Karlsruher Institut für Technologie

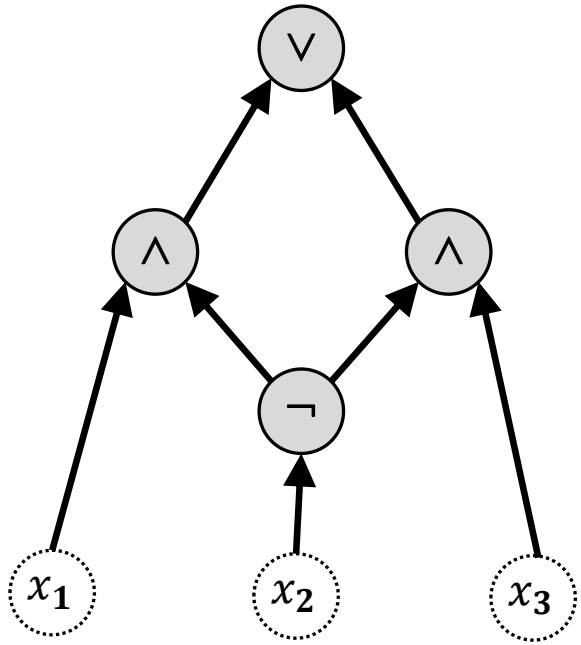
Boolean circuits: ein alternatives Rechenmodell

Simon Bischof, Philipp Christian Loewner

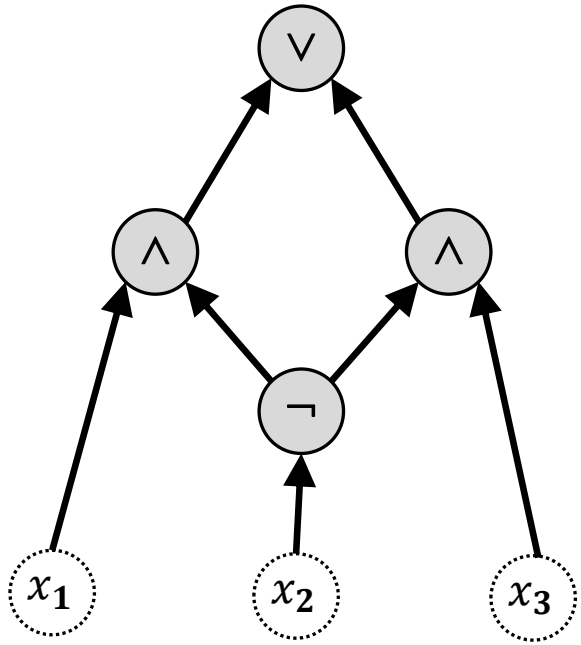
Institut für Theoretische Informatik

Motivation

Mein erster Boolean Circuit

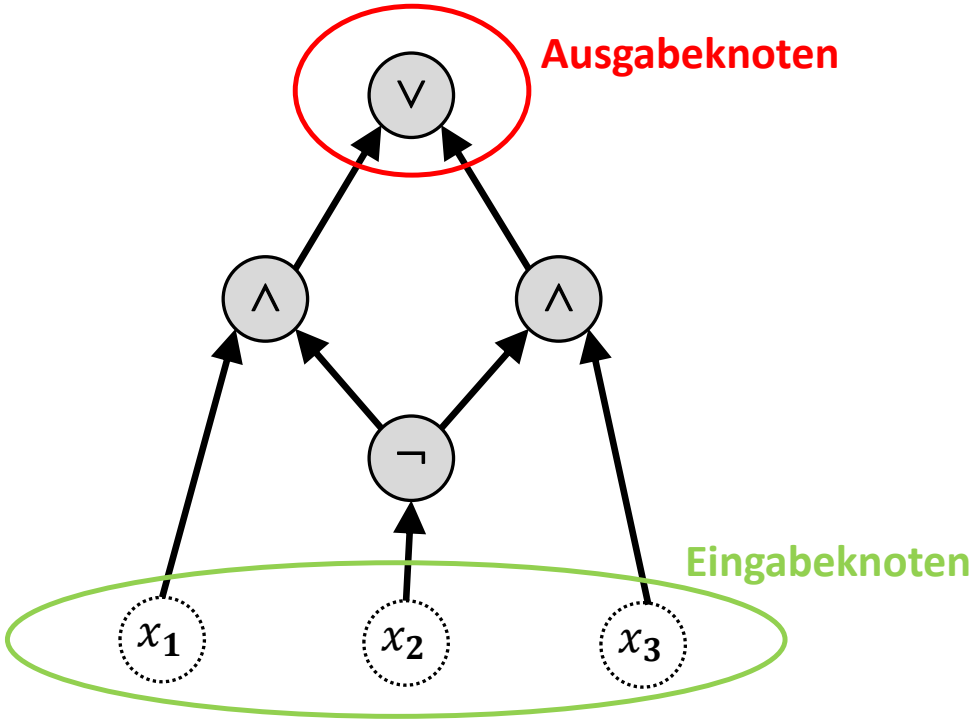


Mein erster Boolean Circuit



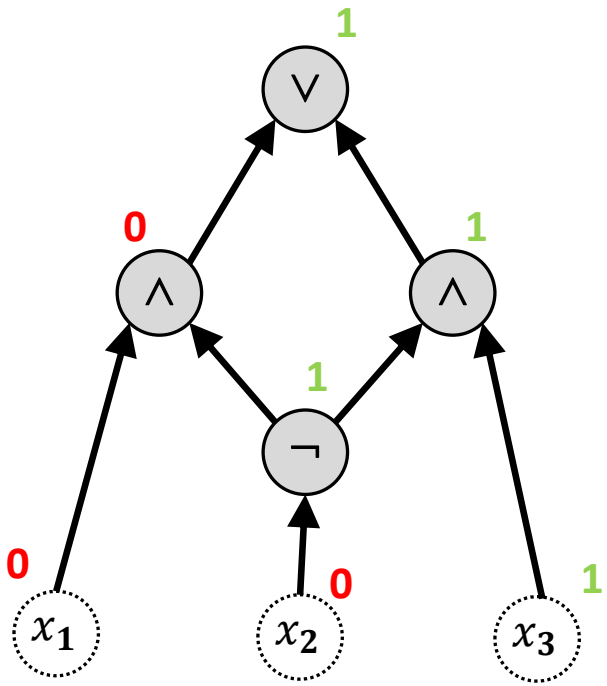
$$C(x_1, x_2, x_3) = x_1\overline{x_2} \vee \overline{x_2}x_3$$

Mein erster Boolean Circuit



$$C(x_1, x_2, x_3) = x_1\overline{x_2} \vee \overline{x_2}x_3$$

Mein erster Boolean Circuit



Auswertung für $x := (x_1, x_2, x_3) = (0, 0, 1)$ ergibt $C(x) = 1$

Eigenschaften

Boolean Circuits entsprechen Schaltnetzen, aber **nicht** Schaltwerken - müssen azyklisch sein

Boolean Circuits sind Boolesche Formeln, wenn jeder Knoten höchstens Ausgangsgrad 1 hat.

Benutzung für Sprachen

Idee: Benutzung zur Erkennung von Wörtern

- Circuit entscheidet Wörter seiner Eingabelänge
- $\Rightarrow$ ein Circuit pro Eingabegröße
- Folge von Circuits pro Sprache

Benutzung für Sprachen

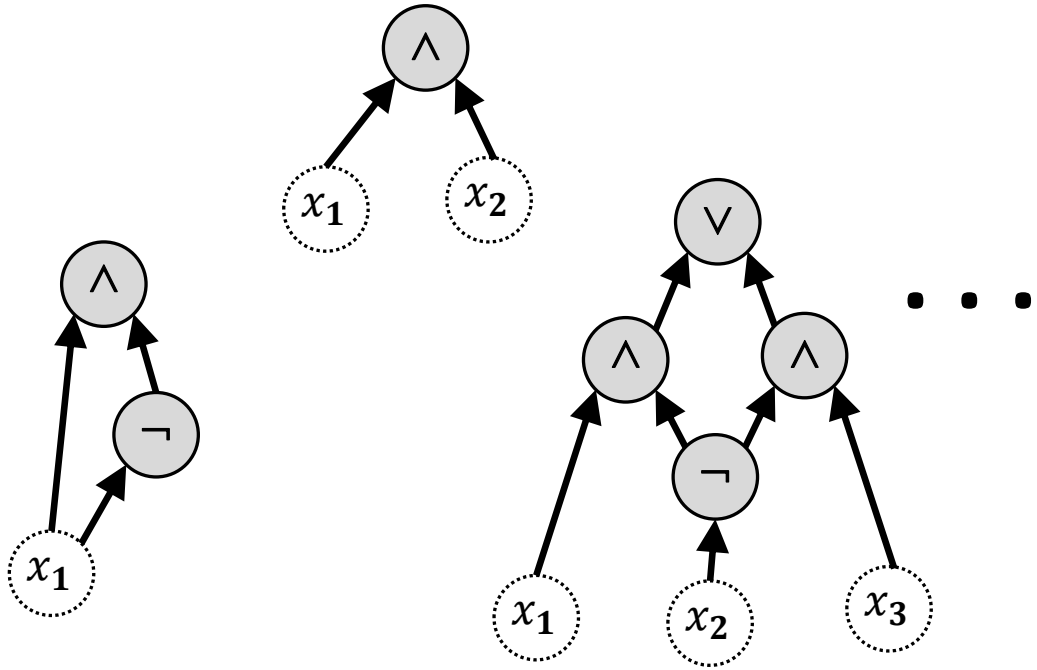
Idee: Benutzung zur Erkennung von Wörtern

- Circuit entscheidet Wörter seiner Eingabelänge
- $\Rightarrow$ ein Circuit pro Eingabegröße
- Folge von Circuits pro Sprache

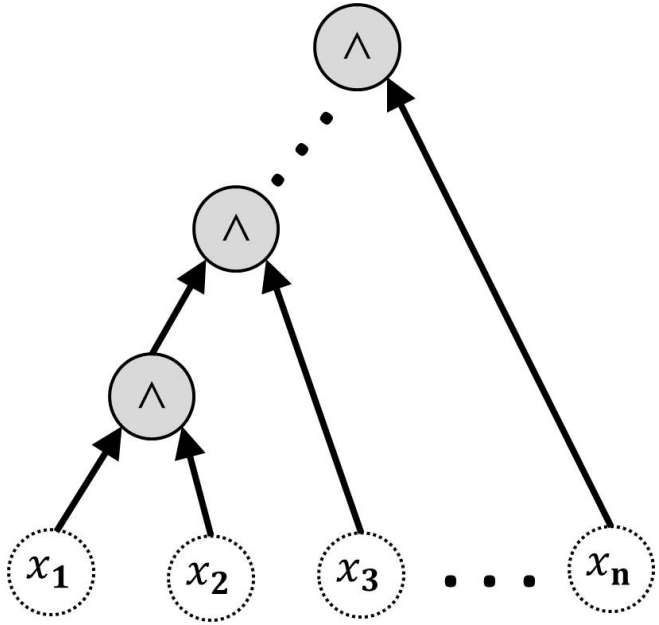
Folge $\{C_n\}_{n \in \mathbb{N}}$ von Circuits heißt **Circuit-Familie**.

Dabei hat jeder Circuit C_n die Eingabegröße n

Beispiel Circuit-Familie



Beispiel Circuit-Familie



$$L := \{1^n \mid n \in \mathbb{N}\}$$

Interessante Eigenschaften

- Boolean Circuits können alle Sprachen entscheiden die TMs entscheiden können
- Turingmaschinen können Boolean Circuits simulieren

Interessante Eigenschaften

- Boolean Circuits können alle Sprachen entscheiden die TMs entscheiden können
- Turingmaschinen können Boolean Circuits simulieren
- **Boolean Circuits sind aber mächtiger als TMs**

Interessante Eigenschaften

- Boolean Circuits können alle Sprachen entscheiden die TMs entscheiden können
- Turingmaschinen können Boolean Circuits simulieren
- Boolean Circuits sind aber mächtiger als TMs
  - betrachte unäre Sprachen
  - eindeutiges Wort w_n für jede Wortlänge

Interessante Eigenschaften

- Boolean Circuits können alle Sprachen entscheiden die TMs entscheiden können
- Turingmaschinen können Boolean Circuits simulieren
- Boolean Circuits sind aber mächtiger als TMs
  - betrachte unäre Sprachen
  - eindeutiges Wort w_n für jede Wortlänge
  - $\Rightarrow w_n \in L$ kann hart in C_n eincodiert werden

Interessante Eigenschaften

- Boolean Circuits können alle Sprachen entscheiden die TMs entscheiden können
- Turingmaschinen können Boolean Circuits simulieren
- Boolean Circuits sind aber mächtiger als TMs
  - betrachte unäre Sprachen
  - eindeutiges Wort w_n für jede Wortlänge
  - $\Rightarrow w_n \in L$ kann hart in C_n eincodiert werden
  - $\Rightarrow$ Circuit-Familien können **unentscheidbare** Sprachen entscheiden
  - Beispiel: $\{1^n : M_n \text{ akzeptiert } 1^n\}$

Weitere Vorgehensweise

- Definition von Komplexitätsklassen
- Einführung sinnvoller Beschränkungen
- Wozu kann man Circuits denn nun verwenden?

Weitere Vorgehensweise

- Definition von Komplexitätsklassen
- Einführung sinnvoller Beschränkungen
- Wozu kann man Circuits denn nun verwenden?

Karlsruher Institut für Technologie

$\mathcal{P}_{/poly}$ und $\mathcal{P}$

Boolean circuits - Größe

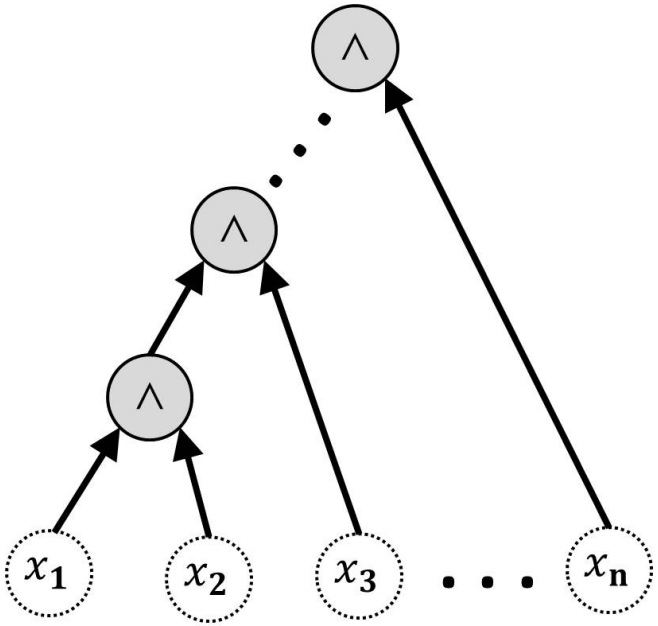
Größe eines **einzelnen Circuits** C_n: Skalar
$|C_n| :=$ Anzahl der Knoten in C_n

Größe einer **Familie** $\{C_n\}_{n\in\mathbb{N}}$: Funktion
$T(n)$, wobei $|C_n| \leq T(n) \quad (n \in \mathbb{N})$

Klassifizierung einer **Sprache** $L \subseteq \{0,1\}^+$
$L \in \mathbf{SIZE}(T(n))$, falls eine Circuit-Familie der Größe $T(n)$ existiert die genau L entscheidet

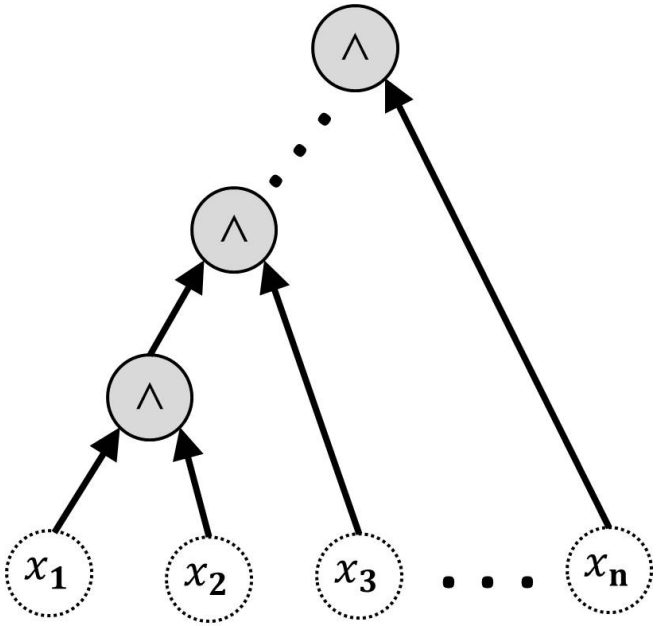
Anwendung auf vorige Beispiele

$L := \{1^n \mid n \in \mathbb{N}\}$:



Anwendung auf vorige Beispiele

$L := \{1^n \mid n \in \mathbb{N}\}$:



Folgerung: $L \in \mathbf{SIZE}(2n)$

Anwendung auf vorige Beispiele

Sei $n \in \mathbb{N}$, $f : \{0,1\}^n \to \{0,1\}$ beliebig.

Anwendung auf vorige Beispiele

Sei $n \in \mathbb{N}$, $f : \{0,1\}^n \to \{0,1\}$ beliebig.

- f lässt sich in konjunktiver Normalform darstellen

Anwendung auf vorige Beispiele

Sei $n \in \mathbb{N}$, $f : \{0,1\}^n \to \{0,1\}$ beliebig.

- f lässt sich in konjunktiver Normalform darstellen
- Circuit dafür hat höchstens $n \cdot 2^n$ Knoten

Anwendung auf vorige Beispiele

Sei $n \in \mathbb{N}$, $f : \{0,1\}^n \to \{0,1\}$ beliebig.

- f lässt sich in konjunktiver Normalform darstellen
- Circuit dafür hat höchstens $n \cdot 2^n$ Knoten
- Folgerung: $\forall L \subseteq \{0,1\}^+ : L \in \mathbf{SIZE}(n \cdot 2^n)$

Die Klasse $\mathcal{P}_{/poly}$

Bei Turingmaschinen haben wir mit $\mathcal{P}$ effizient lösbare Sprachen betrachtet.

Die Klasse $\mathcal{P}_{/poly}$

Bei Turingmaschinen haben wir mit $\mathcal{P}$ effizient lösbare Sprachen betrachtet.
Wir untersuchen nun, welche Komplexitätsklassen Boolean Circuits bei der Einschränkung auf “kleine“ Circuits liefern.

Die Klasse $\mathcal{P}_{/poly}$

Bei Turingmaschinen haben wir mit $\mathcal{P}$ effizient lösbare Sprachen betrachtet.
Wir untersuchen nun, welche Komplexitätsklassen Boolean Circuits bei der Einschränkung auf “kleine“ Circuits liefern.

- $\mathcal{P}_{/poly} := \bigcup_c \mathbf{SIZE}(n^c)$

Die Klasse $\mathcal{P}_{/poly}$

Bei Turingmaschinen haben wir mit $\mathcal{P}$ effizient lösbare Sprachen betrachtet.
Wir untersuchen nun, welche Komplexitätsklassen Boolean Circuits bei der Einschränkung auf “kleine“ Circuits liefern.

- $\mathcal{P}_{/poly} := \bigcup_c \mathbf{SIZE}(n^c)$
- Klasse der durch polynomiell große Familien von Circuits entscheidbaren Sprachen

Die Klasse $\mathcal{P}_{/poly}$

Bei Turingmaschinen haben wir mit $\mathcal{P}$ effizient lösbare Sprachen betrachtet.
Wir untersuchen nun, welche Komplexitätsklassen Boolean Circuits bei der Einschränkung auf “kleine“ Circuits liefern.

- $\mathcal{P}_{/poly} := \bigcup_c \mathbf{SIZE}(n^c)$
- Klasse der durch polynomiell große Familien von Circuits entscheidbaren Sprachen
- Satz: es gilt $\mathcal{P} \subsetneq \mathcal{P}_{/poly}$.

Begründung für $\mathcal{P} \subsetneq \mathcal{P}_{/poly}$

Teil 1: $\mathcal{P} \neq \mathcal{P}_{/poly}$

- Erinnerung: $L := \{1^n : M_n \text{ akzeptiert } 1^n\}$ unentscheidbar
- jede unäre Sprache hat Circuit-Familie linearer Größe
- $\Rightarrow L \in SIZE(\mathcal{O}(n))$
- offensichtlich aber $L \notin \mathcal{P}$

Hinweis: Problem der Unentscheidbarkeit ist eigentlich nur verlagert in die Generierung der Circuit-Familie.
Im nächsten Kapitel: Beschränkung auf eine Klasse die gleich $\mathcal{P}$ ist.

Begründung für $\mathcal{P} \subsetneq \mathcal{P}_{/poly}$

Teil 2: $\mathcal{P} \subseteq \mathcal{P}_{/poly}$

Jede Sprache $L \in \mathcal{P}$ lässt sich in polynomieller Zeit von einer TM M entscheiden.

Wie können wir das auf Circuits übertragen?

Probleme:

- TM ändert Bandinhalt
- TM trifft Entscheidungen basierend auf Bandinhalt
- Circuits können ihre Eingabe nicht verändern
- Betrachtung der Größe von Circuits, aber Laufzeit von TMs

Begründung für $\mathcal{P} \subsetneq \mathcal{P}_{/poly}$

Teil 2: $\mathcal{P} \subseteq \mathcal{P}_{/poly}$

Jede Sprache $L \in \mathcal{P}$ lässt sich in polynomieller Zeit von einer TM M entscheiden.

Wie können wir das auf Circuits übertragen?

Probleme:

- TM ändert Bandinhalt
- TM trifft Entscheidungen basierend auf Bandinhalt
- Circuits können ihre Eingabe nicht verändern
- Betrachtung der Größe von Circuits, aber Laufzeit von TMs

Lösung: Betrachtung von **oblivious** TMs

Begründung für $\mathcal{P} \subsetneq \mathcal{P}_{/poly}$

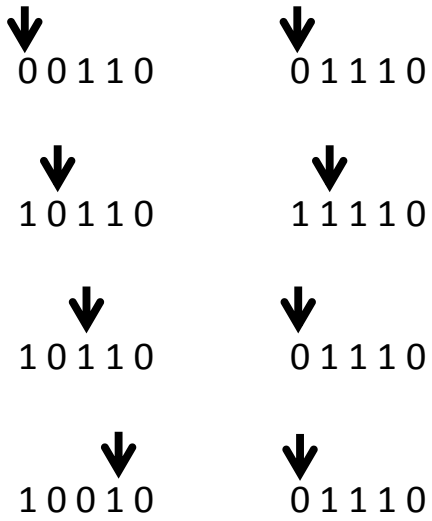
Teil 2: $\mathcal{P} \subseteq \mathcal{P}_{/poly}$

- $L \in \mathcal{P}$
- $\Rightarrow$ ex. TM, die L in polynomieller Zeit entscheidet
- $\Rightarrow$ ex. oblivious TM, die L in polynomieller Zeit entscheidet
- $\Rightarrow$ ex. Circuit-Familie polynomieller Größe, die L entscheidet

Exkurs: Oblivious TM

Normale Turingmaschine

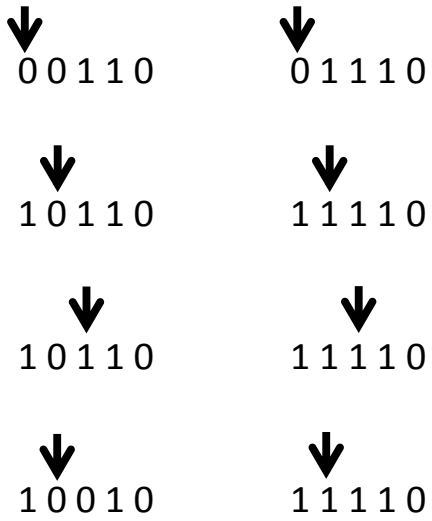
Position auf Band abhängig von Eingabe und Schrittnummer



Exkurs: Oblivious TM

Oblivious Turingmaschine

Position auf Band nur abhängig von Eingabelänge und Schrittnummer



Begründung für $\mathcal{P} \subsetneq \mathcal{P}_{/poly}$

Karlsruher Institut für Technologie

Teil 2: $\mathcal{P} \subseteq \mathcal{P}_{/poly}$

- $L \in \mathcal{P}$
- $\Rightarrow$ ex. TM, die L in polynomieller Zeit entscheidet
- $\Rightarrow$ ex. oblivious TM, die L in polynomieller Zeit entscheidet
- $\Rightarrow$ ex. Circuit-Familie polynomieller Größe, die L entscheidet

TM M_1 existiert mit pol. Laufzeit $T(n)$

- erstelle daraus eine oblivious TM M_2
- $\Rightarrow$ Laufzeit in $\mathcal{O}(T^2(n))$
- $\Rightarrow$ also insgesamt polynomielle Laufzeit

Begründung für $\mathcal{P} \subsetneq \mathcal{P}_{/poly}$

Teil 2: $\mathcal{P} \subseteq \mathcal{P}_{/poly}$

- $L \in \mathcal{P}$
- $\Rightarrow$ ex. TM, die L in polynomieller Zeit entscheidet
- $\Rightarrow$ ex. oblivious TM, die L in polynomieller Zeit entscheidet
- $\Rightarrow$ ex. Circuit-Familie polynomieller Größe, die L entscheidet

Begründung für $\mathcal{P} \subsetneq \mathcal{P}_{/poly}$

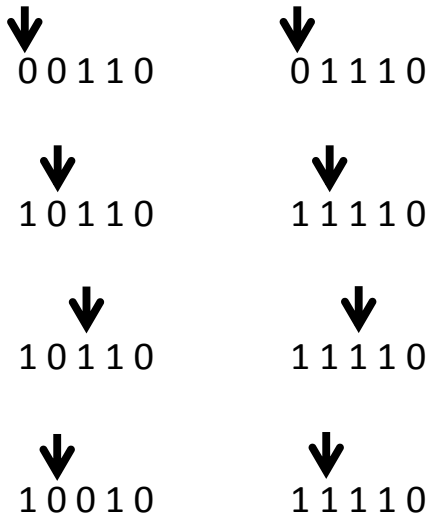
Teil 2: $\mathcal{P} \subseteq \mathcal{P}_{/poly}$

- $L \in \mathcal{P}$
- $\Rightarrow$ ex. TM, die L in polynomieller Zeit entscheidet
- $\Rightarrow$ ex. oblivious TM, die L in polynomieller Zeit entscheidet
- $\Rightarrow$ ex. Circuit-Familie polynomieller Größe, die L entscheidet

oblivious TM M_2 existiert mit pol. Laufzeit

- feste Laufzeit pro Eingabegröße
- akzeptiert falls im letzten Schritt erreichter Zustand akzeptierend ist
- Idee: Codiere Bandalphabet und Zustandsmenge in Knoten

Begründung für $\mathcal{P} \subsetneq \mathcal{P}_{/poly}$



$z_i = \begin{pmatrix} Zustand\ nach\ Schritt\ i \\ geschriebenes\ Zeichen \end{pmatrix}$

z.B. $z_1 = \begin{pmatrix} q_2 \\ 1 \end{pmatrix}$

Begründung für $\mathcal{P} \subsetneq \mathcal{P}_{/poly}$

Karlsruher Institut für Technologie

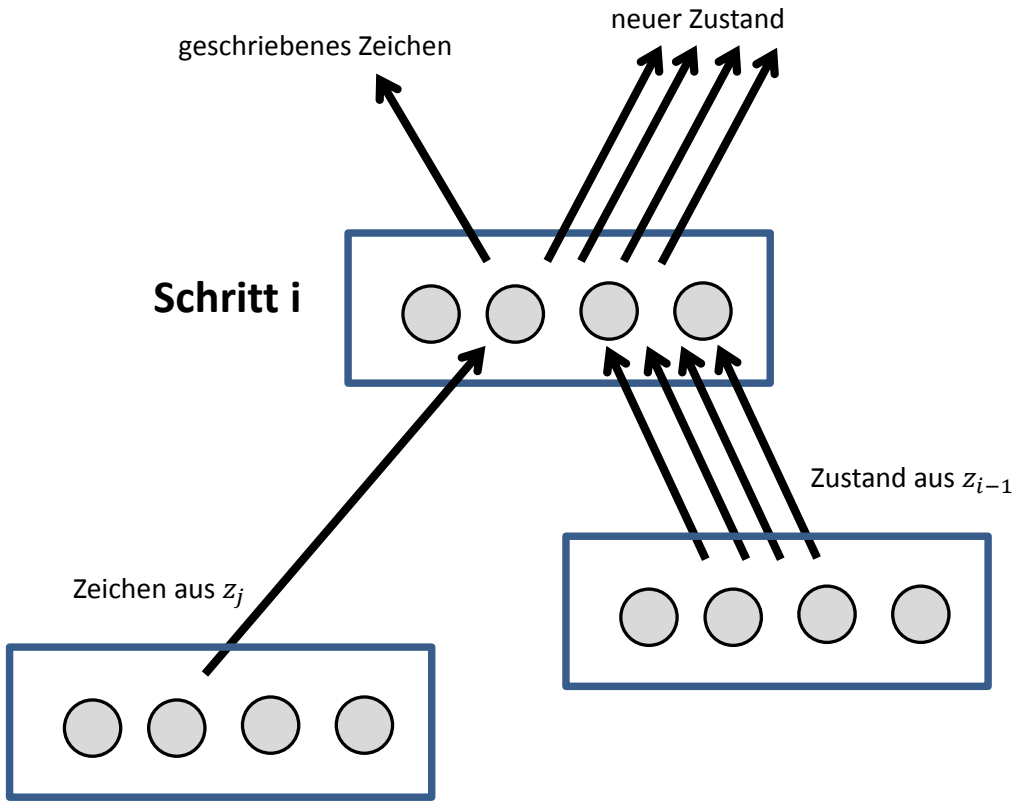
Teil 2: $\mathcal{P} \subseteq \mathcal{P}_{/poly}$

- $L \in \mathcal{P}$
- $\Rightarrow$ ex. TM, die L in polynomieller Zeit entscheidet
- $\Rightarrow$ ex. oblivious TM, die L in polynomieller Zeit entscheidet
- $\Rightarrow$ ex. Circuit-Familie polynomieller Größe, die L entscheidet

Idee: Codiere Bandalphabet und Zustandsmenge in Knoten

- konstantes Alphabet, konstante Zustandsmenge
- $\Rightarrow$ Anzahl der Knoten für beliebige Operationen ist konstant beschränkt
- $\Rightarrow$ jeder Berechnungsschritt lässt sich durch einen Teilcircuit konstant beschränkter Größe darstellen
- $\Rightarrow$ für polynomielle Anzahl Schritte: Polynomielle Anzahl von Knoten

Begründung für $\mathcal{P} \subsetneq \mathcal{P}_{/poly}$



Weitere Vorgehensweise

- Definition von Komplexitätsklassen
- Einführung sinnvoller Beschränkungen
- Wozu kann man Circuits denn nun verwenden?

Karlsruher Institut für Technologie

Uniformität

Motivation zur Uniformität

Wie können wir

- boolean circuits einschränken um sie auf die Mächtigkeit von TMs zu bringen?
- TMs erweitern um sie auf die Mächtigkeit von boolean circuits zu bringen?

$\mathcal{P}$-uniform, logspace-uniform

- Eine Familie $\{C_n\}$ von Circuits heißt $\mathcal{P}$-uniform, wenn es eine polynomielle Turingmaschine gibt, die aus der Eingabe 1^n die Beschreibung von C_n berechnet.

$\mathcal{P}$-uniform, logspace-uniform

- Eine Familie $\{C_n\}$ von Circuits heißt $\mathcal{P}$-uniform, wenn es eine polynomielle Turingmaschine gibt, die aus der Eingabe 1^n die Beschreibung von C_n berechnet.
- Durch die Einschränkung auf $\mathcal{P}$-uniforme Circuits wird $\mathcal{P}_{/poly}$ zu $\mathcal{P}$.

$\mathcal{P}$-uniform, logspace-uniform

- Eine Familie $\{C_n\}$ von Circuits heißt $\mathcal{P}$-uniform, wenn es eine polynomielle Turingmaschine gibt, die aus der Eingabe 1^n die Beschreibung von C_n berechnet.
- Durch die Einschränkung auf $\mathcal{P}$-uniforme Circuits wird $\mathcal{P}_{/poly}$ zu $\mathcal{P}$.

- Eine Circuit-Familie heißt logspace-uniform, wenn die Funktionen

$\mathcal{P}$-uniform, logspace-uniform

- Eine Familie $\{C_n\}$ von Circuits heißt $\mathcal{P}$-uniform, wenn es eine polynomielle Turingmaschine gibt, die aus der Eingabe 1^n die Beschreibung von C_n berechnet.
- Durch die Einschränkung auf $\mathcal{P}$-uniforme Circuits wird $\mathcal{P}_{/poly}$ zu $\mathcal{P}$.

- Eine Circuit-Familie heißt logspace-uniform, wenn die Funktionen
  - $SIZE(n) = |C_n|$
  - $TYPE(n, i)$ (Typ des i-ten Knoten von C_n)
  - $EDGE(n, i, j)$ (1 falls es eine Kante vom i-ten zum j-ten Knoten von C_n gibt)

$\mathcal{P}$-uniform, logspace-uniform

- Eine Familie $\{C_n\}$ von Circuits heißt $\mathcal{P}$-uniform, wenn es eine polynomielle Turingmaschine gibt, die aus der Eingabe 1^n die Beschreibung von C_n berechnet.
- Durch die Einschränkung auf $\mathcal{P}$-uniforme Circuits wird $\mathcal{P}_{/poly}$ zu $\mathcal{P}$.

- Eine Circuit-Familie heißt logspace-uniform, wenn die Funktionen
  - $SIZE(n) = |C_n|$
  - $TYPE(n, i)$ (Typ des i-ten Knoten von C_n)
  - $EDGE(n, i, j)$ (1 falls es eine Kante vom i-ten zum j-ten Knoten von C_n gibt)

  im Sinne der gängigen Definition von logspace berechenbar sind.

$\mathcal{P}$-uniform, logspace-uniform

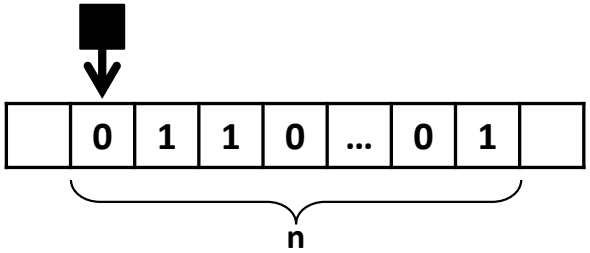
- Eine Familie $\{C_n\}$ von Circuits heißt $\mathcal{P}$-uniform, wenn es eine polynomielle Turingmaschine gibt, die aus der Eingabe 1^n die Beschreibung von C_n berechnet.
- Durch die Einschränkung auf $\mathcal{P}$-uniforme Circuits wird $\mathcal{P}_{/poly}$ zu $\mathcal{P}$.

- Eine Circuit-Familie heißt logspace-uniform, wenn die Funktionen
  - $SIZE(n) = |C_n|$
  - $TYPE(n, i)$ (Typ des i-ten Knoten von C_n)
  - $EDGE(n, i, j)$ (1 falls es eine Kante vom i-ten zum j-ten Knoten von C_n gibt)

  im Sinne der gängigen Definition von logspace berechenbar sind. Logspace-Uniformität ist zusätzlich zu $\mathcal{P}$-Uniformität keine echte Einschränkung.
- Eine Sprache L besitzt polynomiell große, logspace-uniforme Circuits $\Leftrightarrow L \in \mathcal{P}$.

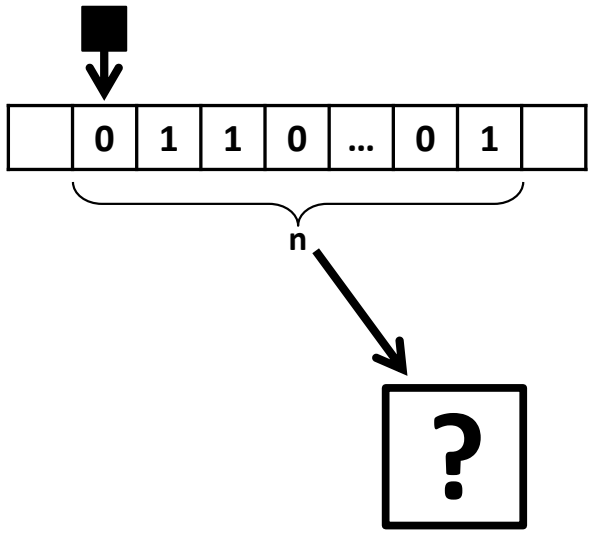
Turingmaschinen mit advice strings

Normale TM



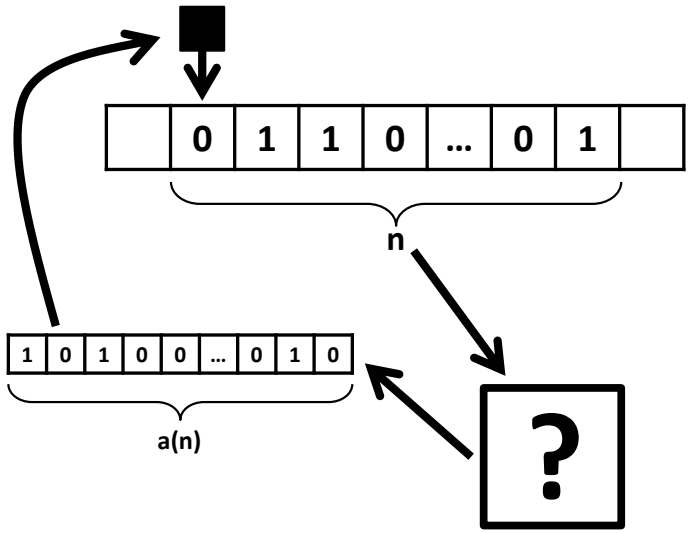
Turingmaschinen mit advice strings

Advice TM fragt Blackbox



Turingmaschinen mit advice strings

Advice TM erhält advice von Blackbox



Turingmaschinen mit advice strings

Bemerkungen:

1. jede unäre Sprache lässt sich mit einem bit advice in linearer Zeit entscheiden
2. $\mathcal{P}_{/poly}$: Sprachen, die mit polynomiellem advice in polynomieller Zeit entscheidbar sind

Die Sprache CKT-SAT

- Ein Wort c ist in CKT-SAT, wenn der durch c beschriebene Circuit C eine erfüllende Belegung besitzt.

Die Sprache CKT-SAT

- Ein Wort c ist in CKT-SAT, wenn der durch c beschriebene Circuit C eine erfüllende Belegung besitzt.
- CKT-SAT$\in \mathcal{NP}$.

Die Sprache CKT-SAT

- Ein Wort c ist in CKT-SAT, wenn der durch c beschriebene Circuit C eine erfüllende Belegung besitzt.
- CKT-SAT$\in \mathcal{NP}$. (Warum?)

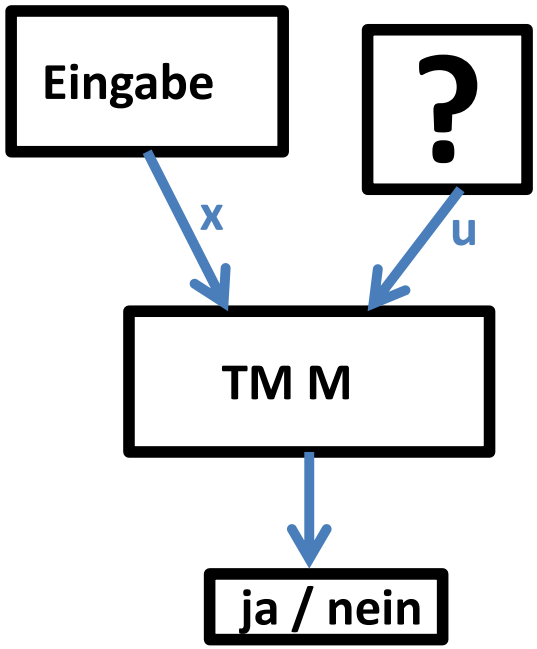
Die Sprache CKT-SAT

- Ein Wort c ist in CKT-SAT, wenn der durch c beschriebene Circuit C eine erfüllende Belegung besitzt.
- CKT-SAT$\in \mathcal{NP}$. (Warum?)
- CKT-SAT ist auch $\mathcal{NP}$-schwer, also $\mathcal{NP}$-vollständig.

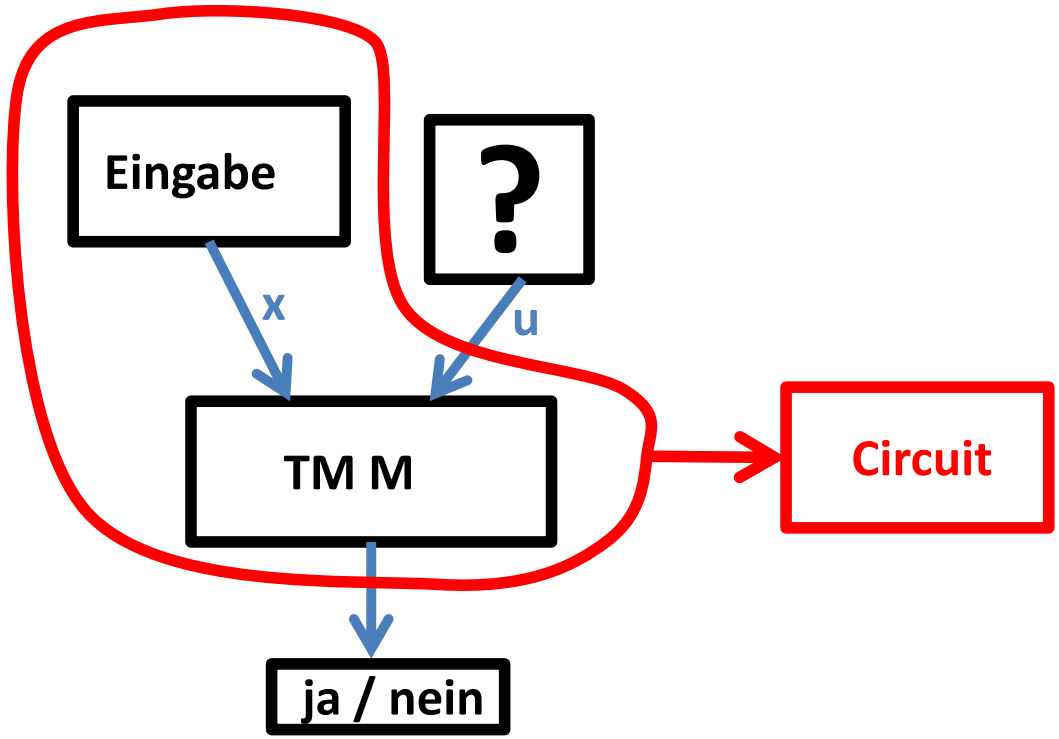
Die Sprache CKT-SAT

- Ein Wort c ist in CKT-SAT, wenn der durch c beschriebene Circuit C eine erfüllende Belegung besitzt.
- CKT-SAT$\in \mathcal{NP}$. (Warum?)
- CKT-SAT ist auch $\mathcal{NP}$-schwer, also $\mathcal{NP}$-vollständig.
- CKT-SAT $\propto_p$ 3SAT.

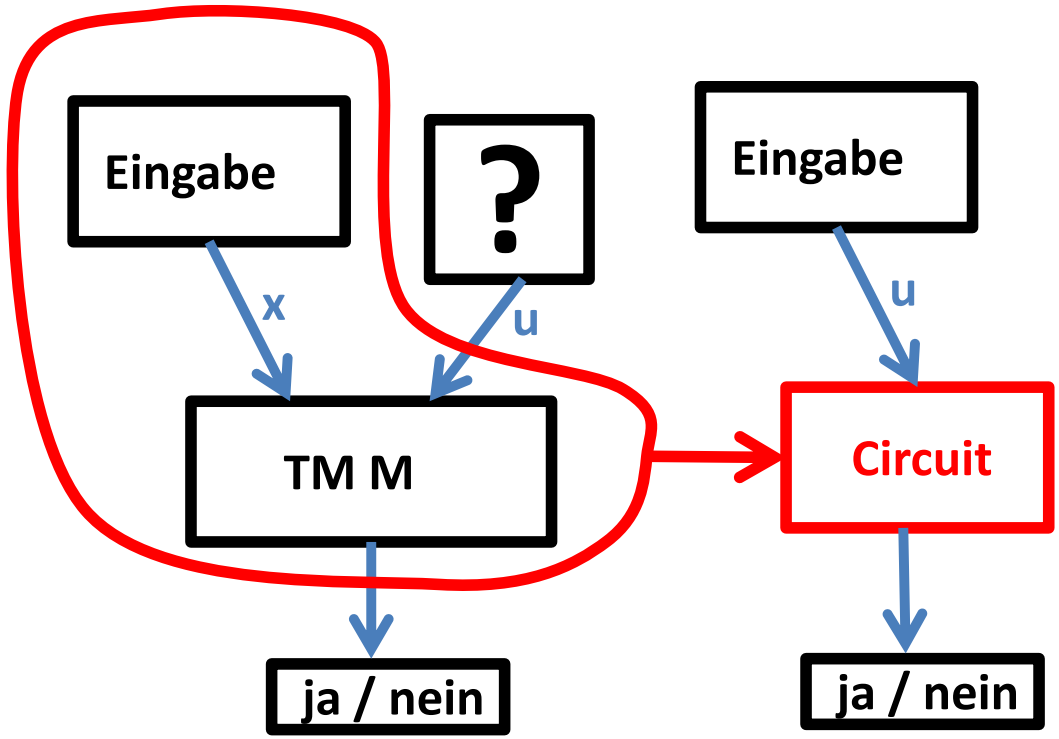
Die Sprache CKT-SAT



Die Sprache CKT-SAT



Die Sprache CKT-SAT



Weitere Vorgehensweise

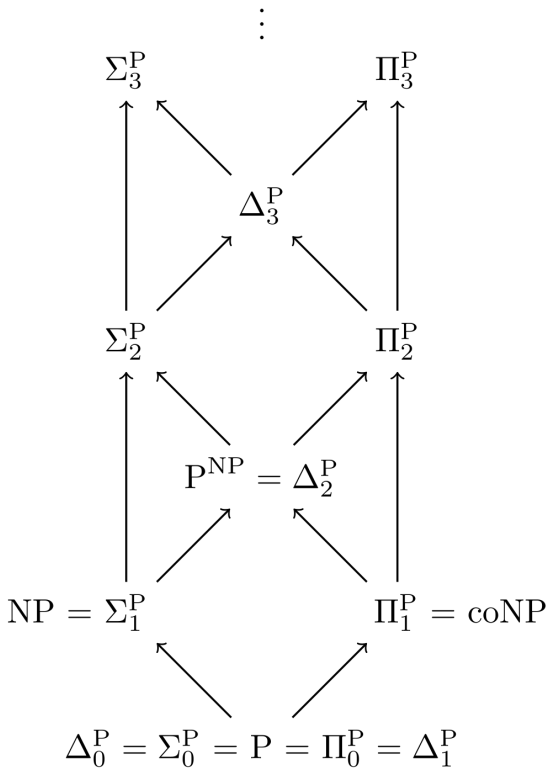
- Definition von Komplexitätsklassen
- Einführung sinnvoller Beschränkungen
- Wozu kann man Circuits denn nun verwenden?
  - Neue Ansätze zur Untersuchung von Komplexitätsklassen
  - Bessere Beschreibung paralleler Algorithmen

Weitere Vorgehensweise

- Definition von Komplexitätsklassen
- Einführung sinnvoller Beschränkungen
- Wozu kann man Circuits denn nun verwenden?
  - Neue Ansätze zur Untersuchung von Komplexitätsklassen
  - Bessere Beschreibung paralleler Algorithmen

$\mathcal{P}_{/poly}$ und $\mathcal{NP}$

Zur Erinnerung: Polynomielle Hierarchie



(Bild aus Wikipedia)

Satz von Meyer

- Falls $EXPTIME \subseteq \mathcal{P}_{/poly}$, dann gilt $EXPTIME = \Sigma_2^p$.

Satz von Meyer

- Falls $\mathit{EXPTIME} \subseteq \mathcal{P}_{/\mathit{poly}}$, dann gilt $\mathit{EXPTIME} = \Sigma_2^p$.
- Folgerung: Falls $\mathit{EXPTIME} \subseteq \mathcal{P}_{/\mathit{poly}}$, dann gilt $\mathcal{P} \neq \mathcal{NP}$. Beweis:

Satz von Meyer

- Falls $EXPTIME \subseteq \mathcal{P}_{/poly}$, dann gilt $EXPTIME = \Sigma_2^p$.
- Folgerung: Falls $EXPTIME \subseteq \mathcal{P}_{/poly}$, dann gilt $\mathcal{P} \neq \mathcal{NP}$. Beweis:
  1. aus $EXPTIME \subseteq \mathcal{P}_{/poly}$ folgt $EXPTIME = \Sigma_2^p$

Satz von Meyer

- Falls $\mathit{EXPTIME} \subseteq \mathcal{P}_{/\mathit{poly}}$, dann gilt $\mathit{EXPTIME} = \Sigma_2^p$.
- Folgerung: Falls $\mathit{EXPTIME} \subseteq \mathcal{P}_{/\mathit{poly}}$, dann gilt $\mathcal{P} \neq \mathcal{NP}$. Beweis:
  1. aus $\mathit{EXPTIME} \subseteq \mathcal{P}_{/\mathit{poly}}$ folgt $\mathit{EXPTIME} = \Sigma_2^p$
  2. wäre $\mathcal{P} = \mathcal{NP}$, so wäre $\mathcal{P} = \Sigma_2^p$

Satz von Meyer

- Falls $\mathit{EXPTIME} \subseteq \mathcal{P}_{/poly}$, dann gilt $\mathit{EXPTIME} = \Sigma_2^p$.
- Folgerung: Falls $\mathit{EXPTIME} \subseteq \mathcal{P}_{/poly}$, dann gilt $\mathcal{P} \neq \mathcal{NP}$. Beweis:
  1. aus $\mathit{EXPTIME} \subseteq \mathcal{P}_{/poly}$ folgt $\mathit{EXPTIME} = \Sigma_2^p$
  2. wäre $\mathcal{P} = \mathcal{NP}$, so wäre $\mathcal{P} = \Sigma_2^p$
  3. $\Rightarrow \mathcal{P} = \mathit{EXPTIME}$

Satz von Meyer

- Falls $\mathit{EXPTIME} \subseteq \mathcal{P}_{/\mathit{poly}}$, dann gilt $\mathit{EXPTIME} = \Sigma_2^p$.
- Folgerung: Falls $\mathit{EXPTIME} \subseteq \mathcal{P}_{/\mathit{poly}}$, dann gilt $\mathcal{P} \neq \mathcal{NP}$. Beweis:
  1. aus $\mathit{EXPTIME} \subseteq \mathcal{P}_{/\mathit{poly}}$ folgt $\mathit{EXPTIME} = \Sigma_2^p$
  2. wäre $\mathcal{P} = \mathcal{NP}$, so wäre $\mathcal{P} = \Sigma_2^p$
  3. $\Rightarrow \mathcal{P} = \mathit{EXPTIME}$
  4. dies widerspricht dem Zeithierarchie-Theorem

Satz von Karp-Lipton

- Falls $\mathcal{NP} \subseteq \mathcal{P}_{/poly}$, dann gilt $\mathcal{PH} = \Sigma_2^p$.

Satz von Karp-Lipton

- Falls $\mathcal{NP} \subseteq \mathcal{P}_{/poly}$, dann gilt $\mathcal{PH} = \Sigma_2^p$.
- d.h. die polynomielle Hierarchie fällt zusammen

Satz von Karp-Lipton

- Falls $\mathcal{NP} \subseteq \mathcal{P}_{/poly}$, dann gilt $\mathcal{PH} = \Sigma_2^p$.

- d.h. die polynomielle Hierarchie fällt zusammen
- es wird aber $\mathcal{PH} \supsetneq \Sigma_2^p$ vermutet

Satz von Karp-Lipton

- Falls $\mathcal{NP} \subseteq \mathcal{P}_{/poly}$, dann gilt $\mathcal{PH} = \Sigma_2^p$.

- d.h. die polynomielle Hierarchie fällt zusammen
- es wird aber $\mathcal{PH} \supsetneq \Sigma_2^p$ vermutet $\Rightarrow \mathcal{NP} \not\subseteq \mathcal{P}_{/poly}$

Satz von Karp-Lipton

- Falls $\mathcal{NP} \subseteq \mathcal{P}_{/poly}$, dann gilt $\mathcal{PH} = \Sigma_2^p$.
- d.h. die polynomielle Hierarchie fällt zusammen
- es wird aber $\mathcal{PH} \supsetneq \Sigma_2^p$ vermutet $\Rightarrow \mathcal{NP} \not\subseteq \mathcal{P}_{/poly}$
- $\Rightarrow \mathcal{P} \neq \mathcal{NP}$

schwere Funktionen

Karlsruher Institut für Technologie

Motivation: Bei TMs ist es schwierig, lowerbounds zu beweisen. Wie sieht es bei Boolean Circuits aus?

Karlsruher Institut für Technologie

Existenz schwerer Funktionen

Für jedes $n > 1$ existiert eine Funktion $f : \{0,1\}^n \to \{0,1\}$, die nicht durch einen Boolean circuit der Größe $2^n/(10n)$ berechenbar ist.
(Beweis: siehe Tafel)

Existenz schwerer Funktionen

Für jedes $n > 1$ existiert eine Funktion $f : \{0,1\}^n \to \{0,1\}$, die nicht durch einen Boolean circuit der Größe $2^n/(10n)$ berechenbar ist.
(Beweis: siehe Tafel)

Beschreibt auch nur ein solches f eine Sprache $L \in \mathcal{NP}$, so wäre $\mathcal{P} \neq \mathcal{NP}$!

Karlsruher Institut für Technologie

Existenz schwerer Funktionen

Für jedes $n > 1$ existiert eine Funktion $f : \{0,1\}^n \to \{0,1\}$, die nicht durch einen Boolean circuit der Größe $2^n/(10n)$ berechenbar ist.
(Beweis: siehe Tafel)

Beschreibt auch nur ein solches f eine Sprache $L \in \mathcal{NP}$, so wäre $\mathcal{P} \neq \mathcal{NP}$!

Bisher ist es aber nicht gelungen, das zu finden.

nonuniform hierachy theorem

In etwa: Haben wir einen größeren Circuit zur Verfügung, dann können wir damit auch mehr berechnen.

nonuniform hierachy theorem

In etwa: Haben wir einen größeren Circuit zur Verfügung, dann können wir damit auch mehr berechnen.

- Es seien $T, T' : \mathbb{N} \to \mathbb{N}$ Funktionen mit
- $2^n/n > T'(n) > 10\,T(n) > n$.

nonuniform hierachy theorem

In etwa: Haben wir einen größeren Circuit zur Verfügung, dann können wir damit auch mehr berechnen.

- Es seien $T, T' : \mathbb{N} \to \mathbb{N}$ Funktionen mit
- $2^n/n > T'(n) > 10\,T(n) > n$.
- Dann gilt $\mathbf{SIZE}(T(n)) \subsetneq \mathbf{SIZE}(T'(n))$

Weitere Vorgehensweise

- Definition von Komplexitätsklassen
- Einführung sinnvoller Beschränkungen
- Wozu kann man Circuits denn nun verwenden?
  - Neue Ansätze zur Untersuchung von Komplexitätsklassen
  - Bessere Beschreibung paralleler Algorithmen

Karlsruher Institut für Technologie

Parallele Algorithmen

Parallele Algorithmen

In diesem Kapitel beschäftigen wir uns mit den folgenden Fragen:

- Wie definiert man effiziente Parallelalgorithmen?

Parallele Algorithmen

In diesem Kapitel beschäftigen wir uns mit den folgenden Fragen:

- Wie definiert man effiziente Parallelalgorithmen?
- Welche Beziehung gibt es zur Klasse der effizienten sequenziellen Algorithmen?

Parallele Algorithmen

In diesem Kapitel beschäftigen wir uns mit den folgenden Fragen:

- Wie definiert man effiziente Parallelalgorithmen?
- Welche Beziehung gibt es zur Klasse der effizienten sequenziellen Algorithmen?
- Welche Beispiele gibt es für effizient lösbare Sprachen?

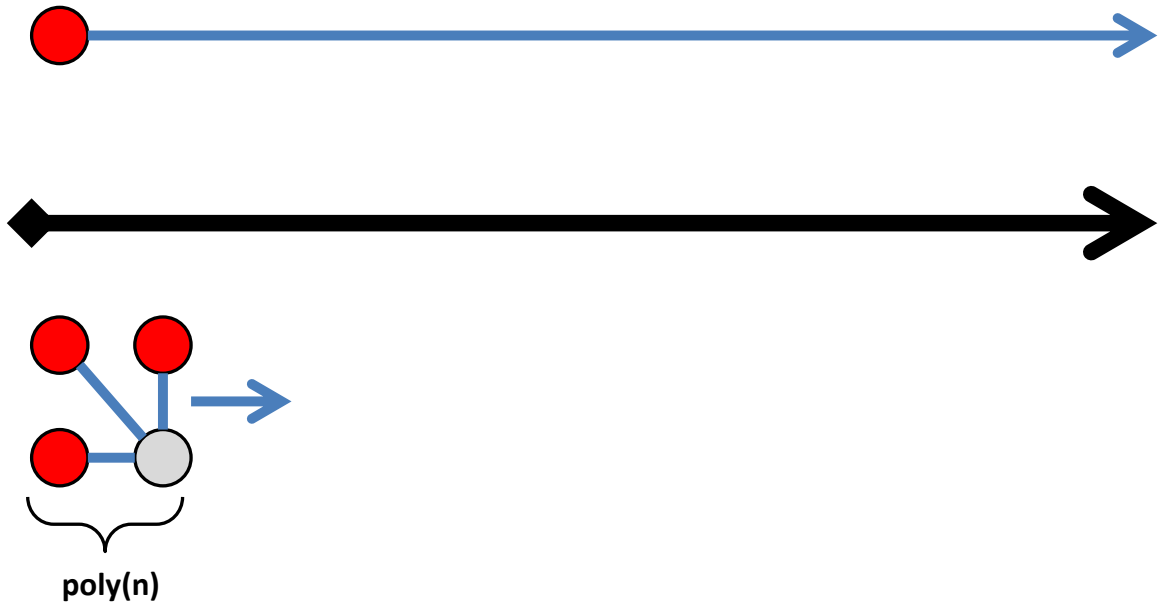
Parallele Algorithmen

In diesem Kapitel beschäftigen wir uns mit den folgenden Fragen:

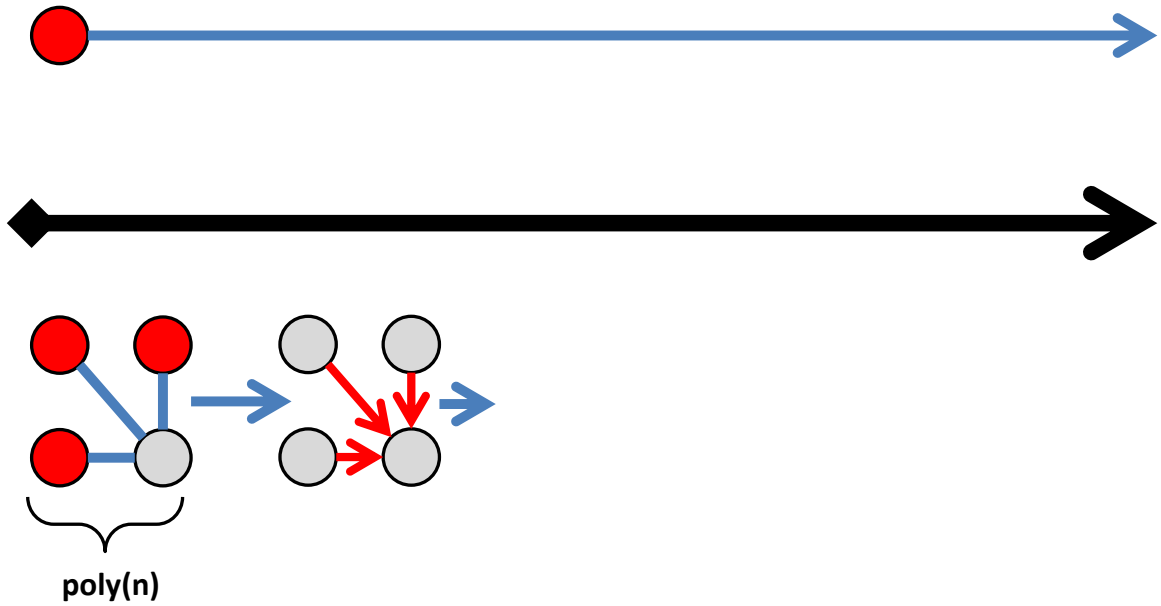
- Wie definiert man effiziente Parallelalgorithmen?
- Welche Beziehung gibt es zur Klasse der effizienten sequenziellen Algorithmen?
- Welche Beispiele gibt es für effizient lösbare Sprachen?
- Was hat das ganze mit Boolean circuits zu tun?

Effiziente Parallelalgorithmen

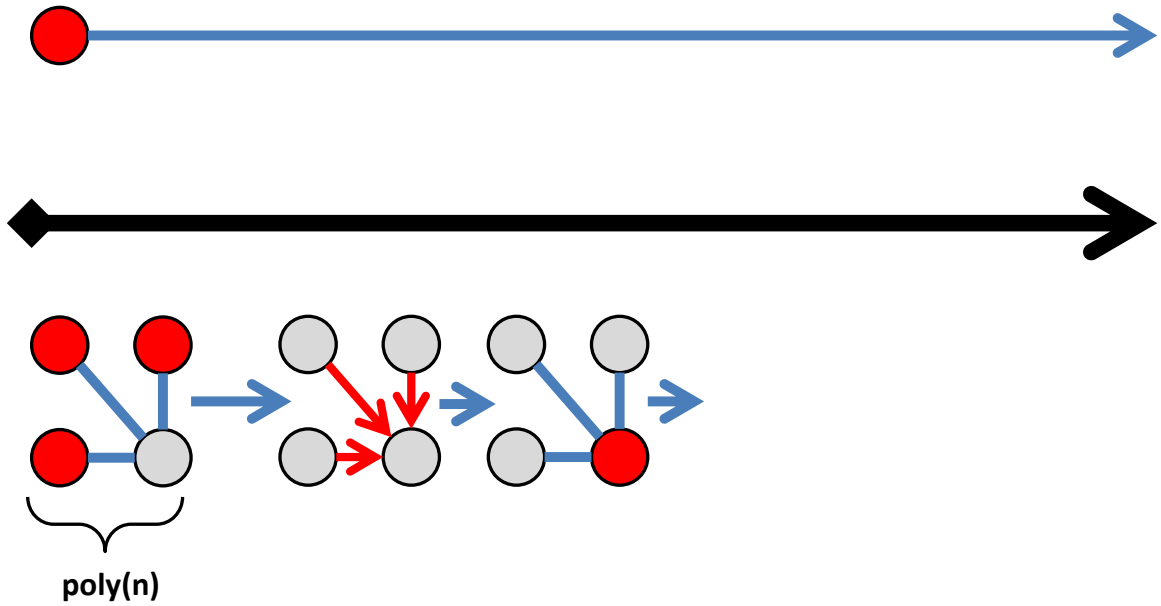
Effiziente Parallelalgorithmen



Effiziente Parallelalgorithmen



Effiziente Parallelalgorithmen



Effiziente Parallelalgorithmen

Karlsruher Institut für Technologie

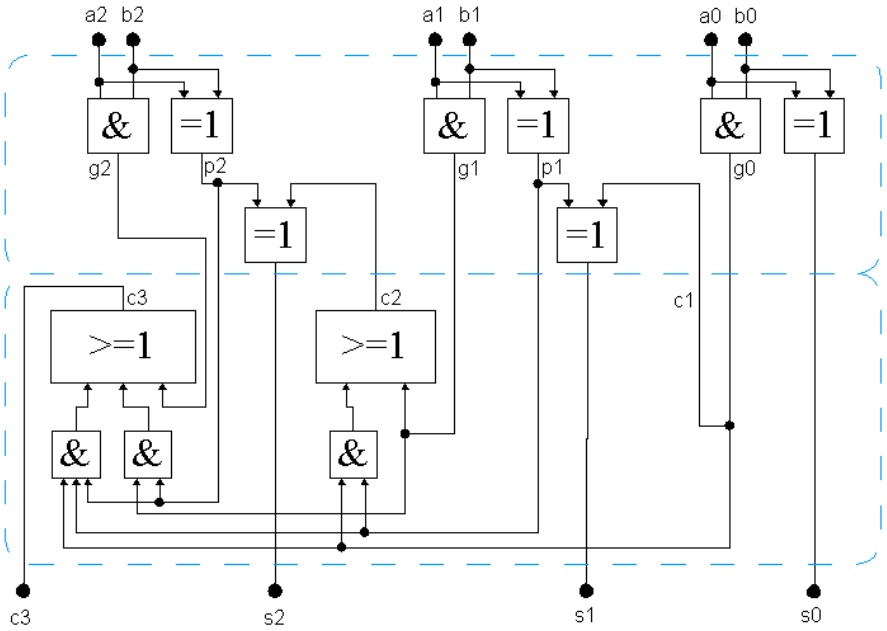
Definition: Ein Problem hat einen **effizienten Parallelalgorithmus** wenn eine Eingabe der Größe n mit $n^{O(1)}$ Prozessoren in einer Zeit von $\log^{O(1)}(n)$ gelöst werden kann.

Beispiele

- Addition zweier Zahlen mit carry lookahead
- Multiplikation und Division zweier Ganzzahlen
- Matrixoperationen: Produkt, Rang, Determinante und Inverse
- Graphen: kürzeste Pfade und minimale Spannbäume

Beispiel: carry-lookahead-Addition

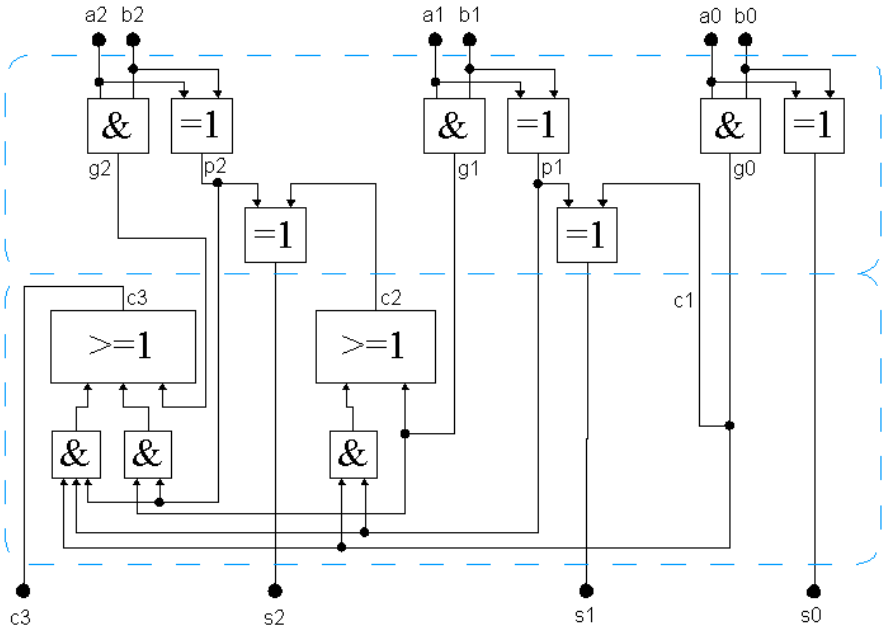
Bekannt und geliebt aus den TI-Vorlesungen



(Bild aus Wikipedia)

Beispiel: carry-lookahead-Addition

Clou hier: Vorberechnung des carry-flags



(Bild aus Wikipedia)

Die Klasse $\mathcal{NC}$

Für jedes d ist eine Sprache L genau dann in $\mathcal{NC}^d$, wenn

- sie von einer Circuit-Familie $\{C_n\}$ erkannt wird, wobei
- C_n eine Größe von $poly(n)$ und
- eine Tiefe von $O(\log^d n)$ hat und
- *logspace-uniform* ist

$$\mathcal{NC} := \bigcup_{i \geq 1} \mathcal{NC}^i$$

Die Klasse $\mathcal{NC}$

Für jedes d ist eine Sprache L genau dann in $\mathcal{NC}^d$, wenn

- sie von einer Circuit-Familie $\{C_n\}$ erkannt wird, wobei
- C_n eine Größe von $poly(n)$ und
- eine Tiefe von $O(\log^d n)$ hat und
- *logspace-uniform* ist

$$\mathcal{NC} := \bigcup_{i \geq 1} \mathcal{NC}^i$$

Vergleich zu $\mathcal{P}_{/poly}$: $\mathcal{NC}$ hat stärkere Anforderungen an Tiefe und Uniformität

Die Klasse $\mathcal{AC}$

- $\mathcal{AC}^i$ wird ähnlich zu $\mathcal{NC}^i$ definiert
- Unterschied: die UND- bzw. ODER-Gatter dürfen eine unbegrenzte Anzahl Eingänge haben
- $\mathcal{AC} := \bigcup_{i \geq 1} \mathcal{AC}^i$

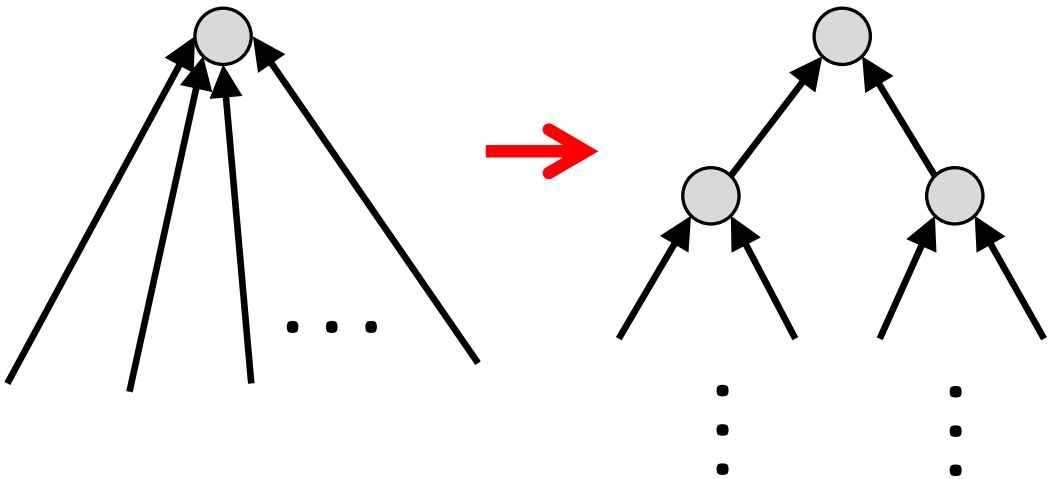
Die Klasse $\mathcal{AC}$

- $\mathcal{AC}^i$ wird ähnlich zu $\mathcal{NC}^i$ definiert
- Unterschied: die UND- bzw. ODER-Gatter dürfen eine unbegrenzte Anzahl Eingänge haben
- $\mathcal{AC} := \bigcup_{i \geq 1} \mathcal{AC}^i$

Bemerkung: es gilt

- $\mathcal{NC}^i \subseteq \mathcal{AC}^i \subseteq \mathcal{NC}^{i+1}$
- $\mathcal{NC} = \mathcal{AC}$

Zur Bemerkung $\mathcal{AC}^i \subseteq \mathcal{NC}^{i+1}$



Beispiel

$PARITY := \{x |$ x hat ungerade Anzahl von 1en$\}$

$PARITY \in \mathcal{NC}^1$

Beispiel

$PARITY := \{x |$ x hat ungerade Anzahl von 1en$\}$

$PARITY \in \mathcal{NC}^1$

Begründung: **Binärbaum**

- Wurzel des Baumes: Ausgangsknoten
- Blätter: Eingangsknoten
- linker / rechter Vorgänger: Parität des linken / rechten Teilbaumes
- logarithmische Tiefe des Baumes (klar)

(Wurzel und Blätter: Umkehren der Richtung)

Beispiel

$PARITY := \{x |$ x hat ungerade Anzahl von 1en$\}$

$PARITY \in \mathcal{NC}^1$

Begründung: **Binärbaum**

- Wurzel des Baumes: Ausgangsknoten
- Blätter: Eingangsknoten
- linker / rechter Vorgänger: Parität des linken / rechten Teilbaumes
- logarithmische Tiefe des Baumes (klar)

(Wurzel und Blätter: Umkehren der Richtung)

Bemerkung: $PARITY \notin \mathcal{AC}^0$

$\mathcal{P}$-Vollständigkeit

Analog zur $\mathcal{NP}$-Vollständigkeit, aber logspace-Transformation statt polynomieller Transformation

$\mathcal{P}$-Vollständigkeit

Analog zur $\mathcal{NP}$-Vollständigkeit, aber logspace-Transformation statt polynomieller Transformation

Beobachtungen: Sei L $\mathcal{P}$-Vollständig

- $L \in \mathcal{NC}$ gdw. $\mathcal{P} = \mathcal{NC}$
- $L \in \mathit{LOGSPACE}$ gdw. $\mathcal{P} = \mathit{LOGSPACE}$

$\mathcal{P}$-Vollständigkeit

Analog zur $\mathcal{NP}$-Vollständigkeit, aber logspace-Transformation statt polynomieller Transformation

Beobachtungen: Sei L $\mathcal{P}$-Vollständig

- $L \in \mathcal{NC}$ gdw. $\mathcal{P} = \mathcal{NC}$
- $L \in \mathit{LOGSPACE}$ gdw. $\mathcal{P} = \mathit{LOGSPACE}$

Beispiel: *CIRCUIT-EVAL*

$\mathcal{PH}$ und *EXPTIME*

Definition: Eine Circuit-Familie $\{C_n\}_{n\geq 1}$ heißt DC-uniform wenn die Funktionen SIZE, TYPE und EDGE in polynomieller Zeit berechenbar sind.

$L \in \mathcal{PH}$ genau dann wenn L durch eine DC-uniforme Circuit-Familie erkannt wird, wobei

1. die Größe ist $2^{n^{O(1)}}$ und die Tiefe konstant
2. die UND- bzw. ODER-Gatter unbegrenzte Anzahl Eingänge haben können
3. alle NOT-Gatter am Eingang des Circuits sind

Wird die Forderung nach konstanter Tiefe weggelassen, erhalten wir *EXPTIME*.

Fazit

Karlsruher Institut für Technologie

Boolean Circuits

- sind interessant (hoffentlich)
- haben tatsächlich sinnvolle Anwendungsgebiete
- bieten eine neue Perspektive auf das $\mathcal{P} \neq \mathcal{NP}$-Problem
- sind aber hinsichtlich der Lösung des Problems nicht weniger kompliziert

Karlsruher Institut für Technologie

Danke für eure Aufmerksamkeit!