, D)$ ist ein Spannbaum $G' = (V, E')$ in (V, E) mit minimalem Gewicht.

Bem:
- Diese Definitionen können analog auch auf Multigraphen übertragen werden.
- Minimale Spannbäume können in Polynomialzeit berechnet werden.

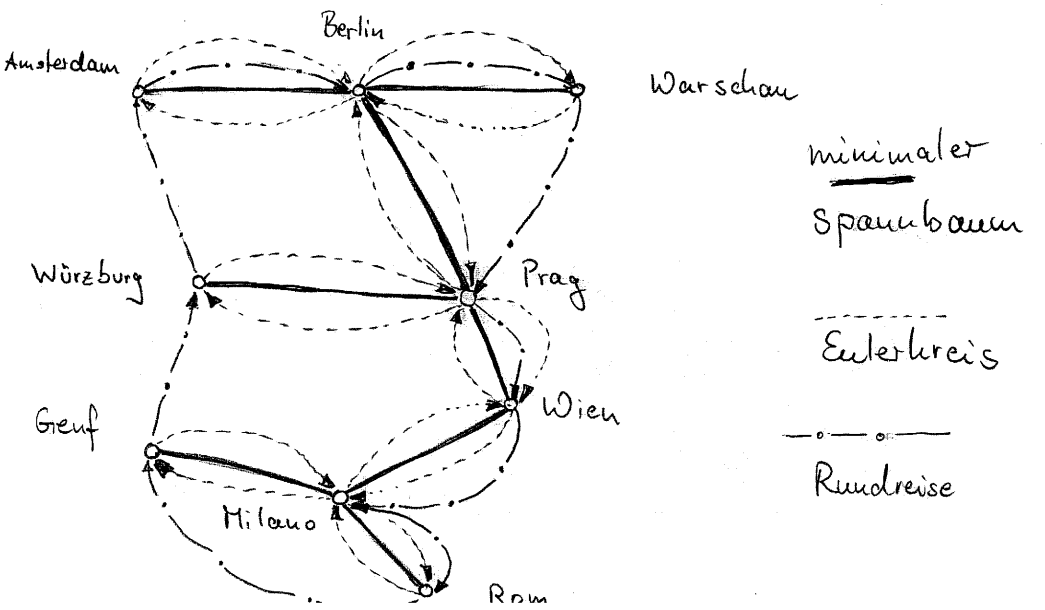
Algorithmus Tree TSP:

Eingabe: $D = (d_{i,j})_{1 \le i,j \le n}$, Instanz von Min MTSP

Sei $G_1 = (V, E, D)$ gewichteter Graph mit
$V = \{1, \dots, n\}$ und $E = V \times V$

berechne minimalen Spannbaum $T = (V, E_T)$ in G_1;
sei $M = (V, E'_T)$ der Multigraph in dem jede
Kante von T genau zweimal vorkommt;
berechne einen Eulerkreis $W = \{v_1, v_2, \dots, v_1\}$ in M;
berechne aus W eine Rundreise wie folgt:
von jeden Knoten nur das erste Vorkommen
belassen alle anderen löschen, also
$(v'_1, v'_2, \dots, v'_n)$, wobei $v'_1 = v_1$ und
v'_i = der Knoten v_k, so dass $\{v_1, \dots, v_{k-1}\}$
$= \{v'_1, \dots, v'_{k-1}\} \subsetneq \{v_1, \dots, v_k\}$
return $(v'_1, \dots, v'_n, v_1)$.

Bsp:



Satz: Tree TSP ist ein 2-Approximationsalgorithmus für Min MTSP

Beweis: Sei $x = D = (d_{i,j})$ eine Instanz von Min MTSP
Jede TSP-Rundreise in D geht in einen Spannbaum über, wenn eine Kante aus ihr entfernt wird.
Also gilt:

Gewicht der Kanten in $T \leq m^*(x)$

Gewicht der Kanten in $M \leq 2 \cdot m^*(x)$

Gewicht der Kanten in der Rundreise $\leq 2 \cdot m^*(x)$, da diese Kanten ja eben eine Abkürzung darstellen.

$\Rightarrow m_{\text{Tree TSP}}(x) \leq 2 \cdot m^*(x)$. #

Eine kleine Modifikation führt zum besten bekannten Algorithmus, der für Min MTSP bekannt ist.

Def: Sei $G = (V, E)$ ein Graph. Eine Kantenmenge $M \subseteq E$ heißt Matching (von G), wenn zwei verschiedene Kanten aus M keinen gemeinsamen Knoten haben, d.h. $(u,v), (u',v') \in M$ und $(u,v) \neq (u',v')$, dann $\{u,v\} \cap \{u',v'\} = \emptyset$

Ein Matching M heißt perfekt (oder vollständig), falls $\#M = \frac{1}{2} \#V$, d.h. jeder Knoten kommt als Endpunkt einer Kante in M vor.

Bem: Perfekte Matchings mit minimalem Gewicht sind in Polynomialzeit berechenbar.

Algorithmus von Christofides:

Eingabe: $D = (d_{i,j})_{1 \le i,j \le n}$, Instanz von Min MTSP

sei $G = (V, E, D)$ der gewichtete Graph und $T = (V, E_T)$ der minimale Spannbaum wie in Tree TSP;

sei $C \subseteq V$ die Menge der Knoten mit ungeradem Grad;

sei $G' = (C, E_C)$ der Teilgraph mit $E_C = E \cap (C \times C)$;

sei H ein perfektes Matching mit minimalem Gewicht in G';
/* $\#C = 2 \cdot k$, da die Summe aller Grade gerade ist, also ex. ein perfektes Matching */

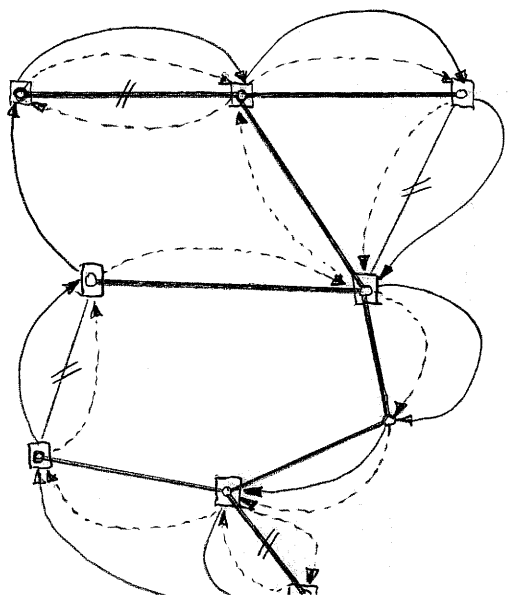
(höhe Grad um eins f. ungerade Knoten)

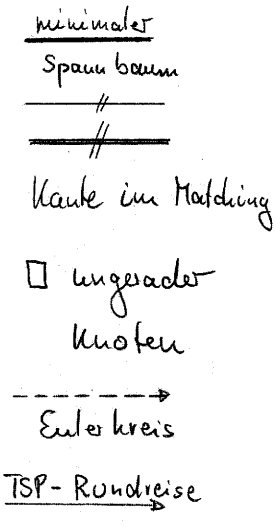
bestimme einen Eulerkreis W im Multigraphen $M = (V, E_T \cup H)$;
/* M enthält nur Knoten mit geradem Grad $\Rightarrow$ Eulerkreis ex. */

gewinne Tour $P = (v_1', \dots, v_n')$ indem in W nur erste Vorkommen von Knoten berücksichtigt werden;

return $(v_1', \dots, v_n', v_1')$.

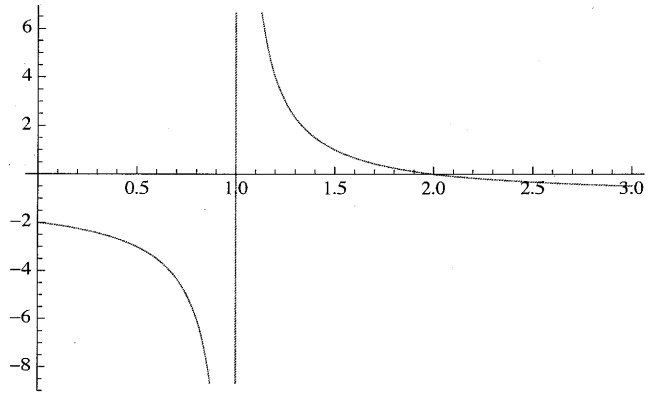
Bsp: (analog zu letztem Beispiel)





In[3]:= `Plot[(2 - r) / (r - 1), {r, 0, 3}]`

Out[3]=



Satz: Der Algorithmus von Christofides ist ein $3/2$-Approximationsalgorithmus für Min MTSP.

Beweis: vgl. Ausiello et.al, Complexity and Approximation #

2.3.2. Das Partitionierungsproblem

PROBLEM: Minimum Partition (Min PART)
INSTANZ: $\langle a_1, \dots, a_n \rangle$, $a_i \in \mathbb{N}^+$
LÖSUNG: Mengen Y_1, Y_2 mit $Y_1 \cup Y_2 = \{1, \dots, n\}$, $Y_1 \cap Y_2 = \emptyset$
MAß: $\max \{ \sum_{i \in Y_1} a_i , \sum_{i \in Y_2} a_i \}$

Bem: Das Problem Min $PART_D$ ist NP-vollständig, also ist Min PART NP-hart.

Algorithmus Part AS:

Eingabe: $\langle a_1, \dots, a_n \rangle$ mit $a_i \in \mathbb{N}^+$, $r > 1$
if $r \geq 2$ then
  return $Y_1 = \{1, \dots, n\}$, $Y_2 = \emptyset$;
else
  sortiere Eingabe; /* o.B.d.A. $a_1 \geq a_2 \geq \dots \geq a_n$ */
  $k_r = \lceil \frac{2-r}{r-1} \rceil$;
  finde optimale Partition Y_1, Y_2 von $\langle a_1, \dots, a_{k_r} \rangle$ durch Ausprobieren aller Lsgen;
  for $j = k_r + 1$ to n do
    if $\sum_{i \in Y_1} a_i \leq \sum_{i \in Y_2} a_i$ then

$Y_1 = Y_1 \cup \{j\}$;
else
$Y_2 = Y_2 \cup \{j\}$;
endif;
endfor;
return (Y_1, Y_2).

Satz: Bei Eingabe einer Instanz $x \in \hat{I}_{\text{MinPART}}$, $X = \langle a_1, \dots, a_n \rangle$ und $r > 1$ liefert PartAS eine Lösung mit Performanzrate $\leq r$.

Beweis: Für $Z \subseteq \{1, \dots, n\}$ sei $\omega(Z) =_{def} \sum_{i \in Z} a_i$
und $X =_{def} \{1, \dots, n\}$

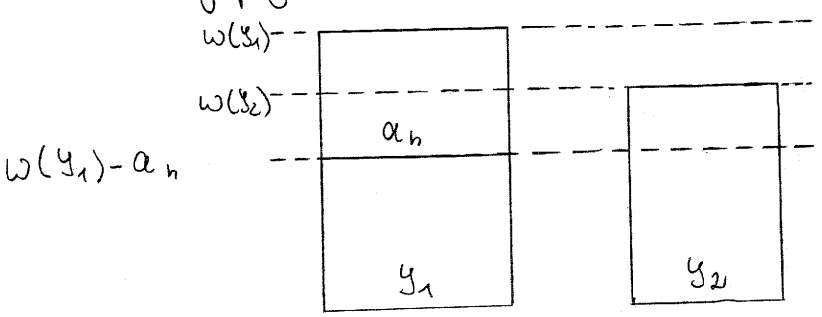
Sei nun $r \geq 2$ und Y_1, Y_2 eine bel. Lsg. zu x. Es gilt:

i) $m(x, (Y_1, Y_2)) \geq \underbrace{\frac{1}{2}\omega(X)}_{\text{optimale Lsg kann nicht kleiner sein}}$

ii) $m(x, (X, \emptyset)) = \omega(X)$

Also $R(x, (X, \emptyset)) \leq 2 \leq r$

Sei nun $r < 2$ und (Y_1, Y_2) die von PartAS berechnete Lsg. O.B.d.A $\omega(Y_1) \geq \omega(Y_2)$.
Sei $h \in X$ die Zahl, die als letztes zu Y_1 hinzugefügt wurde:



Es gilt: $\omega(y_1) - a_n \leq \omega(y_2)$.

Also: $2\omega(y_1) - a_n \leq \omega(y_1) + \omega(y_2) = \omega(x)$

$\Leftrightarrow \quad \omega(y_1) \leq \frac{\omega(x) + a_n}{2} \qquad (*)$

Fall $h \leq k_r$: d.h. h wurde zu Y_1 bei der optimalen Aufteilung von $a_1, \dots, a_{k_r}$ hinzugefügt.

$\Rightarrow$ in der for-Schleife wurde nur y_2 erweitert, denn h wurde zuletzt zu y_1 hinzugefügt

Also ist die Lsg (Y_1, Y_2) optimal (sie war schon optimal für $\langle a_1, \dots, a_{k_r} \rangle$

Fall $h > k_r$: Es gilt $a_j \geq a_{k_r}$ für $1 \leq j \leq k_r$, da die Eingabe absteigend sortiert wurde.

Also $\omega(x) \geq (k_r + 1) \cdot a_n$ und damit

$$\frac{a_n}{\omega(x)} \leq \frac{1}{k_r + 1} \qquad (**)$$

Weiterhin gilt $\underbrace{\omega(Y_1) \geq \overbrace{\frac{1}{2}\omega(x)}^{2\omega(y_2) \leq \omega(x)} \geq \omega(y_2)}_{\omega(y_1) + \omega(y_2) = \omega(x)}$, also

$\Rightarrow 2\omega(y_1) \geq \omega(x)$, da $\omega(y_1) \geq \omega(y_2)$

ist $m^*(x) \geq \frac{1}{2}\omega(x) \qquad (***)$

Zusammen:

$$R_{PartAS}(x) = \frac{m(x, (y_1, y_2))}{m^*(x)} = \frac{\omega(y_1)}{m^*(x)}$$

$$\overset{(***)}{\leq} \frac{\omega(y_1)}{\frac{1}{2}\omega(x)} \overset{(*)}{\leq} \frac{\omega(x) + a_n}{2 \cdot \frac{1}{2}\omega(x)}$$

$$= 1 + \frac{a_n}{\omega(x)} \overset{(**)}{\leq} 1 + \frac{1}{k_r + 1}$$

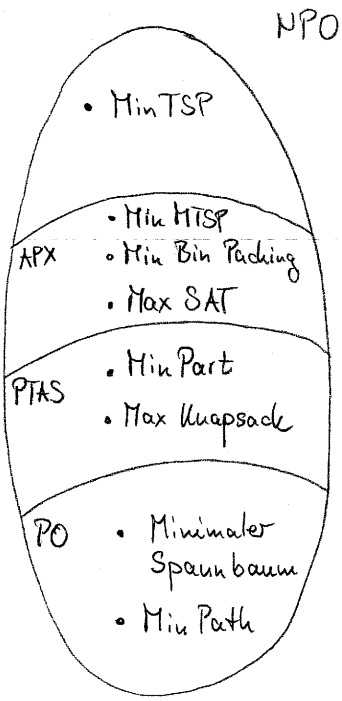
$$\leq 1 + \frac{1}{\frac{2-r}{r-1} + 1} = r. \qquad \#$$

Damit gibt es für jede beliebige Zahl $r > 1$ einen (Polynomialzeit-) Approximationsalgorithmus für Min PART mit Performanzrate $\leq r$.

Alle Probleme mit dieser Eigenschaft werden zur Klasse PTAS („polynomial time approximation scheme") zusammengefasst.

APX ist die Klasse aller Probleme mit einem c-Approximationsalgorithmus für $c > 1$, $c \in \mathbb{Q}$

Sei nun $P \neq NP$ angenommen, dann ergibt sich folgendes Bild



3. Berechenbarkeitstheorie

Literatur: Hartley Rogers, Theory of Recursive Functions and Effective Computability

3.1. Das Halteproblem

Für diesen Abschnitt halten wir eine bel. Gödelisierung von TMen fest. Sei $\hat{M}$ eine bel. TM, dann

$$M_w =_{def} \begin{cases} M, & \text{falls } w \text{ Gödelnummer von } M \\ \hat{M}, & \text{sonst} \end{cases}$$

Def: Die Menge

$$H =_{def} \{w \mid M_w \text{ hält mit Eingabe } w\}$$

heißt spezielles Halteproblem.

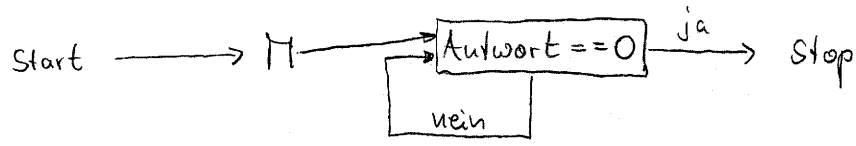
Satz: Das spezielle Halteproblem ist nicht entscheidbar.

Beweis: Angenommen H wäre entscheidbar, dann wäre die charakteristische Fkt

$$c_H(w) = \begin{cases} 1, & \text{falls } w \in H \\ 0, & \text{sonst} \end{cases}$$

berechenbar vermöge M. Sei

M':



mit der Gödelnummer ω', dann

M' hält auf ω' gdw M gibt mit ω' 0 aus
gdw $c_H(\omega') = 0$
gdw $\omega' \notin H$
gdw $M_{\omega'}$ hält auf ω' nicht
gdw M' hält auf ω' nicht ↯

$\Rightarrow$ M kann nicht ex., d.h. H ist nicht entscheidbar. #

Def: Seien $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$ Sprachen. Dann heißt A auf B reduzierbar (kurz: $A \leq_m B$), wenn eine berechenbare Fkt $f: \Sigma^* \to \Gamma^*$ mit $x \in A$ gdw $f(x) \in B$.

Proposition: Gilt $A \leq_m B$ und A unentscheidbar, dann ist auch B unentscheidbar.

Def: $H_1 =_{def} \{ \omega \# x \mid M_\omega$ hält bei Eingabe x an $\}$
(allgemeines Halteproblem)

Satz: H_1 ist nicht entscheidbar.

Beweis: $f(\omega) = \omega \# \omega$ zeigt $H \leq_m H_1$. Die Beh. folgt mit der letzten Proposition. #

Def: $H_0 = \{ w \mid M_w$ hält mit leerem Band $\}$

Satz: H_0 ist nicht entscheidbar.

Beweis: Reduzieren H auf H_0 vermöge f:

Eingabe von f: $w \# x$

berechne Turingprogramm von w;
füge vor das Turingprg. Befehl ein, die x auf das Band schreiben;
berechne Gödelnummer dieser neuen Maschine.

Klar: $w \# x \in H_1$ gdw $f(w \# x) \in H_0$ und damit ist H_0 unentscheidbar. #

Def: Eine Sprache $A \subseteq \Sigma^*$ heißt semi-entscheidbar, wenn die Fkt

$$c'_A(w) =_{def} \begin{cases} 1, & \text{falls } w \in A \\ \text{undef.}, & \text{sonst} \end{cases}$$

berechenbar ist. D.h. eine Maschine, die c'_A berechnet, hält für Eingaben $w \notin A$ nie an.

Satz: Eine Sprache A ist entscheidbar gdw A und $\bar{A}$ semi-entscheidbar sind.

Beweis:

„⇒" Da A entscheidbar ist, ex. eine TM M, die c_A berechnet. Einfügen einer Endlosschleife liefert c'_A und $c'_{\bar{A}}$.

„⇐" Sei M_1 die Maschine für c'_A und M_2 die für $c'_{\bar{A}}$. Dann entscheidet

```
s=1;
While(1) {
  if (M1 stoppt nach s Schritten) return 1;
  if (M2 stoppt nach s Schritten) return 0;
  s=s+1;
endwhile.
```

#

Korollar: $\bar{H}_1$, $\bar{H}_1$ und $\bar{H}_0$ sind nicht semientscheidbar.

Beweis: Die Simulation der gegebenen TMen liefert direkt die Semi-entscheidbarkeit von $\bar{H}$, $\bar{H}_1$ und $\bar{H}_0$.

#

Frage: Kann das Halteproblem vielleicht mit einem anderen Berechnungsmodell gelöst werden?

These von Church: Die durch die formale Definition der Turing-Berechenbarkeit (RAM-Berechenbarkeit, ...) festgelegte Klasse von Funktionen stimmt genau mit der im intuitiven Sinn berechenbaren Fkten überein.

⇒ Auch mit anderen Berechenbarkeitsmodellen bleibt das Halteproblem unentscheidbar.

3.2. Der Satz von Rice

Satz: Sei TM die Menge aller Turing-berechenbaren Fkten und $S \subseteq TM$, wobei $S \neq \emptyset$ und $S \neq TM$.

Dann ist die Menge

$$C(S) =_{def} \{w \mid \text{die von } M_w \text{ berechnete Fkt liegt in } S\}$$

unentscheidbar.

Bsp: Sei $S_1 = \{f \mid f \text{ verdoppelt die Eingabe}\}$, dann ist

$C(S_1)$ unentscheidbar.

D.h. nahezu jede semantische Eigenschaft von Turingmaschinen ist unentscheidbar.

Beweis: Sei Ω die überall undefinierte Fkt, die durch while (1); /* hält nie */ berechnet wird.

Fall $\Omega \in S$: Sei $f \in TM \setminus S$ und Q eine TM, die f berechnet.

Sei nun w eine Gödelnummer, dann arbeitet TM M wie folgt:

angesetzt auf Eingabe y wird y ignoriert;
verhalte dich wie M_w;
falls M_w hält, so verhalte dich wie Q.

M berechnet also folgende Fkt f'

$$f' = \begin{cases} \Omega, & \text{falls } M_w \text{ stoppt auf leerem Band nicht} \\ f, & \text{sonst} \end{cases}$$

Klar: Die Abbildung g, die w die Maschine M zuordnet ist berechenbar.

$w \in H_0 \Rightarrow M_w$ stoppt mit leerem Band
$\Rightarrow M$ berechnet f
$\Rightarrow$ die von $M_{g(w)}$ berechnet Fkt liegt nicht in S
$\Rightarrow g(w) \notin C(S)$

$w \notin H_0 \Rightarrow M_w$ stoppt nicht mit leerem Band
$\Rightarrow M$ berechnet Ω
$\Rightarrow$ die von $M_{g(w)}$ berechnete Fkt liegt in S
$\Rightarrow g(w) \in C(S)$

Damit $\overline{H_0} \leq_m C(S)$ und da $\overline{H_0}$ nicht entscheidbar, gilt dies auch für $C(S)$

Fall $\Omega \notin S$: Zeige analog $H_0 \leq_m C(S)$.

$\Rightarrow$ In allen Fällen gilt $H_0 \leq_m C(S)$ und damit ist $C(S)$ immer unentscheidbar. #

nicht semi-entscheidbar sein kann, da H_0 semi-entscheidbar ist (für eine Eingabe ω simulieren wir einfach die Maschine M_ω mit dem leeren Band als Eingabe). Mit Hilfe von H_0 wollen wir nun den Satz von Rice zeigen:

Satz 2.25 (Satz von Rice) *Sei $\mathcal{TM}$ die Menge aller Turing-berechenbaren Funktionen und sei $S \subseteq \mathcal{TM}$, wobei $S \neq \emptyset$ und $S \neq \mathcal{TM}$. Dann ist die Menge*

$$C(S) =_{\text{def}} \{\omega \mid \textit{die von } M_\omega \textit{ berechnete Funktion liegt in } S\}$$

unentscheidbar.

Beispiel 2.26 *Sei $S = \{S \mid f \textit{ ist eine konstante Funktion}\}$. Dann ist*

$$C(S) = \{\omega \mid M_\omega \textit{ berechnet eine konstante Funktion}\}$$

unentscheidbar, d.h. nahezu jede (semantische) Eigenschaft von Turingmaschinen ist nicht entscheidbar!

Beweis: Wir benutzen die überall undefinierte Funktion Ω, die durch `while(1); /* hält für keine Eingabe */` „berechnet“ wird.

Fall 1: Sei $\Omega \in S$. Da $S \neq \mathcal{TM}$ gibt es eine Fkt $f \in \mathcal{TM} \backslash S$. Sei Q eine TM, die f berechnet.

Wir ordnen nun jeder Gödelnummer $\omega \in \{0,1\}^*$ die folgende Turingmaschine M zu:

angesetzt auf Eingabe y, ignoriere zunächst y;

verhalte dich wie M_ω;

falls M_ω hält, verhalte dich wie Q;

Die TM M berechnet also die folgende Funktion f'

$$f' = \begin{cases} \Omega, & \text{falls } M_\omega \text{ auf leerem Band nicht stoppt} \\ f, & \text{sonst} \end{cases}$$

Wieder ist die Abbildung g, die ω die Maschine M zuordnet, berechenbar und es gilt sogar:

$$\begin{aligned} \omega \in H_0 &\Rightarrow M_\omega \text{ stoppt mit leerem Band} \\ &\Rightarrow M \text{ berechnet } f \\ &\Rightarrow \text{die von } M_{g(\omega)} \text{ berechnete Funktion liegt nicht in } S \\ &\Rightarrow g(\omega) \notin C(S) \end{aligned}$$

und

$$\begin{aligned} \omega \notin H_0 &\Rightarrow M_\omega \text{ stoppt nicht mit leerem Band} \\ &\Rightarrow M \text{ berechnet } \Omega \\ &\Rightarrow \text{die von } M_{g(\omega)} \text{ berechnete Funktion liegt in } S \\ &\Rightarrow g(\omega) \in C(S) \end{aligned}$$

Damit haben wir gezeigt, dass $\overline{H_0} \leq_{\text{m}} C(S)$, da ja gilt $\omega \in \overline{H_0}$ gdw. $g(\omega) \in C(S)$. Wir wissen schon, dass $\overline{H_0}$ aufgrund von Lemma 2.16 nicht semi-entscheidbar sein kann (da H_0 semi-entscheidbar, aber nicht entscheidbar ist). Also ist $\overline{H_0}$ erst recht nicht entscheidbar und damit auch $C(S)$ nicht entscheidbar.

Fall 2: Sei $\Omega \notin S$. Zeige völlig analog $H_0 \leq_{\text{m}} C(S)$.

In beiden möglichen Fällen haben wir durch eine geeignete Reduktion gezeigt, dass $C(S)$ nicht entscheidbar ist, wodurch der Satz von Rice gezeigt ist. #

Exkurs: Randomisierte Algorithmen

Wir haben in verschiedenen Berechnungsmodellen Nichtdeterminismus eingeführt:

z.B. führe das Schlüsselwort „guess" ein:

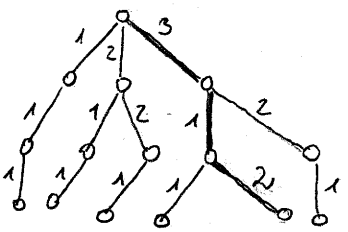
i) Berechnung verzweigt sich in unterschiedliche Zweige (≙ „Kopie der Maschine")

ii) Die Kopien können nicht kommunizieren

Klar: Dies Konzept kann nicht/schwer technisch realisiert werden

---

Idee: Führen das Schlüsselwort „toss" ein.
Wähle genau einen der Äste im Berechnungsbaum zufällig aus.



Def: Die Menge aller Entscheidungsprobleme mit einem Lösungsalgorithmus A und polynomieller Laufzeit und

Wahrscheinlichkeit

- $x \in L \Rightarrow Pr[A(x) = 1] \geq 2/3$
- $x \notin L \Rightarrow Pr[A(x) = 0] \geq 2/3$

heißt BPP („Bounded Error Probabilistic Polynomial Time")

Bem: Algorithmen dieses Typs nennt man auch „Monte-Carlo Algorithmen"

Wird ein BPP-Algorithmus mehrfach ausgeführt, so sinkt die Irrtumswahrscheinlichkeit („Amplification")

Def: Die Menge aller Sprachen L aus BPP mit

- $x \in L \Rightarrow Pr[A(x) = 1] \geq 2/3$
- $x \in L \Rightarrow Pr[A(x) = 0] = 1$

heißt RP („Randomized Polynomial Time").

Wenn

- $x \in L \Rightarrow Pr[A(x) = 1] = 1$
- $x \notin L \Rightarrow Pr[A(x) = 0] \geq 2/3$, dann

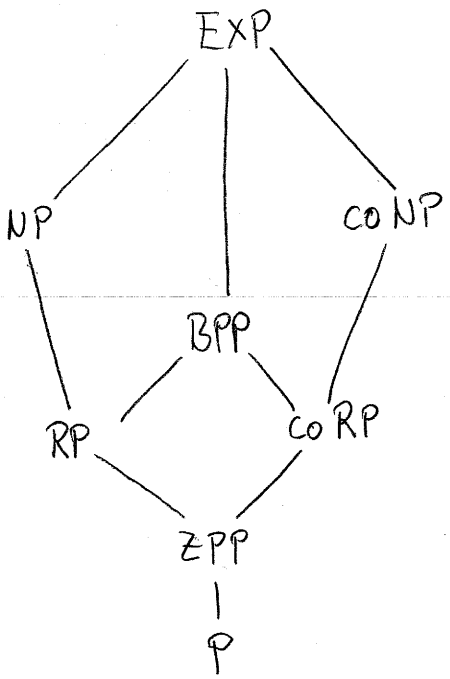
ist L aus co RP.

Def: $ZPP =_{def} RP \cap co\text{-}RP$ („Zero Error Probabilistic Polynomial Time").

Bem: ZPP-Algorithmen heißen auch „Las Vegas Algorithmen"

ZPP-Algorithmen dürfen mit „weiss nicht" antworten

Übersicht:



Ziel: Zeige, dass $PRIMES \in coRP$

Bem: Es gilt sogar $PRIMES \in P$, aber der bekannte Algorithmus ist zu langsam.

Idee: Verwende den kleinen Fermat als Primzahltest:

Wenn p prim und $p \nmid a$, dann gilt

$$a^{p-1} \equiv 1 \bmod p$$

Also: Wenn $a^{p-1} \equiv 1 \bmod p$, dann p prim?

$\Rightarrow$ Funktioniert nicht!

Def: Sei $n \geq 3$ eine zusammengesetzte Zahl. Gilt für alle a mit $ggT(a,n) = 1$

$$a^{n-1} \equiv 1 \bmod n,$$

dann heißt n Carmichael-Zahl.

Es ist bekannt, dass unendlich viele Carmichael-Zahlen ex. ($\Rightarrow$ Idee kann nicht mit einer Tabelle gerettet werden). Die ersten Carmichael-Zahlen sind 561, 1105, 1729.

Satz: Sei n eine Carmichal-Zahl und $p \mid n$, p prim. Dann gilt $p^2 \nmid n$. (n ist quadratfrei)

Beweis: Delfs & Knebl, Introduction to Cryptography, Springer #

Satz: Jede Carmichael-Zahl enthält mindestens drei verschiedene Primzahlen.

Beweis: Defs & Knebl #

Def: Sei $n \in \mathbb{N}$, $n > 2$ und $a \in \mathbb{Z}$. Die Zahl a heißt quadratischer Rest mod n (kurz: QR_n), wenn $y^2 \equiv a \bmod n$ eine Lsg hat.
Sonst heißt a quadratischer Nichtrest mod n (kurz: QNR_n)

Def: Sei $p > 2$ prim und $a \in \mathbb{Z}$, dann heißt

$$\left(\frac{a}{p}\right) =_{def} \begin{cases} +1, & \text{falls } a\ QR_p \text{ und } p \nmid a \\ 0, & \text{falls } p \mid a \\ -1, & \text{falls } x\ QNR_p \text{ und } p \nmid a \end{cases}$$

Legendre-Symbol. Sei $n \in \mathbb{N}$, $n > 2$ und $n = \prod_{i=1}^{r} p_i^{e_i}$. Für $a \in \mathbb{N}$ heißt

$$\left(\frac{a}{n}\right) =_{def} \prod_{i=1}^{r} \left(\frac{x}{p_i}\right)^{e_i}$$ Jacobi-Symbol.

Satz (Euler-Kriterium): Sei p prim, $p>2$ und $a \in \mathbb{Z}$, dann

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \mod p.$$

Beweis: bel. Buch zur Zahlentheorie #

Bem: Es gibt einen effizienten Algorithmus zur Berechnung des Legendre-Symbol der das quadratische Reziprozitätsgesetzt verwendet.

Def: Sei $n>2$ ungerade, dann heißt

$$SSZ_n =_{def} \{ a \in \mathbb{Z}_n^* \mid \left(\frac{a}{n}\right) \not\equiv a^{(p-1)/2} \mod n \}$$

Menge der Solovay-Strassen Zeugen gegen die Primalität von n.

Lemma: Sei p prim, $p>2$, dann $SSZ_p = \emptyset$

Beweis: Euler-Kriterium. #

Lemma: Sei $n>2$ ungerade und nicht prim, dann gilt $|SSZ_n| \geq \varphi(n)/2$.

„viele Zeugen gegen die Primalität von n"

Beweis: Sei $\overline{SSZ}_n =_{def} \mathbb{Z}_n^* \setminus SSZ_n = \{a \in \mathbb{Z}_n^* \mid a^{(n-1)/2} \cdot \left(\frac{a}{n}\right)^{-1} \equiv 1 \bmod n\}$.

Beh: $\overline{SSZ}$ ist eine Untergruppe von $\mathbb{Z}_n^*$

Bew: Übung #

Gilt $\overline{SSZ}_n \subset \mathbb{Z}_n^*$, dann muß $|SSZ_n| = |\mathbb{Z}_n^*| - |\overline{SSZ}_n| \geq \varphi(n) - \varphi(n)/2 = \varphi(n)/2$ gelten, da nach dem Satz von Lagrange für $|SSZ_n|$ nur Teiler von $|\mathbb{Z}_n^*| = \varphi(n)$ in Frage kommen. D.h. in diesem Fall gibt es viele Zeugen gegen die Primalität von n.

Beh: Wenn $\overline{SSZ}_n = \mathbb{Z}_n^*$, dann ist n eine Primzahl

Beweis: Angenommen $\overline{SSZ}_n = \mathbb{Z}_n^*$ und n ist nicht prim. Da ja $\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \bmod n$ für alle $a \in \mathbb{Z}_n^*$, gilt $a^{n-1} \equiv 1 \bmod n$, d.h. a ist eine Carmichael-Zahl. Also kommt kein Primfaktor in n doppelt vor und $n = p_1 p_2 \cdot \ldots \cdot p_k$, $k \geq 3$.

Sei v ein QNR_{p_1}, d.h. $\left(\frac{v}{p_1}\right) = -1$. Nun kann man ein x wählen, so dass

- $x \equiv v \bmod p_1$
- $x \equiv 1 \bmod n/p_1$ (vgl. chinesischen Restsatz)

Klar: $\underbrace{\left(\frac{x}{p_2}\right) = \left(\frac{x}{p_3}\right) = \ldots = \left(\frac{x}{p_k}\right) = 1}_{1 \text{ ist } QR_{p_i}}$, d.h. $\left(\frac{x}{n}\right) = \left(\frac{x}{p_1}\right) \cdot \ldots \cdot \left(\frac{x}{p_k}\right) = -1$

Da $\overline{SSZ} = \mathbb{Z}_n^*$ gilt $x^{(n-1)/2} \equiv \left(\frac{x}{n}\right) \bmod n$ und damit $x^{(n-1)/2} \equiv -1 \bmod n$. Also auch $x^{(n-1)/2} \equiv -1 \bmod p_2$. Dies ist ein Widerspruch, da ja $x \equiv 1 \bmod n/p_1$ und damit $x \equiv 1 \bmod p_2$ #

Es geht aber noch besser:

Def: Sei $n > 2$, n ungerade und $n-1 = 2^s t$, wobei $2 \nmid t$. Dann heißt

$$MRZ_n =_{def} \{ a \in \mathbb{Z}^* \mid a^t \neq 1 \bmod n \wedge a^{2^j t} \neq -1 \bmod n \text{ für } 0 \leq j \leq s-1 \}$$

Menge der Miller-Rabin Zeugen gegen die Primeigenschaft von n.

Satz: Ist n eine Primzahl, dann $MRZ_n = \emptyset$, sonst $|MRZ_n| \geq \frac{3}{4} \varphi(n)$

Beweis: z.B. Koblitz, A Course in Number Theory and Cryptography. #

Algorithmus A („Miller-Rabin Test"):

würfle ein $a \in \mathbb{Z}_n^*$;
if $a \in MRZ_n$ then
  return „composite";
else
  return „prim";
endif.

Also

$$n \in PRIM \Rightarrow Pr[A(n) = \text{„prim"}] = 1$$
$$n \notin PRIM \Rightarrow Pr[A(n) = \text{„composite"}] \geq 3/4$$

$\Rightarrow$ PRIMES $\in$ co RP.

Bem:
- Es ist sogar ein Algorithmus bekannt für PRIMES $\in$ RP („Kompliziert")
- Der AKS-Algorithmus zeigt PRIMES $\in$ P

– Ende der Vorlesung –