Diagonalisierung und polynomielle Hierarchie

Corvin Paul, Matthias Schimek

Institut für Theoretische Informatik - Algorithmik I

Gliederung

Diagonalisierung
- Einleitung
- Time Hierarchy
- Satz von Ladner
- Orakelmaschinen - Grenzen der Diagonalisierung
- Grenzen der Diagonalisierung

Die polynomielle Hierarchie
- Einleitung
- Die Klasse $\sum_2^p$
- Die Klasse **PH**

Vorraussetzungen

Wiederholung :

- Für $i \in \mathbb{N}$ beschreibt i die TM M_i
- Jede TM wird von unendlich vielen $i \in \mathbb{N}$ beschrieben
- Es existiert eine universelle TM U, die jede TM mit logarithmischem Overhead simulieren kann

Vorraussetzungen

Wiederholung :

- Für $i \in \mathbb{N}$ beschreibt i die TM M_i
- Jede TM wird von unendlich vielen $i \in \mathbb{N}$ beschrieben
- Es existiert eine universelle TM U, die jede TM mit logarithmischem Overhead simulieren kann

Vorraussetzungen

Wiederholung :

- Für $i \in \mathbb{N}$ beschreibt i die TM M_i
- Jede TM wird von unendlich vielen $i \in \mathbb{N}$ beschrieben
- Es existiert eine universelle TM U, die jede TM mit logarithmischem Overhead simulieren kann

Vorraussetzungen

Karlsruher Institut für Technologie

Universelle TM

TM M_i läuft bei Eingabe x in $\mathcal{O}(f(n)) \Rightarrow$ TM U läuft bei Eingabe i, x in $\mathcal{O}(f(n)log(fn))$

Vorraussetzungen

Definition Time-constructible functions

Wir nennen eine Funktion f time-constructible, falls gilt :
f(n) ist in $\mathcal{O}(f(n))$ berechenbar.

Definition DTIME

DTIME(f(n)) = { L |∃ deterministische Turingmaschine , die L in $\mathcal{O}(f(n))$ entscheidet }

Fakultät für Informatik
Institut für Theoretische Informatik

Vorraussetzungen

Definition Time-constructible functions

Wir nennen eine Funktion f time-constructible, falls gilt :
f(n) ist in $\mathcal{O}(f(n))$ berechenbar.

Definition DTIME

DTIME(f(n)) = { L $|\exists$ deterministische Turingmaschine , die L in $\mathcal{O}(f(n))$ entscheidet }

Fakultät für Informatik
Institut für Theoretische Informatik

Deterministische Time Hierarchy

Satz Time Hierarchy Theorem, 65

Wenn f, g time-constructible Funktionen sind die $f(n)\log(f(n)) \in o(g(n))$ erfüllen, dann gilt
$\mathbf{DTIME}(f(n)) \subsetneq \mathbf{DTIME}(g(n))$

Frage : Warum brauchen wir den Faktor $\log(f(n))$?

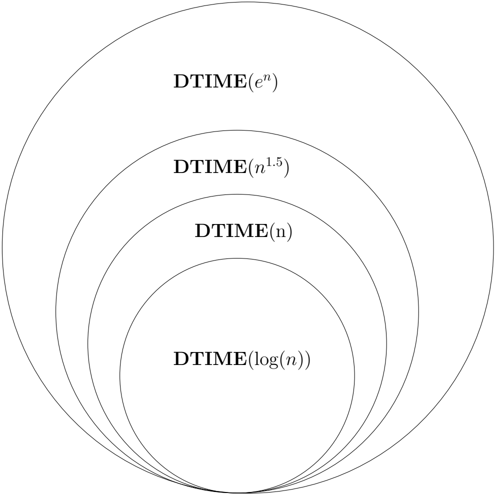
Deterministische Time Hierarchy

Satz Time Hierarchy Theorem, 65

Wenn f, g time-constructible Funktionen sind die $f(n)\log(f(n)) \in o(g(n))$ erfüllen, dann gilt
$\mathbf{DTIME}(f(n)) \subsetneq \mathbf{DTIME}(g(n))$

Frage : Warum brauchen wir den Faktor $\log(f(n))$?

Deterministische Time Hierarchy



Beweis det. Time Hierarchy

Wir zeigen nur $\mathbf{DTIME}(n) \subsetneq \mathbf{DTIME}(n^{1.5})$

Definition Turing Maschine D

Bei Eingabe x : Simuliere die TM M_x mit Eingabe x genau für $|x|^{1.4}$ Schritte. Danach gebe folgendes aus :

$$D(x) = \begin{cases} \overline{M_x(x)} & \text{falls die Simulation eine Ausgabe hatte} \\ 0 & \text{sonst} \end{cases} \tag{1}$$

Die von D erzeugte Sprache

Sei L = $\{x | D(x) = 1\}$

Beweis det. Time Hierarchy

Wir zeigen nur $\mathbf{DTIME}(n) \subsetneq \mathbf{DTIME}(n^{1.5})$

Definition Turing Maschine D

Bei Eingabe x : Simuliere die TM M_x mit Eingabe x genau für $|x|^{1.4}$ Schritte. Danach gebe folgendes aus :

$$D(x) = \begin{cases} \overline{M_x(x)} & \text{falls die Simulation eine Ausgabe hatte} \\ 0 & \text{sonst} \end{cases} \tag{1}$$

Die von D erzeugte Sprache

Sei L = $\{x | D(x) = 1\}$

Beweis det. Time Hierarchy

Wir zeigen nur $\mathbf{DTIME}(n) \subsetneq \mathbf{DTIME}(n^{1.5})$

Definition Turing Maschine D

Bei Eingabe x : Simuliere die TM M_x mit Eingabe x genau für $|x|^{1.4}$ Schritte. Danach gebe folgendes aus :

$$D(x) = \begin{cases} \overline{M_x(x)} & \text{falls die Simulation eine Ausgabe hatte} \\ 0 & \text{sonst} \end{cases} \tag{1}$$

Die von D erzeugte Sprache

Sei L = $\{x | D(x) = 1\}$

Beweis det. Time Hierarchy

Behauptung

$L \in \textbf{DTIME}(n^{1.5})$ und $L \notin \textbf{DTIME}(n)$

- Wir nehmen an , dass $L \in \textbf{DTIME}(n)$
- $\Rightarrow \exists$ Turing Maschine M , die L entscheidet und für Eingabe x höchstens c|x| Schritte benötigt. (c ist konstant)
- $\Leftrightarrow \forall x \in \{0, 1\}^*$ D(x) = M(x)

Beweis det. Time Hierarchy

Behauptung

$L \in \mathbf{DTIME}(n^{1.5})$ und $L \notin \mathbf{DTIME}(n)$

- Wir nehmen an , dass $L \in \mathbf{DTIME}(n)$
- $\Rightarrow \exists$ Turing Maschine M , die L entscheidet und für Eingabe x höchstens c|x| Schritte benötigt. (c ist konstant)
- $\Leftrightarrow \forall x \in \{0, 1\}^*$ D(x) = M(x)

Beweis det. Time Hierarchy

Behauptung

$L \in \mathbf{DTIME}(n^{1.5})$ und $L \notin \mathbf{DTIME}(n)$

- Wir nehmen an , dass $L \in \mathbf{DTIME}(n)$
- $\Rightarrow \exists$ Turing Maschine M , die L entscheidet und für Eingabe x höchstens c|x| Schritte benötigt. (c ist konstant)
- $\Leftrightarrow \forall x \in \{0,1\}^*$ D(x) = M(x)

Beweis det. Time Hierarchy

Behauptung

$L \in \mathbf{DTIME}(n^{1.5})$ und $L \notin \mathbf{DTIME}(n)$

- Wir nehmen an , dass $L \in \mathbf{DTIME}(n)$
- $\Rightarrow \exists$ Turing Maschine M , die L entscheidet und für Eingabe x höchstens c|x| Schritte benötigt. (c ist konstant)
- $\Leftrightarrow \forall x \in \{0, 1\}^*$ D(x) = M(x)

Beweis det. Time Hierarchy

- Wollen nun M auf D simulieren können
- M simuliert auf U läuft in $c|x|\log(|x|)$
- Wir wählen dazu n_0 so groß, dass $\forall\ n \geq n_0$ gilt : $n^{1.4} > cn\log(n)$
- Nun wählen wir eine Gödelnummer x , so dass $|x| > n_0$ und $M_x = M$
- Nun gilt $D(x) \neq M(x)$

$\square$

Beweis det. Time Hierarchy

- Wollen nun M auf D simulieren können
- M simuliert auf U läuft in $c|x|\log(|x|)$
- Wir wählen dazu n_0 so groß, dass $\forall\ n \geq n_0$ gilt : $n^{1.4} > cn\log(n)$
- Nun wählen wir eine Gödelnummer x , so dass $|x| > n_0$ und M_x = M
- Nun gilt $D(x) \neq M(x)$

$\square$

Fakultät für Informatik
Institut für Theoretische Informatik

Beweis det. Time Hierarchy

- Wollen nun M auf D simulieren können
- M simuliert auf U läuft in $c|x|\log(|x|)$
- Wir wählen dazu n_0 so groß, dass $\forall\ n \geq n_0$ gilt : $n^{1.4} > cn\log(n)$
- Nun wählen wir eine Gödelnummer x , so dass $|x| > n_0$ und $M_x = M$
- Nun gilt $D(x) \neq M(x)$

$\square$

Beweis det. Time Hierarchy

- Wollen nun M auf D simulieren können
- M simuliert auf U läuft in $c|x|\log(|x|)$
- Wir wählen dazu n_0 so groß, dass $\forall\ n \geq n_0$ gilt : $n^{1.4} > cn\log(n)$
- Nun wählen wir eine Gödelnummer x , so dass $|x| > n_0$ und M_x = M
- Nun gilt $D(x) \neq M(x)$

$\square$

Beweis det. Time Hierarchy

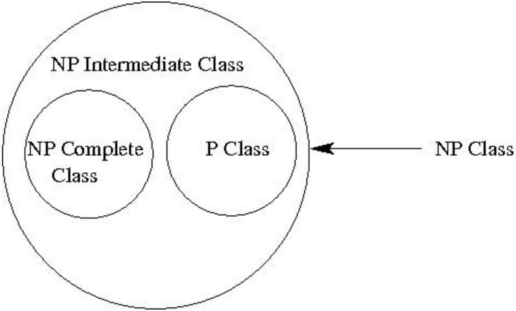
- Wollen nun M auf D simulieren können
- M simuliert auf U läuft in $c|x|\log(|x|)$
- Wir wählen dazu n_0 so groß, dass $\forall\ n \geq n_0$ gilt : $n^{1.4} > cn\log(n)$
- Nun wählen wir eine Gödelnummer x , so dass $|x| > n_0$ und M_x = M
- Nun gilt $D(x) \neq M(x)$

$\square$

Frage : Gibt es **NP** Probleme , die nicht **NP**-vollständig sind , aber auch nicht in **P** liegen

NP-intermediate

Figure: **NP**-Intermediate



http://functionspace.org/articles/28/Complexity-Zoo]

NP-intermediate Probleme

Mögliche Kandidaten :

- Graphisomorphie (kommt in Vortrag 7)
- Faktorisierungsproblem
- Kein "natürlicher" Kandidat bekannt

aber,

Satz von Ladner

Behauptung

Existenz einer NP-intermediate Sprache, Ladner, 75

Wenn $\mathbf{P} \neq \mathbf{NP}$ dann gilt :
Es existiert eine Sprache $L \in \mathbf{NP} \setminus \mathbf{P}$ die nicht **NP**-vollständig ist

Beweis von Ladner

Beweisidee

Wollen eine Sprache konstruieren mit diesen Eigenschaften und zeigen, dass sie in **NP** - intermediate ist, falls **P** $\neq$ **NP** :

Die Sprache $\mathbf{SAT}_H$

Für eine Funktion $H : \mathbb{N} \to \mathbb{N}$ definieren wir :
$\mathbf{SAT}_H = \{\psi 0 1^{n^{H(n)}} : \psi \in \mathbf{SAT} \text{ und } n = |\psi|\}$

Beispiel für $\mathbf{SAT}_H$

Für $H(n) = n - 1$ und $\psi = a \wedge b$ gilt :
$(a \wedge b)01^{3^2} = (a \wedge b)0111111111 \in \mathbf{SAT}_H$

$\mathbf{SAT}_H$ ist also **SAT** mit 1er padding

Beweis von Ladner

Beweisidee

Wollen eine Sprache konstruieren mit diesen Eigenschaften und zeigen, dass sie in **NP** - intermediate ist, falls **P** $\neq$ **NP** :

Die Sprache SAT$_H$

Für eine Funktion $H : \mathbb{N} \to \mathbb{N}$ definieren wir :
SAT$_H = \{\psi 01^{n^{H(n)}} : \psi \in$ **SAT** und $n = |\psi|\}$

Beispiel für SAT$_H$

Für $H(n) = n - 1$ und $\psi = a \wedge b$ gilt :
$(a \wedge b)01^{3^2} = (a \wedge b)0111111111 \in$ **SAT**$_H$

SAT$_H$ ist also **SAT** mit 1er padding

Beweis von Ladner

Beweisidee

Wollen eine Sprache konstruieren mit diesen Eigenschaften und zeigen, dass sie in **NP** - intermediate ist, falls $\mathbf{P} \neq \mathbf{NP}$:

Die Sprache SAT$_H$

Für eine Funktion $H : \mathbb{N} \to \mathbb{N}$ definieren wir :

$\mathbf{SAT}_H = \{\psi 0 1^{n^{H(n)}} : \psi \in \mathbf{SAT} \text{ und } n = |\psi|\}$

Beispiel für SAT$_H$

Für $H(n) = n - 1$ und $\psi = a \wedge b$ gilt :

$(a \wedge b)01^{3^2} = (a \wedge b)0111111111 \in \mathbf{SAT}_H$

$\mathbf{SAT}_H$ ist also **SAT** mit 1er padding

Beweis von Ladner

Beweisidee

Wollen eine Sprache konstruieren mit diesen Eigenschaften und zeigen, dass sie in **NP** - intermediate ist, falls $\mathbf{P} \neq \mathbf{NP}$:

Die Sprache SAT_H

Für eine Funktion $H : \mathbb{N} \to \mathbb{N}$ definieren wir :
$\mathbf{SAT}_H = \{\psi 0 1^{n^{H(n)}} : \psi \in \mathbf{SAT} \text{ und } n = |\psi|\}$

Beispiel für SAT_H

Für $H(n) = n - 1$ und $\psi = a \wedge b$ gilt :
$(a \wedge b)01^{3^2} = (a \wedge b)0111111111 \in \mathbf{SAT}_H$

$\mathbf{SAT}_H$ ist also **SAT** mit 1er padding

Fakultät für Informatik
Institut für Theoretische Informatik

Beweis von Ladner

Lemma

Wir müssen nun also H geschickt wählen !

Eigenschaft, die wir von H wollen

$\mathbf{SAT}_H \in \mathbf{P} \Leftrightarrow H(n) \in O(1)$ (also $H(n) \leq C$ für alle n)
und damit insbesondere $\lim_{n\to\infty} H(n) = \infty$ für $\mathbf{SAT}_H \notin \mathbf{P}$

Beweis von Ladner

Lemma

Wir müssen nun also H geschickt wählen !

Eigenschaft, die wir von H wollen

$\mathbf{SAT}_H \in \mathbf{P} \Leftrightarrow H(n) \in O(1)$ (also $H(n) \leq C$ für alle n)
und damit insbesondere $\lim_{n\to\infty} H(n) = \infty$ für $\mathbf{SAT}_H \notin \mathbf{P}$

Beweis von Ladner

Nachweis der Eigenschaften von H

Definition von H

$H(n)$ ist die kleinste Gödelnummer $i < \log(\log(n))$ so dass für alle $x \in \{0,1\}^*$ mit $|x| \leq \log(n)$ die Turing Maschine M_i genau $\textbf{SAT}_H(x)$ in $i|x|^i$ Schritten berechnet. Falls dieses i nicht existiert setzen wir $H(n) = \log(\log(n))$

Beweis von Ladner

Nachweis der Eigenschaften von H

Definition von H

$H(n)$ ist die kleinste Gödelnummer $i < \log(\log(n))$ so dass für alle $x \in \{0,1\}^*$ mit $|x| \leq \log(n)$ die Turing Maschine M_i genau $\mathbf{SAT}_H(x)$ in $i|x|^i$ Schritten berechnet. Falls dieses i nicht existiert setzen wir $H(n) = \log(\log(n))$

Zuerst zeigen wir : $\mathbf{SAT}_H \in \mathbf{P} \Rightarrow H(n) \in O(1)$

Beweis von Ladner

Nachweis der Eigenschaften von H

Definition von H

$H(n)$ ist die kleinste Gödelnummer $i < \log(\log(n))$ so dass für alle $x \in \{0,1\}^*$ mit $|x| \leq \log(n)$ die Turing Maschine M_i genau $\mathbf{SAT}_H(x)$ in $i|x|^i$ Schritten berechnet. Falls dieses i nicht existiert setzen wir $H(n) = \log(\log(n))$

Zuerst zeigen wir : $\mathbf{SAT}_H \in \mathbf{P} \Rightarrow H(n) \in O(1)$

- $\mathbf{SAT}_H \in \mathbf{P} \Rightarrow \exists$ Turing Maschine M, die $\mathbf{SAT}_H$ in höchstens cn^c Schritten entscheidet.
- $\exists i > c$, so dass M_i = M
- Für n > 2^{2^i} gilt $H(n) \leq i$ und damit $H(n) \in O(1)$

Fakultät für Informatik
Institut für Theoretische Informatik

Beweis von Ladner

Nachweis der Eigenschaften von H

Definition von H

$H(n)$ ist die kleinste Gödelnummer $i < \log(\log(n))$ so dass für alle $x \in \{0,1\}^*$ mit $|x| \leq \log(n)$ die Turing Maschine M_i genau $\mathbf{SAT}_H(x)$ in $i|x|^i$ Schritten berechnet. Falls dieses i nicht existiert setzen wir $H(n) = \log(\log(n))$

Zuerst zeigen wir : $\mathbf{SAT}_H \in \mathbf{P} \Rightarrow H(n) \in O(1)$

- $\mathbf{SAT}_H \in \mathbf{P} \Rightarrow \exists$ Turing Maschine M, die $\mathbf{SAT}_H$ in höchstens cn^c Schritten entscheidet.
- $\exists i > c$, so dass M_i = M
- Für n > 2^{2^i} gilt $H(n) \leq i$ und damit $H(n) \in O(1)$

Beweis von Ladner

Nachweis der Eigenschaften von H

Definition von H

$H(n)$ ist die kleinste Gödelnummer $i < \log(\log(n))$ so dass für alle $x \in \{0,1\}^*$ mit $|x| \leq \log(n)$ die Turing Maschine M_i genau $\mathbf{SAT}_H(x)$ in $i|x|^i$ Schritten berechnet. Falls dieses i nicht existiert setzen wir $H(n) = \log(\log(n))$

Zuerst zeigen wir : $\mathbf{SAT}_H \in \mathbf{P} \Rightarrow H(n) \in O(1)$

- $\mathbf{SAT}_H \in \mathbf{P} \Rightarrow \exists$ Turing Maschine M, die $\mathbf{SAT}_H$ in höchstens cn^c Schritten entscheidet.
- $\exists i > c$, so dass M_i = M
- Für n > 2^{2^i} gilt $H(n) \leq i$ und damit $H(n) \in O(1)$

Beweis von Ladner

Nachweis der Eigenschaften von H

Definition von H

$H(n)$ ist die kleinste Gödelnummer $i < \log(\log(n))$ so dass für alle $x \in \{0,1\}^*$ mit $|x| \le \log(n)$ die Turing Maschine M_i genau $\mathbf{SAT}_H(x)$ in $i|x|^i$ Schritten berechnet. Falls dieses i nicht existiert setzen wir $H(n) = \log(\log(n))$

Nun : $H(n) \in \mathcal{O}(1) \Rightarrow \mathbf{SAT}_H \in \mathbf{P}$

- Da $H(n) \in \mathcal{O}(1)$ ist Bild von H endlich $\Rightarrow \exists i$ mit $H(n) = i$ für unendlich viele n
- TM M_i löst SAT_H in in^i Schritten
- Denn, angenommen $\exists x$ für welches M_i dies nicht in dieser Grenze schafft $\Rightarrow H(n) \neq i$ für alle $n > 2^{|x|}$ nach Definition von H

Beweis von Ladner

Nachweis der Eigenschaften von H

Definition von H

$H(n)$ ist die kleinste Gödelnummer $i < \log(\log(n))$ so dass für alle $x \in \{0,1\}^*$ mit $|x| \leq \log(n)$ die Turing Maschine M_i genau $\mathbf{SAT}_H(x)$ in $i|x|^i$ Schritten berechnet. Falls dieses i nicht existiert setzen wir $H(n) = \log(\log(n))$

Nun : $H(n) \in \mathcal{O}(1) \Rightarrow \mathbf{SAT}_H \in \mathbf{P}$

- Da $H(n) \in \mathcal{O}(1)$ ist Bild von H endlich $\Rightarrow \exists i$ mit $H(n) = i$ für unendlich viele n
- TM M_i löst SAT_H in in^i Schritten
- Denn, angenommen $\exists x$ für welches M_i dies nicht in dieser Grenze schafft $\Rightarrow H(n) \neq i$ für alle $n > 2^{|x|}$ nach Definition von H

Fakultät für Informatik
Institut für Theoretische Informatik

Beweis von Ladner

Nachweis der Eigenschaften von H

Definition von H

$H(n)$ ist die kleinste Gödelnummer $i < \log(\log(n))$ so dass für alle $x \in \{0,1\}^*$ mit $|x| \leq \log(n)$ die Turing Maschine M_i genau $\mathbf{SAT}_H(x)$ in $i|x|^i$ Schritten berechnet. Falls dieses i nicht existiert setzen wir $H(n) = \log(\log(n))$

Nun : $H(n) \in \mathcal{O}(1) \Rightarrow \mathbf{SAT}_H \in \mathbf{P}$

- Da $H(n) \in \mathcal{O}(1)$ ist Bild von H endlich $\Rightarrow \exists i$ mit $H(n) = i$ für unendlich viele n
- TM M_i löst SAT_H in in^i Schritten
- Denn, angenommen $\exists x$ für welches M_i dies nicht in dieser Grenze schafft $\Rightarrow H(n) \neq i$ für alle $n > 2^{|x|}$ nach Definition von H

Fakultät für Informatik
Institut für Theoretische Informatik

Beweis von Ladner

Nachweis der Eigenschaften von H

Definition von H

$H(n)$ ist die kleinste Gödelnummer $i < \log(\log(n))$ so dass für alle $x \in \{0,1\}^*$ mit $|x| \leq \log(n)$ die Turing Maschine M_i genau $\mathbf{SAT}_H(x)$ in $i|x|^i$ Schritten berechnet. Falls dieses i nicht existiert setzen wir $H(n) = \log(\log(n))$

Nun : $H(n) \in \mathcal{O}(1) \Rightarrow \mathbf{SAT}_H \in \mathbf{P}$

- Da $H(n) \in \mathcal{O}(1)$ ist Bild von H endlich $\Rightarrow \exists i$ mit $H(n) = i$ für unendlich viele n
- TM M_i löst SAT_H in in^i Schritten
- Denn, angenommen $\exists x$ für welches M_i dies nicht in dieser Grenze schafft $\Rightarrow H(n) \neq i$ für alle $n > 2^{|x|}$ nach Definition von H

Beweis von Ladner

$\mathbf{SAT}_H$ weder in P noch NP-complete

Definiton von SAT_H

Für eine Funktion $H : \mathbb{N} \to \mathbb{N}$ definieren wir :
$\mathbf{SAT}_H = \{\psi 0 1^{n^{H(n)}} : \psi \in \mathbf{SAT} \text{ und } n = |\psi|\}$

$\mathbf{SAT}_H$ ist nicht in P

- Angenommen $\mathbf{SAT}_H \in \mathbf{P} \Rightarrow H(n) \leq C$, C Konstante wegen des gerade bewiesenen Lemmas
- $\mathbf{SAT}_H$ ist also **SAT** mit höchsten n^C angehänten 1en
- **SAT** kann somit durch dieselbe TM gelöst werden $\Rightarrow \mathbf{P} = \mathbf{NP}$

Beweis von Ladner

$\textbf{SAT}_H$ weder in P noch NP-complete

Definiton von SAT_H

Für eine Funktion $H : \mathbb{N} \rightarrow \mathbb{N}$ definieren wir :
$\textbf{SAT}_H = \{\psi 0 1^{n^{H(n)}} : \psi \in \textbf{SAT} \text{ und } n = |\psi|\}$

$\textbf{SAT}_H$ ist nicht in P

- Angenommen $\textbf{SAT}_H \in \textbf{P} \Rightarrow H(n) \leq C$, C Konstante wegen des gerade bewiesenen Lemmas
- $\textbf{SAT}_H$ ist also **SAT** mit höchsten n^C angehänten 1en
- **SAT** kann somit durch dieselbe TM gelöst werden $\Rightarrow \textbf{P} = \textbf{NP}$

Beweis von Ladner

$\mathbf{SAT}_H$ weder in P noch NP-complete

Definiton von SAT_H

Für eine Funktion $H : \mathbb{N} \to \mathbb{N}$ definieren wir :
$\mathbf{SAT}_H = \{\psi 0 1^{n^{H(n)}} : \psi \in \mathbf{SAT} \text{ und } n = |\psi|\}$

$\mathbf{SAT}_H$ ist nicht in P

- Angenommen $\mathbf{SAT}_H \in \mathbf{P} \Rightarrow H(n) \leq C$, C Konstante wegen des gerade bewiesenen Lemmas
- $\mathbf{SAT}_H$ ist also **SAT** mit höchsten n^C angehänten 1en
- **SAT** kann somit durch dieselbe TM gelöst werden $\Rightarrow \mathbf{P} = \mathbf{NP}$

Fakultät für Informatik
Institut für Theoretische Informatik

Beweis von Ladner

$\mathbf{SAT}_H$ weder in P noch NP-complete

Definiton von SAT_H

Für eine Funktion $H : \mathbb{N} \to \mathbb{N}$ definieren wir :
$\mathbf{SAT}_H = \{\psi 0 1^{n^{H(n)}} : \psi \in \mathbf{SAT} \text{ und } n = |\psi|\}$

$\mathbf{SAT}_H$ ist nicht in P

- Angenommen $\mathbf{SAT}_H \in \mathbf{P} \Rightarrow H(n) \leq C$, C Konstante wegen des gerade bewiesenen Lemmas
- $\mathbf{SAT}_H$ ist also **SAT** mit höchsten n^C angehänten 1en
- **SAT** kann somit durch dieselbe TM gelöst werden $\Rightarrow \mathbf{P} = \mathbf{NP}$

Fakultät für Informatik
Institut für Theoretische Informatik

Beweis von Ladner

SAT$_H$ weder in P noch NP-complete

Definiton von SAT$_H$

Für eine Funktion $H : \mathbb{N} \to \mathbb{N}$ definieren wir :
$\textbf{SAT}_H = \{\psi 01^{n^{H(n)}} : \psi \in \textbf{SAT} \text{ und } n = |\psi|\}$

- Angenommen $\textbf{SAT}_H \in \textbf{NP} - \textit{complete} \Rightarrow$ es existiert poly. Reduktion f von **SAT** auf **SAT**$_H$ in $\mathcal{O}(n^i)$
- Da $\textbf{SAT}_H \notin \textbf{P}$ geht H gegen ∞
- **SAT**-Instanz φ wird mit f auf **SAT**$_H$-Instanz der Form $\psi 01^{n^{H(n)}}$ abgebildet und mit $f \in \mathcal{O}(n^i)$ folgt $|\psi| + |\psi|^{H(|\psi|)}$ und damit $|\psi| \in o(n)$
- Wegen $\psi| \in o(n)$ existiert dann ein Polynomialzeitalgorithmus für **SAT** und damit $\textbf{P} = \textbf{NP} \Rightarrow$ Widerspruch!

Beweis von Ladner

$\mathbf{SAT}_H$ weder in P noch NP-complete

Definiton von $\mathbf{SAT}_H$

Für eine Funktion $H : \mathbb{N} \to \mathbb{N}$ definieren wir :
$\mathbf{SAT}_H = \{\psi 0 1^{n^{H(n)}} : \psi \in \mathbf{SAT} \text{ und } n = |\psi|\}$

$\mathbf{SAT}_H$ ist nicht NP-complete

- Angenommen $\mathbf{SAT}_H \in \mathbf{NP} - \mathit{complete} \Rightarrow$ es existiert poly. Reduktion f von **SAT** auf $\mathbf{SAT}_H$ in $\mathcal{O}(n^i)$
- Da $\mathbf{SAT}_H \notin \mathbf{P}$ geht H gegen ∞
- **SAT**-Instanz φ wird mit f auf $\mathbf{SAT}_H$-Instanz der Form $\psi 0 1^{n^{H(n)}}$ abgebildet und mit $f \in \mathcal{O}(n^i)$ folgt $|\psi| + |\psi|^{H(|\psi|)}$ und damit $|\psi| \in o(n)$
- Wegen $\psi| \in o(n)$ existiert dann ein Polynomialzeitalgorithmus für **SAT** und damit $\mathbf{P} = \mathbf{NP} \Rightarrow$ Widerspruch!

Beweis von Ladner

SAT_H weder in P noch NP-complete

Definiton von SAT_H

Für eine Funktion $H : \mathbb{N} \to \mathbb{N}$ definieren wir :
$\textbf{SAT}_H = \{\psi 0 1^{n^{H(n)}} : \psi \in \textbf{SAT} \text{ und } n = |\psi|\}$

SAT_H ist nicht NP-complete

- Angenommen $\textbf{SAT}_H \in \textbf{NP} - \textit{complete} \Rightarrow$ es existiert poly. Reduktion f von **SAT** auf $\textbf{SAT}_H$ in $\mathcal{O}(n^i)$
- Da $\textbf{SAT}_H \notin \textbf{P}$ geht H gegen ∞
- **SAT**-Instanz φ wird mit f auf $\textbf{SAT}_H$-Instanz der Form $\psi 0 1^{n^{H(n)}}$ abgebildet und mit $f \in \mathcal{O}(n^i)$ folgt $|\psi| + |\psi|^{H(|\psi|)}$ und damit $|\psi| \in o(n)$
- Wegen $\psi| \in o(n)$ existiert dann ein Polynomialzeitalgorithmus für **SAT** und damit $\textbf{P} = \textbf{NP} \Rightarrow$ Widerspruch!

Fakultät für Informatik
Institut für Theoretische Informatik

Beweis von Ladner

SAT$_H$ weder in P noch NP-complete

Definiton von SAT$_H$

Für eine Funktion $H : \mathbb{N} \rightarrow \mathbb{N}$ definieren wir :
$\mathbf{SAT}_H = \{\psi 01^{n^{H(n)}} : \psi \in \mathbf{SAT} \text{ und } n = |\psi|\}$

SAT$_H$ ist nicht NP-complete

- Angenommen $\mathbf{SAT}_H \in \mathbf{NP} - \textit{complete} \Rightarrow$ es existiert poly. Reduktion f von **SAT** auf $\mathbf{SAT}_H$ in $\mathcal{O}(n^i)$
- Da $\mathbf{SAT}_H \notin \mathbf{P}$ geht H gegen ∞
- **SAT**-Instanz φ wird mit f auf $\mathbf{SAT}_H$-Instanz der Form $\psi 01^{n^{H(n)}}$ abgebildet und mit $f \in \mathcal{O}(n^i)$ folgt $|\psi| + |\psi|^{H(|\psi|)}$ und damit $|\psi| \in o(n)$
- Wegen $\psi| \in o(n)$ existiert dann ein Polynomialzeitalgorithmus für **SAT** und damit $\mathbf{P} = \mathbf{NP} \Rightarrow$ Widerspruch!

Beweis von Ladner

$\mathbf{SAT}_H$ weder in P noch NP-complete

Definiton von $\mathbf{SAT}_H$

Für eine Funktion $H : \mathbb{N} \to \mathbb{N}$ definieren wir :
$\mathbf{SAT}_H = \{\psi 01^{n^{H(n)}} : \psi \in \mathbf{SAT} \text{ und } n = |\psi|\}$

$\mathbf{SAT}_H$ ist nicht NP-complete

- Angenommen $\mathbf{SAT}_H \in \mathbf{NP} - \mathit{complete} \Rightarrow$ es existiert poly. Reduktion f von **SAT** auf $\mathbf{SAT}_H$ in $\mathcal{O}(n^i)$
- Da $\mathbf{SAT}_H \notin \mathbf{P}$ geht H gegen ∞
- **SAT**-Instanz φ wird mit f auf $\mathbf{SAT}_H$-Instanz der Form $\psi 01^{n^{H(n)}}$ abgebildet und mit $f \in \mathcal{O}(n^i)$ folgt $|\psi| + |\psi|^{H(|\psi|)}$ und damit $|\psi| \in o(n)$
- Wegen $\psi| \in o(n)$ existiert dann ein Polynomialzeitalgorithmus für **SAT** und damit $\mathbf{P} = \mathbf{NP} \Rightarrow$ Widerspruch!

Fakultät für Informatik
Institut für Theoretische Informatik

Grenzen der Diagonalisierung

Wiederholung Diagonalisierung

Was ist Diagonalisierung

Als Diagonalisierung wird (hier) ein Beweis bezeichnet, der nur auf den beiden folgenden Eigenschaften von TM aufbaut.

1. Die Existenz einer Repräsentation von TM durch Zeichenketten (Gödelnummer)
2. Die Fähigkeit eine andere TM mit geringem zusätzlichen Zeit- oder Platzbedarf zu simulieren (Universelle TM)

Grenzen der Diagonalisierung

Wiederholung Diagonalisierung

Was ist Diagonalisierung

Als Diagonalisierung wird (hier) ein Beweis bezeichnet, der nur auf den beiden folgenden Eigenschaften von TM aufbaut.

1. Die Existenz einer Repräsentation von TM durch Zeichenketten (Gödelnummer)
2. Die Fähigkeit eine andere TM mit geringem zusätzlichen Zeit- oder Platzbedarf zu simulieren (Universelle TM)

Grenzen der Diagonalisierung

Wiederholung Diagonalisierung

Was ist Diagonalisierung

Als Diagonalisierung wird (hier) ein Beweis bezeichnet, der nur auf den beiden folgenden Eigenschaften von TM aufbaut.

1. Die Existenz einer Repräsentation von TM durch Zeichenketten (Gödelnummer)
2. Die Fähigkeit eine andere TM mit geringem zusätzlichen Zeit- oder Platzbedarf zu simulieren (Universelle TM)

Grenzen der Diagonalisierung

Definition von Orakelmschinen

Definition Orakel-Turingmaschine

Eine Orakel-Turingmaschine M ist eine TM, die folgende zusätzliche Eigenschaften hat:

- ein spezielles zusätzliches Band (Orakelband) und 3 spezielle zusätzliche Zustände q_{query}, q_{yes}, q_{no}.
- ein Orakel $O \subset \{0,1\}^*$
- Wenn M den Zustand q_{query} betritt, ist der Folgezustand
  - q_{yes}, wenn für Inhalt s des Orakelbands gilt $s \in O$ und
  - q_{no}, wenn $s \notin O$
- Das Orakel liefert die Antwort in einem Berechnungsschritt

Grenzen der Diagonalisierung

Definition von Orakelmschinen

Definition Orakel-Turingmaschine

Eine Orakel-Turingmaschine M ist eine TM, die folgende zusätzliche Eigenschaften hat:

- ein spezielles zusätzliches Band (Orakelband) und 3 spezielle zusätzliche Zustände $q_{query}, q_{yes}, q_{no}$.
- ein Orakel $O \subset \{0,1\}^*$
- Wenn M den Zustand q_{query} betritt, ist der Folgezustand
  - q_{yes}, wenn für Inhalt s des Orakelbands gilt $s \in O$ und
  - q_{no}, wenn $s \notin O$
- Das Orakel liefert die Antwort in einem Berechnungsschritt

Grenzen der Diagonalisierung

Definition von Orakelmschinen

Definition Orakel-Turingmaschine

Eine Orakel-Turingmaschine M ist eine TM, die folgende zusätzliche Eigenschaften hat:

- ein spezielles zusätzliches Band (Orakelband) und 3 spezielle zusätzliche Zustände $q_{query}, q_{yes}, q_{no}$.
- ein Orakel $O \subset \{0, 1\}^*$
- Wenn M den Zustand q_{query} betritt, ist der Folgezustand
  - q_{yes}, wenn für Inhalt s des Orakelbands gilt $s \in O$ und
  - q_{no}, wenn $s \notin O$
- Das Orakel liefert die Antwort in einem Berechnungsschritt

Grenzen der Diagonalisierung

Definition von Orakelmschinen

Definition Orakel-Turingmaschine

Eine Orakel-Turingmaschine M ist eine TM, die folgende zusätzliche Eigenschaften hat:

- ein spezielles zusätzliches Band (Orakelband) und 3 spezielle zusätzliche Zustände $q_{query}, q_{yes}, q_{no}$.
- ein Orakel $O \subset \{0,1\}^*$
- Wenn M den Zustand q_{query} betritt, ist der Folgezustand
  - q_{yes}, wenn für Inhalt s des Orakelbands gilt $s \in O$ und
  - q_{no}, wenn $s \notin O$
- Das Orakel liefert die Antwort in einem Berechnungsschritt

Grenzen der Diagonalisierung

Definition von Orakelmschinen

Definition Orakel-Turingmaschine

Eine Orakel-Turingmaschine M ist eine TM, die folgende zusätzliche Eigenschaften hat:

- ein spezielles zusätzliches Band (Orakelband) und 3 spezielle zusätzliche Zustände $q_{query}, q_{yes}, q_{no}$.
- ein Orakel $O \subset \{0,1\}^*$
- Wenn M den Zustand q_{query} betritt, ist der Folgezustand
  - q_{yes}, wenn für Inhalt s des Orakelbands gilt $s \in O$ und
  - q_{no}, wenn $s \notin O$
- Das Orakel liefert die Antwort in einem Berechnungsschritt

Grenzen der Diagonalisierung

Satz von Baker-Gill-Solovay

Satz (Baker,Gill,Solovay, 75)

Es existieren Orakel A, B so dass $\mathbf{P}^A = \mathbf{NP}^A$ und $\mathbf{P}^B \neq \mathbf{NP}^B$

Fakultät für Informatik
Institut für Theoretische Informatik

Grenzen der Diagonalisierung

Satz von Baker-Gill-Solovay

Satz (Baker,Gill,Solovay, 75)

Es existieren Orakel A, B so dass $\mathbf{P}^A = \mathbf{NP}^A$ und $\mathbf{P}^B \neq \mathbf{NP}^B$

relativierende Beweise

Wir nennen einen Beweis, der auch für TM mit Orakel gilt, einen *relativierenden Beweis*

- Diagonalisierung ist relativierend und kann damit nicht für die $\mathbf{P} - \mathbf{NP}$ Frage genutzt werden.
- $\Rightarrow$ ein Beweis für die $\mathbf{P} - \mathbf{NP}$ Frage muss ein nicht relativierendes Verfahren nutzen !

Grenzen der Diagonalisierung

Satz von Baker-Gill-Solovay

Satz (Baker,Gill,Solovay, 75)

Es existieren Orakel A, B so dass $\mathbf{P}^A = \mathbf{NP}^A$ und $\mathbf{P}^B \neq \mathbf{NP}^B$

relativierende Beweise

Wir nennen einen Beweis, der auch für TM mit Orakel gilt, einen *relativierenden Beweis*

- Diagonalisierung ist relativierend und kann damit nicht für die $\mathbf{P} - \mathbf{NP}$ Frage genutzt werden.
- $\Rightarrow$ ein Beweis für die $\mathbf{P} - \mathbf{NP}$ Frage muss ein nicht relativierendes Verfahren nutzen !

Grenzen der Diagonalisierung

Beweis : $\mathbf{P}^A = \mathbf{NP}^A$

- $\mathbf{P}^A = \mathbf{NP}^A$ haben wir gerade schon gesehen: Nutze einfach das Orakel A = **EXPCOM**
- B zu konstruieren ist schwieriger (und interessanter!)

Grenzen der Diagonalisierung

Beweis : $\mathbf{P}^A = \mathbf{NP}^A$

- $\mathbf{P}^A = \mathbf{NP}^A$ haben wir gerade schon gesehen: Nutze einfach das Orakel A = **EXPCOM**
- B zu konstruieren ist schwieriger (und interessanter!)

Grenzen der Diagonalisierung

Beweis : $\mathbf{P}^B \neq \mathbf{NP}^B$

Definition unäre Sprache U_B

Für eine Sprache B sei $U_B = \{1^n$: Es gibt einen String der Länge n in B $\}$

- Wir sehen sofort ein : $U_B \in \mathbf{NP}^B$, da eine nicht det. TM ein Zertifikat raten kann.
- Müssen also nur noch B so konstruieren, dass $U_B \notin \mathbf{P}^B$

Grenzen der Diagonalisierung

Beweis : $\mathbf{P}^B \neq \mathbf{NP}^B$

Definition unäre Sprache U_B

Für eine Sprache B sei $U_B = \{1^n$: Es gibt einen String der Länge n in B $\}$

- Wir sehen sofort ein : $U_B \in \mathbf{NP}^B$, da eine nicht det. TM ein Zertifikat raten kann.
- Müssen also nur noch B so konstruieren, dass $U_B \notin \mathbf{P}^B$

Grenzen der Diagonalisierung

Beweis : $\mathbf{P}^B \neq \mathbf{NP}^B$

Definition unäre Sprache U_B

Für eine Sprache B sei $U_B = \{1^n$: Es gibt einen String der Länge n in B $\}$

- Wir sehen sofort ein : $U_B \in \mathbf{NP}^B$, da eine nicht det. TM ein Zertifikat raten kann.
- Müssen also nur noch B so konstruieren, dass $U_B \notin \mathbf{P}^B$

Grenzen der Diagonalisierung

Konstruktion von B

Wir konstruieren eine Folge von Sprachen $(B_i)_{i \in \mathbb{N}}$ so , dass $B = \lim_{n \to \infty} B_i$

- Wie stellen wir sicher, dass alle Turing Maschinen U_B nicht in polynomieller Zeit entscheiden können?
- Tipp: Die Menge aller Turing Maschinen ist abzählbar

Grenzen der Diagonalisierung

Konstruktion von B

Wir konstruieren eine Folge von Sprachen $(B_i)_{i \in \mathbb{N}}$ so , dass
$B = \lim_{n \to \infty} B_i$

- Wie stellen wir sicher, dass alle Turing Maschinen U_B nicht in polynomieller Zeit entscheiden können?
- Tipp: Die Menge aller Turing Maschinen ist abzählbar

Grenzen der Diagonalisierung

Konstruktion von B

- Genau : Wir iterieren über alle Turing Maschinen M_i und stellen sicher, dass M_i nicht in polynomieller Zeit U_B entscheiden kann
- Nutze dabei, dass die Anzahl der Wörter exponentiell in der Eingabelänge wächst

Grenzen der Diagonalisierung

Konstruktion von B

- Genau : Wir iterieren über alle Turing Maschinen M_i und stellen sicher, dass M_i nicht in polynomieller Zeit U_B entscheiden kann
- Nutze dabei, dass die Anzahl der Wörter exponentiell in der Eingabelänge wächst

Grenzen der Diagonalisierung

Konstruktion von B

Wir fangen an mit $B_0 = \emptyset$. Konstruktion fr̈ B_i :

- Wähle n so , dass n größer als alle Strings in B_{i-1}
- Lasse M_i auf Eingabe 1^n genau $2^n/10$ Schritte laufen (Beachte, dass M_i das Orakel B hat!)
-

Grenzen der Diagonalisierung

Konstruktion von B

$B_i = \{11011011, 10, 101, 111, 000111\}$

Orakel

B

Turing Maschine

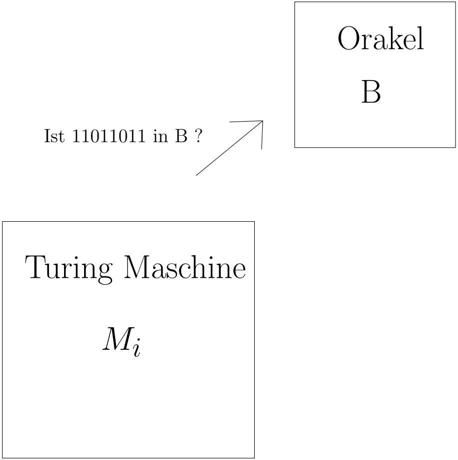
M_i

- Das Orakel antwortet konsistent auf dem bisherigen B_i
- Wir merken uns alle Strings der Länge n, die M_i an fragt!

Grenzen der Diagonalisierung

Konstruktion von B

$B_i = \{11011011, 10, 101, 111, 000111\}$

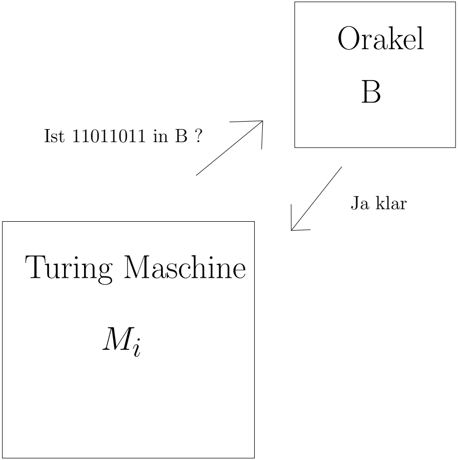


- Das Orakel antwortet konsistent auf dem bisherigen B_i
- Wir merken uns alle Strings der Länge n, die M_i an fragt!

Grenzen der Diagonalisierung

Konstruktion von B

$B_i = \{11011011, 10, 101, 111, 000111\}$



- Das Orakel antwortet konsistent auf dem bisherigen B_i
- Wir merken uns alle Strings der Länge n, die M_i an fragt!

Grenzen der Diagonalisierung

Konstruktion von B

$B_i = \{11011011, 10, 101, 111, 000111\}$

Orakel

B

Turing Maschine

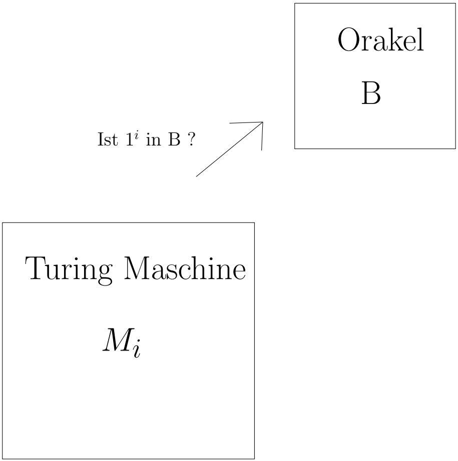
M_i

- Das Orakel antwortet konsistent auf dem bisherigen B_i
- Wir merken uns alle Strings der Länge n, die M_i an fragt!

Grenzen der Diagonalisierung

Konstruktion von B

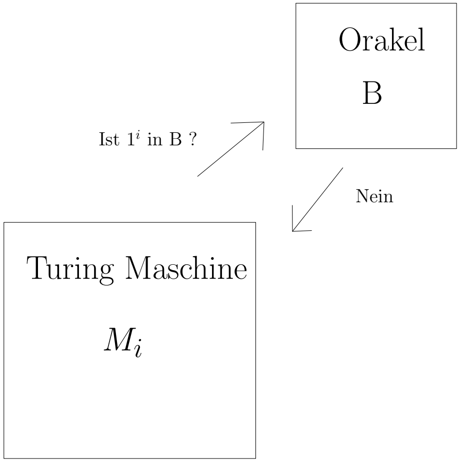
$B_i = \{11011011, 10, 101, 111, 000111\}$



- Das Orakel antwortet konsistent auf dem bisherigen B_i
- Wir merken uns alle Strings der Länge n, die M_i an fragt!

Grenzen der Diagonalisierung
Konstruktion von B

$B_i = \{11011011, 10, 101, 111, 000111\}$



- Das Orakel antwortet konsistent auf dem bisherigen B_i
- Wir merken uns alle Strings der Länge n, die M_i an fragt!

Grenzen der Diagonalisierung

Konstruktion von B

- Wir definieren nun B_{i+1} wie folgt :
- Wenn M_i nicht gehalten hat : $B_{i+1} = B_i$
- ansonsten :
  - M_i akzeptiert 1^n : Wir definieren, dass kein String der Länge n in B ist.
  - M_i lehnt ab : Wähle $x \in \{0, 1\}^n$, welches nicht von M_i an gefragt wurde und setze $B_{i+1} = B_i \cup \{x\}$
  - warum existiert dieses x?

Grenzen der Diagonalisierung
Konstruktion von B

- Wir definieren nun B_{i+1} wie folgt :
- Wenn M_i nicht gehalten hat : $B_{i+1} = B_i$
- ansonsten :
  - M_i akzeptiert 1^n : Wir definieren, dass kein String der Länge n in B ist.
  - M_i lehnt ab : Wähle $x \in \{0,1\}^n$, welches nicht von M_i an gefragt wurde und setze $B_{i+1} = B_i \cup \{x\}$
  - warum existiert dieses x?

Grenzen der Diagonalisierung

Konstruktion von B

- Wir definieren nun B_{i+1} wie folgt :
- Wenn M_i nicht gehalten hat : $B_{i+1} = B_i$
- ansonsten :
  - M_i akzeptiert 1^n : Wir definieren, dass kein String der Länge n in B ist.
  - M_i lehnt ab : Wähle $x \in \{0, 1\}^n$, welches nicht von M_i an gefragt wurde und setze $B_{i+1} = B_i \cup \{x\}$
  - warum existiert dieses x?

Grenzen der Diagonalisierung

Konstruktion von B

- Wir definieren nun B_{i+1} wie folgt :
- Wenn M_i nicht gehalten hat : $B_{i+1} = B_i$
- ansonsten :
  - M_i akzeptiert 1^n : Wir definieren, dass kein String der Länge n in B ist.
  - M_i lehnt ab : Wähle $x \in \{0, 1\}^n$, welches nicht von M_i an gefragt wurde und setze $B_{i+1} = B_i \cup \{x\}$
  - warum existiert dieses x?

Grenzen der Diagonalisierung

Konstruktion von B

- Wir definieren nun B_{i+1} wie folgt :
- Wenn M_i nicht gehalten hat : $B_{i+1} = B_i$
- ansonsten :
  - M_i akzeptiert 1^n : Wir definieren, dass kein String der Länge n in B ist.
  - M_i lehnt ab : Wähle $x \in \{0, 1\}^n$, welches nicht von M_i an gefragt wurde und setze $B_{i+1} = B_i \cup \{x\}$
  - warum existiert dieses x?

Grenzen der Diagonalisierung

Konstruktion von B

- Wir definieren nun B_{i+1} wie folgt :
- Wenn M_i nicht gehalten hat : $B_{i+1} = B_i$
- ansonsten :
  - M_i akzeptiert 1^n : Wir definieren, dass kein String der Länge n in B ist.
  - M_i lehnt ab : Wähle $x \in \{0,1\}^n$, welches nicht von M_i an gefragt wurde und setze $B_{i+1} = B_i \cup \{x\}$
  - warum existiert dieses x?

Grenzen der Diagonalisierung

Beweis Schluss

- Haben oben ein gesehen, dass $U_B \in \mathbf{NP}^B$
- Und für jede polynomiell beschränkte TM M existiert ein i,so dass
  - M = M_i
  - M auf der Eingabe 1^i weniger als $2^i/10$ Schritte benötigt
  - und damit M_i nach Konstruktion die Frage $1^i \in U_B$ falsch beantwortet
- $\Rightarrow U_B \notin \mathbf{P}^B$ und damit $P^B \neq \mathbf{NP}^B$ $\square$

Grenzen der Diagonalisierung

Beweis Schluss

- Haben oben ein gesehen, dass $U_B \in \mathbf{NP}^B$
- Und für jede polynomiell beschränkte TM M existiert ein i,so dass
  - M = M_i
  - M auf der Eingabe 1^i weniger als $2^i/10$ Schritte benötigt
  - und damit M_i nach Konstruktion die Frage $1^i \in U_B$ falsch beantwortet
- $\Rightarrow U_B \notin \mathbf{P}^B$ und damit $P^B \neq \mathbf{NP}^B$ □

Corvin Paul, Matthias Schimek:

Fakultät für Informatik
Institut für Theoretische Informatik

Grenzen der Diagonalisierung

Beweis Schluss

- Haben oben ein gesehen, dass $U_B \in \mathbf{NP}^B$
- Und für jede polynomiell beschränkte TM M existiert ein i,so dass
  - M = M_i
  - M auf der Eingabe 1^i weniger als $2^i/10$ Schritte benötigt
  - und damit M_i nach Konstruktion die Frage $1^i \in U_B$ falsch beantwortet
- $\Rightarrow U_B \notin \mathbf{P}^B$ und damit $P^B \neq \mathbf{NP}^B$ $\square$

Grenzen der Diagonalisierung

Beweis Schluss

- Haben oben ein gesehen, dass $U_B \in \mathbf{NP}^B$
- Und für jede polynomiell beschränkte TM M existiert ein i,so dass
  - M = M_i
  - M auf der Eingabe 1^i weniger als $2^i / 10$ Schritte benötigt
  - und damit M_i nach Konstruktion die Frage $1^i \in U_B$ falsch beantwortet
- $\Rightarrow U_B \notin \mathbf{P}^B$ und damit $P^B \neq \mathbf{NP}^B$ $\square$

Grenzen der Diagonalisierung

Beweis Schluss

- Haben oben ein gesehen, dass $U_B \in \mathbf{NP}^B$
- Und für jede polynomiell beschränkte TM M existiert ein i,so dass
  - M = M_i
  - M auf der Eingabe 1^i weniger als $2^i/10$ Schritte benötigt
  - und damit M_i nach Konstruktion die Frage $1^i \in U_B$ falsch beantwortet
- $\Rightarrow U_B \notin \mathbf{P}^B$ und damit $P^B \neq \mathbf{NP}^B$ □

Grenzen der Diagonalisierung

Beweis Schluss

- Haben oben ein gesehen, dass $U_B \in \mathbf{NP}^B$
- Und für jede polynomiell beschränkte TM M existiert ein i,so dass
  - M = M_i
  - M auf der Eingabe 1^i weniger als $2^i/10$ Schritte benötigt
  - und damit M_i nach Konstruktion die Frage $1^i \in U_B$ falsch beantwortet
- $\Rightarrow U_B \notin \mathbf{P}^B$ und damit $P^B \neq \mathbf{NP}^B$ $\square$

hier muss noch die Definition von **PH** rein ;)

Die Klasse PH
Eigenschaften von PH

- Vermutung: $\mathbf{P} \neq \mathbf{NP}$ und $\mathbf{NP} \neq \mathbf{coNP}$
- Verallgemeinerung: $\sum_i^p \subsetneq \sum_{i+1}^p$ für alle i
- "The polynomial hierarchy does not collapse"

Satz Kollaps von PH und Auswirkungen auf P – NP

- Für alle $i \geq 0$ gilt: $\sum_i^p = \prod_i^p \Rightarrow \mathbf{PH} = \sum_i^p$
- Wenn $\mathbf{P} = \mathbf{NP}$, dann folgt $\mathbf{PH} = \mathbf{P}$

Die Klasse PH

Eigenschaften von PH

- Vermutung: $\mathbf{P} \neq \mathbf{NP}$ und $\mathbf{NP} \neq \mathbf{coNP}$
- Verallgemeinerung: $\sum_i^p \subsetneq \sum_{i+1}^p$ für alle i
- "The polynomial hierarchy does not collapse"

Satz Kollaps von PH und Auswirkungen auf $\mathbf{P} = \mathbf{NP}$

- Für alle $i \geq 0$ gilt: $\sum_i^p = \prod_i^p \Rightarrow \mathbf{PH} = \sum_i^p$
- Wenn $\mathbf{P} = \mathbf{NP}$, dann folgt $\mathbf{PH} = \mathbf{P}$

Die Klasse PH
Eigenschaften von PH

- Vermutung: $\mathbf{P} \neq \mathbf{NP}$ und $\mathbf{NP} \neq \mathbf{coNP}$
- Verallgemeinerung: $\sum_i^p \subsetneq \sum_{i+1}^p$ für alle i
- "The polynomial hierarchy does not collapse"

Satz Kollaps von PH und Auswirkungen auf P – NP

- Für alle $i \geq 0$ gilt: $\sum_i^p = \prod_i^p \Rightarrow \mathbf{PH} = \sum_i^p$
- Wenn $\mathbf{P} = \mathbf{NP}$, dann folgt $\mathbf{PH} = \mathbf{P}$

Die Klasse PH
Eigenschaften von PH

- Vermutung: $\mathbf{P} \neq \mathbf{NP}$ und $\mathbf{NP} \neq \mathbf{coNP}$
- Verallgemeinerung: $\sum_i^p \subsetneq \sum_{i+1}^p$ für alle i
- "The polynomial hierarchy does not collapse"

Satz Kollaps von PH und Auswirkungen auf P – NP

- Für alle $i \geq 0$ gilt: $\sum_i^p = \prod_i^p \Rightarrow \mathbf{PH} = \sum_i^p$
- Wenn $\mathbf{P} = \mathbf{NP}$, dann folgt $\mathbf{PH} = \mathbf{P}$

Die Klasse PH

Eigenschaften von PH

- Vermutung: $\mathbf{P} \neq \mathbf{NP}$ und $\mathbf{NP} \neq \mathbf{coNP}$
- Verallgemeinerung: $\sum_i^p \subsetneq \sum_{i+1}^p$ für alle i
- "The polynomial hierarchy does not collapse"

Satz Kollaps von PH und Auswirkungen auf P – NP

1. Für alle $i \geq 0$ gilt: $\sum_i^p = \prod_i^p \quad \Rightarrow \quad \mathbf{PH} = \sum_i^p$
2. Wenn $\mathbf{P} = \mathbf{NP}$, dann folgt $\mathbf{PH} = \mathbf{P}$

Die Klasse PH
Eigenschaften von PH

- Vermutung: $\mathbf{P} \neq \mathbf{NP}$ und $\mathbf{NP} \neq \mathbf{coNP}$
- Verallgemeinerung: $\sum_i^p \subsetneq \sum_{i+1}^p$ für alle i
- "The polynomial hierarchy does not collapse"

Satz Kollaps von PH und Auswirkungen auf P – NP

1. Für alle $i \geq 0$ gilt: $\sum_i^p = \prod_i^p \quad \Rightarrow \quad \mathbf{PH} = \sum_i^p$
2. Wenn $\mathbf{P} = \mathbf{NP}$, dann folgt $\mathbf{PH} = \mathbf{P}$

Die Klasse PH

Beweis von 2.

Beweis von $\mathbf{P} = \mathbf{NP} \Rightarrow \mathbf{PH} = \mathbf{P}$

- Sei $\mathbf{P} = \mathbf{NP}$, beweisen über Induktion $\sum_i^p, \prod_i^p \subset \mathbf{P}$

hallo