

# Parallelität

- Welche Probleme in  $\mathbb{P}$  sind parallelisierbar?

Gibt es vom Standpunkt der Parallelisierbarkeit  
*schwierigste Probleme* in  $\mathbb{P}$ ?

Gibt es dazu einen Reduktionsbegriff?

- Was soll „parallelisierbar“ überhaupt bedeuten?

Wenn es eine *rasant schnelle*(!?) Berechnung  
bei *vernünftiger Größe*(!?) gibt.

Zur Klärung der Begrifflichkeiten verwenden wir Schaltkreise.

# Schaltkreise

Ein Schaltkreis  $S$  wird durch den Vektor

$$S = (G, Q, R, \text{gatter}, n, \text{eingabe})$$

beschrieben:

- $G = (V, E)$  ist ein gerichteter, azyklischer Graph.
- $Q$  ist die Menge der Quellen (Knoten mit nur ausgehenden Kanten) und  $R$  die Menge der Senken (Knoten mit nur eingehenden Kanten) von  $G$ .
- Die Funktion  $\text{eingabe}: Q \rightarrow \{1, \dots, n\}$  weist jeder Quelle die Position des entsprechenden Eingabebits zu. Die Funktion  $\text{gatter}: V \setminus Q \rightarrow \{\neg, \vee, \wedge\}$  weist jedem inneren Knoten von  $G$  eine Gatter-Funktion zu.

Ein Schaltkreis  $S$  wird durch den Vektor

$$S = (G, Q, R, \text{gatter}, n, \text{eingabe})$$

beschrieben:

- $G = (V, E)$  ist ein gerichteter, azyklischer Graph.
- $Q$  ist die Menge der Quellen (Knoten mit nur ausgehenden Kanten) und  $R$  die Menge der Senken (Knoten mit nur eingehenden Kanten) von  $G$ .
- Die Funktion  $\text{eingabe}: Q \rightarrow \{1, \dots, n\}$  weist jeder Quelle die Position des entsprechenden Eingabebits zu. Die Funktion  $\text{gatter}: V \setminus Q \rightarrow \{\neg, \vee, \wedge\}$  weist jedem inneren Knoten von  $G$  eine Gatter-Funktion zu.

$S$  berechnet die Funktion  $f_S: \{0, 1\}^n \rightarrow \{0, 1\}^{|R|}$ , indem

- die  $n$  Eingabebits an die **Quellen** in  $G$  angelegt werden,
- jeder Knoten das Ergebnis seiner Gatterfunktion weiterleitet
- und das Resultat an den Senken abgelesen wird.

Sei  $S = (G, Q, R, \text{gatter}, n, \text{eingabe})$  ein Schaltkreis.

- 1 **Tiefe**( $S$ ), die *Tiefe* von  $S$ , ist die Länge des längsten Weges in  $G$ .
- 2 **Größe**( $S$ ), die *Größe* von  $S$ , ist die Anzahl der Knoten von  $G$ , wobei Quellen nicht mitgezählt werden.
- 3 **Fanin**( $S$ ), der *Fanin* von  $S$  ist das Maximum, über alle Knoten  $v$ , der Anzahl eingehender Kanten von  $v$ .

Wie viele Gatter und welche Tiefe benötigen die *meisten* booleschen Funktionen?

- Bestimme die Anzahl der Funktionen  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  :

Wie viele Gatter und welche Tiefe benötigen die *meisten* booleschen Funktionen?

- Bestimme die Anzahl der Funktionen  $f : \{0, 1\}^n \rightarrow \{0, 1\} : 2^{2^n}$ .
- Wieviele Schaltkreise gibt es mit Fanin 2
  - ▶ und  $m$  Gattern: Höchstens



Wie viele Gatter und welche Tiefe benötigen die *meisten* booleschen Funktionen?

- Bestimme die Anzahl der Funktionen  $f : \{0, 1\}^n \rightarrow \{0, 1\} : 2^{2^n}$ .
- Wieviele Schaltkreise gibt es mit Fanin 2
  - ▶ und  $m$  Gattern: Höchstens  $m^{O(m)}$ ,
  - ▶ bzw. Tiefe  $d$ :

Wie viele Gatter und welche Tiefe benötigen die *meisten* booleschen Funktionen?

- Bestimme die Anzahl der Funktionen  $f : \{0, 1\}^n \rightarrow \{0, 1\} : 2^{2^n}$ .
- Wieviele Schaltkreise gibt es mit Fanin 2
  - ▶ und  $m$  Gattern: Höchstens  $m^{O(m)}$ ,
  - ▶ bzw. Tiefe  $d$ : Höchstens  $2^{2^{O(d)}}$ .
- Für die **meisten** Funktionen  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  werden Schaltkreise der  
Größe

Wie viele Gatter und welche Tiefe benötigen die *meisten* booleschen Funktionen?

- Bestimme die Anzahl der Funktionen  $f : \{0, 1\}^n \rightarrow \{0, 1\} : 2^{2^n}$ .
- Wieviele Schaltkreise gibt es mit Fanin 2
  - ▶ und  $m$  Gattern: Höchstens  $m^{O(m)}$ ,
  - ▶ bzw. Tiefe  $d$ : Höchstens  $2^{2^{O(d)}}$ .
- Für die **meisten** Funktionen  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  werden Schaltkreise der Größe  $\Omega(2^n/n)$  und Tiefe  $\Omega(n)$

benötigt. (Man kann sogar Tiefe mindestens  $n - O(\log n)$  zeigen.)

Für jede Funktion  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  gilt

$$\text{DEPTH}(f) \leq n + \lceil \log_2 n \rceil \text{ sowie } \text{SIZE}(f) \stackrel{\text{Lupanov}}{\leq} (1 + o(1)) \cdot \frac{2^n}{n}$$

(a) Eine **Schaltkreisfamilie** ist eine Folge

$$\mathcal{S} = (S_n)_{n \in \mathbb{N}}$$

von Schaltkreisen, so dass  $S_n$  eine Boolesche Funktion auf genau  $n$  Eingaben berechnet.  $\mathcal{S}$  berechnet die Funktion

$$f : \{0, 1\}^* \rightarrow \{0, 1\},$$

wenn  $S_n$  die Funktion  $f$ , eingeschränkt auf  $\{0, 1\}^n$ , berechnet.

- (a) Eine **Schaltkreisfamilie** ist eine Folge

$$\mathcal{S} = (S_n)_{n \in \mathbb{N}}$$

von Schaltkreisen, so dass  $S_n$  eine Boolesche Funktion auf genau  $n$  Eingaben berechnet.  $\mathcal{S}$  berechnet die Funktion

$$f : \{0, 1\}^* \rightarrow \{0, 1\},$$

wenn  $S_n$  die Funktion  $f$ , eingeschränkt auf  $\{0, 1\}^n$ , berechnet.

- (b) Eine Schaltkreisfamilie  $(S_n)_{n \in \mathbb{N}}$  ist **uniform**, wenn es eine  $\log_2(\text{Größe}(S_n) + n)$ -platzbeschränkte, deterministische Turingmaschine gibt, die für Eingabe  $1^n$

- alle Knoten von  $S_n$  aufzählt,
- jedem inneren Knoten eine Gatter-Funktion aus  $\{\neg, \vee, \wedge\}$ , zuweist,
- sämtliche Kanten von  $S_n$  aufzählt und
- jeder Quelle von  $S_n$  eine Bitposition zuweist.

## *Nicht-uniforme* Schaltkreisfamilien

$$\mathcal{S} = (\mathcal{S}_n \mid n \in \mathbb{N})$$

können „unanständig mächtig“ sein.

- Das spezielle Halteproblem

$$H_\varepsilon = \{1^{(M)} \mid M \text{ hält auf dem leeren Wort} \}$$

ist unentscheidbar.

## *Nicht-uniforme* Schaltkreisfamilien

$$\mathcal{S} = (\mathcal{S}_n \mid n \in \mathbb{N})$$

können „unanständig mächtig“ sein.

- Das spezielle Halteproblem

$$H_\varepsilon = \{1^{\langle M \rangle} \mid M \text{ hält auf dem leeren Wort}\}$$

ist unentscheidbar.

- $\mathcal{S}_n$  akzeptiert  $w \in \{0, 1\}^n$  genau dann, wenn  $n = \langle M \rangle$  für eine Turingmaschine  $M$ , die auf dem leeren Wort hält.

## *Nicht-uniforme* Schaltkreisfamilien

$$\mathcal{S} = (\mathcal{S}_n \mid n \in \mathbb{N})$$

können „unanständig mächtig“ sein.

- Das spezielle Halteproblem

$$H_\varepsilon = \{1^{\langle M \rangle} \mid M \text{ hält auf dem leeren Wort} \}$$

ist unentscheidbar.

- $\mathcal{S}_n$  akzeptiert  $w \in \{0, 1\}^n$  genau dann, wenn  $n = \langle M \rangle$  für eine Turingmaschine  $M$ , die auf dem leeren Wort hält.

Eine uniforme Schaltkreisfamilie muss *einfach konstruierbar* sein.



Für Funktionen  $d, s : \mathbb{N} \rightarrow \mathbb{N}$  definiere

$$\text{DEPTH}_{\text{uniform}}(d)$$

als die Klasse aller Sprachen  $L$ , die durch eine uniforme Schaltkreisfamilie in Tiefe  $O(d(n))$  und mit Fanin zwei berechnet werden.

Für Funktionen  $d, s : \mathbb{N} \rightarrow \mathbb{N}$  definiere

$$\text{DEPTH}_{\text{uniform}}(d)$$

als die Klasse aller Sprachen  $L$ , die durch eine uniforme Schaltkreisfamilie in Tiefe  $O(d(n))$  und mit Fanin zwei berechnet werden.

$$\text{SIZE}_{\text{uniform}}(s)$$

ist die Klasse aller Sprachen  $L$ , die durch eine uniforme Schaltkreisfamilie in Größe  $O(s(n))$  und mit Fanin zwei berechnet werden.

Für Funktionen  $d, s : \mathbb{N} \rightarrow \mathbb{N}$  definiere

$$\text{DEPTH}_{\text{uniform}}(d)$$

als die Klasse aller Sprachen  $L$ , die durch eine uniforme Schaltkreisfamilie in Tiefe  $O(d(n))$  und mit Fanin zwei berechnet werden.

$$\text{SIZE}_{\text{uniform}}(s)$$

ist die Klasse aller Sprachen  $L$ , die durch eine uniforme Schaltkreisfamilie in Größe  $O(s(n))$  und mit Fanin zwei berechnet werden. Schließlich besteht

$$\text{DEPTH-SIZE}_{\text{uniform}}(d, s) = \text{DEPTH}_{\text{uniform}}(d) \cap \text{SIZE}_{\text{uniform}}(s)$$

aus allen Sprachen, die sowohl in Tiefe  $O(d)$  wie auch in Größe  $O(s)$  durch eine uniforme Schaltkreisfamilie vom Fanin zwei berechnet werden.

*Problem:* Welche Tiefe und welche Größe sind asymptotisch notwendig und hinreichend für

- die Paritätsfunktion  $x_1 \oplus x_2 \oplus \dots \oplus x_n$
- und allgemein für reguläre Sprachen?

NC, Nick's class

Sei  $k \in \mathbb{N}$ .

①  $NC^k := \bigcup_{\ell=0}^{\infty} \text{DEPTH-SIZE}_{\text{uniform}}(\log_2^k n, n^\ell)$ .

②  $NC := \bigcup_{k \in \mathbb{N}} NC^k$

③  $AC^k$  ist wie  $NC^k$  definiert, allerdings ist der Fanin unbeschränkt.

④  $AC := \bigcup_{k \in \mathbb{N}} AC^k$

Sei  $k \in \mathbb{N}$ .

①  $NC^k := \bigcup_{\ell=0}^{\infty} \text{DEPTH-SIZE}_{\text{uniform}}(\log_2^k n, n^\ell)$ .

②  $NC := \bigcup_{k \in \mathbb{N}} NC^k$

③  $AC^k$  ist wie  $NC^k$  definiert, allerdings ist der Fanin unbeschränkt.

④  $AC := \bigcup_{k \in \mathbb{N}} AC^k$

**Rasant schnell** = **poly-logarithmische Zeit**,  
**Vernünftige Größe** = **polynomielle Größe**.

(a)  $AC^k \subseteq NC^{k+1} \subseteq AC^{k+1}$ .

(b)  $AC = NC \subseteq P$ .



(a)  $AC^k \subseteq NC^{k+1} \subseteq AC^{k+1}$ .

(b)  $AC = NC \subseteq P$ .

(a)  $NC^{k+1} \subseteq AC^{k+1}$  ist offensichtlich.

(a)  $AC^k \subseteq NC^{k+1} \subseteq AC^{k+1}$ .

(b)  $AC = NC \subseteq P$ .

(a)  $NC^{k+1} \subseteq AC^{k+1}$  ist offensichtlich.

$AC^k \subseteq NC^{k+1}$ : Die uniforme Schaltkreisfamilie  $(S_n)_{n \in \mathbb{N}}$  sei gegeben.

$$(a) \text{ AC}^k \subseteq \text{NC}^{k+1} \subseteq \text{AC}^{k+1}.$$

$$(b) \text{ AC} = \text{NC} \subseteq \text{P}.$$

(a)  $\text{NC}^{k+1} \subseteq \text{AC}^{k+1}$  ist offensichtlich.

$\text{AC}^k \subseteq \text{NC}^{k+1}$ : Die uniforme Schaltkreisfamilie  $(S_n)_{n \in \mathbb{N}}$  sei gegeben.

- ▶ Ein *UND*- bzw. *ODER*-Gatter mit  $p$  Eingängen kann durch einen binären Baum der Tiefe  $\lceil \log_2 p \rceil$  und Größe höchstens  $2p + 1$  simuliert werden.
- ▶ Der Fanin von  $S_n$  ist durch  $n + \text{Größe}(S_n)$  beschränkt.
- ▶ Die Ersetzung der Knoten von  $S_n$  durch Binärbäume erhöht die Tiefe um höchstens den Faktor  $O(\log_2 n)$ , die Größe wird höchstens quadriert.

(a)  $AC^k \subseteq NC^{k+1} \subseteq AC^{k+1}$ .

(b)  $AC = NC \subseteq P$ .

(a)  $NC^{k+1} \subseteq AC^{k+1}$  ist offensichtlich.

$AC^k \subseteq NC^{k+1}$ : Die uniforme Schaltkreisfamilie  $(S_n)_{n \in \mathbb{N}}$  sei gegeben.

- ▶ Ein *UND*- bzw. *ODER*-Gatter mit  $p$  Eingängen kann durch einen binären Baum der Tiefe  $\lceil \log_2 p \rceil$  und Größe höchstens  $2p + 1$  simuliert werden.
- ▶ Der Fanin von  $S_n$  ist durch  $n + \text{Größe}(S_n)$  beschränkt.
- ▶ Die Ersetzung der Knoten von  $S_n$  durch Binärbäume erhöht die Tiefe um höchstens den Faktor  $O(\log_2 n)$ , die Größe wird höchstens quadriert.

(b)  $AC = NC$  folgt aus Teil (a).

$NC \subseteq P$ : Uniforme Schaltkreise polynomieller Größe können in polynomieller Zeit ausgewertet werden.

# Was geht wie schnell? (Ohne Beweis)

- In  $AC^0$  liegen
  - ▶ die Addition von zwei  $n$ -Bit Zahlen,
  - ▶ die Multiplikation Boolescher Matrizen.

# Was geht wie schnell? (Ohne Beweis)

- In  $AC^0$  liegen
  - ▶ die Addition von zwei  $n$ -Bit Zahlen,
  - ▶ die Multiplikation Boolescher Matrizen.
- In  $NC^1$  liegen
  - ▶ die Berechnung  $x_1 \oplus x_2 \oplus \dots \oplus x_n$  der XOR-Funktion und, allgemeiner, die Klasse der regulären Sprachen,
  - ▶ die Multiplikation von zwei  $n$ -Bit Zahlen.

# Was geht wie schnell? (Ohne Beweis)

- In  $AC^0$  liegen
  - ▶ die Addition von zwei  $n$ -Bit Zahlen,
  - ▶ die Multiplikation Boolescher Matrizen.
- In  $NC^1$  liegen
  - ▶ die Berechnung  $x_1 \oplus x_2 \oplus \dots \oplus x_n$  der XOR-Funktion und, allgemeiner, die Klasse der regulären Sprachen,
  - ▶ die Multiplikation von zwei  $n$ -Bit Zahlen.
- In  $AC^1$  liegen
  - ▶ D-REACHABILITY,
  - ▶ das Wortproblem  $\{ w : w \in L(G) \}$  für jede kontextfreie Grammatik  $G$ .

# Was geht wie schnell? (Ohne Beweis)

- In  $AC^0$  liegen
  - ▶ die Addition von zwei  $n$ -Bit Zahlen,
  - ▶ die Multiplikation Boolescher Matrizen.
- In  $NC^1$  liegen
  - ▶ die Berechnung  $x_1 \oplus x_2 \oplus \dots \oplus x_n$  der XOR-Funktion und, allgemeiner, die Klasse der regulären Sprachen,
  - ▶ die Multiplikation von zwei  $n$ -Bit Zahlen.
- In  $AC^1$  liegen
  - ▶ D-REACHABILITY,
  - ▶ das Wortproblem  $\{w : w \in L(G)\}$  für jede kontextfreie Grammatik  $G$ .
- In Random-NC liegt das Maximum Matching Problem  
Bestimme eine größtmögliche Menge knoten-disjunkter Kanten.



# Was geht wie schnell? (Ohne Beweis)

- In  $AC^0$  liegen
  - ▶ die Addition von zwei  $n$ -Bit Zahlen,
  - ▶ die Multiplikation Boolescher Matrizen.
- In  $NC^1$  liegen
  - ▶ die Berechnung  $x_1 \oplus x_2 \oplus \dots \oplus x_n$  der XOR-Funktion und, allgemeiner, die Klasse der regulären Sprachen,
  - ▶ die Multiplikation von zwei  $n$ -Bit Zahlen.
- In  $AC^1$  liegen
  - ▶ D-REACHABILITY,
  - ▶ das Wortproblem  $\{w : w \in L(G)\}$  für jede kontextfreie Grammatik  $G$ .
- In Random-NC liegt das Maximum Matching Problem  
Bestimme eine größtmögliche Menge knoten-disjunkter Kanten.
- In  $P$ , aber wahrscheinlich nicht in  $NC$ , liegen
  - ▶ das Auswertungsproblem für einen Schaltkreis  $S$ :  
Bestimme die Ausgabe von  $S$  für Eingabe  $x$ .
  - ▶ das Problem der linearen Programmierung.
  - ▶ das Wortproblem für kontextfreie Sprachen  $\{(G, w) : w \in L(G), G \text{ k.f.}\}$ .

# Parallele Rechenzeit und Speicherplatz

Für die uniforme Schaltkreisfamilie  $\mathcal{S} = (S_n)_{n \in \mathbb{N}}$  gelte  $\text{Tiefe}(S_n) = \Omega(\log_2 n)$ .

*Eine platz-effiziente Simulation von  $S_n$  in Platz  $O(\text{Tiefe}(S_n))$ :*

Für die uniforme Schaltkreisfamilie  $\mathcal{S} = (S_n)_{n \in \mathbb{N}}$  gelte  $\text{Tiefe}(S_n) = \Omega(\log_2 n)$ .

*Eine platz-effiziente Simulation von  $S_n$  in Platz  $O(\text{Tiefe}(S_n))$ :*

- Werte  $S_n$  durch eine in der Senke von  $S$  beginnende **Tiefensuche** aus.
- Benutze die Uniformität von  $\mathcal{S}$ , um die aktuell benötigte Information über  $S_n$  neu zu berechnen.

Für die uniforme Schaltkreisfamilie  $\mathcal{S} = (S_n)_{n \in \mathbb{N}}$  gelte  $\text{Tiefe}(S_n) = \Omega(\log_2 n)$ .

*Eine platz-effiziente Simulation von  $S_n$  in Platz  $O(\text{Tiefe}(S_n))$ :*

- Werte  $S_n$  durch eine in der Senke von  $S$  beginnende **Tiefensuche** aus.
- Benutze die Uniformität von  $\mathcal{S}$ , um die aktuell benötigte Information über  $S_n$  neu zu berechnen.
  - ▶ Der Weg der Tiefensuche wird durch die Bitfolge  $\vec{b}$  repräsentiert.
    - ★ Falls  $b_i = 1$  (bzw.  $b_i = 0$ ), ist der  $(i + 1)$ -te Knoten des Wegs der rechte (bzw. linke) Nachfolger des  $i$ -ten Knoten.
    - ★ Die Länge von  $\vec{b}$  ist höchstens proportional zu  $\text{Tiefe}(S_n)$ .

Für die uniforme Schaltkreisfamilie  $\mathcal{S} = (S_n)_{n \in \mathbb{N}}$  gelte  $\text{Tiefe}(S_n) = \Omega(\log_2 n)$ .

*Eine platz-effiziente Simulation von  $S_n$  in Platz  $O(\text{Tiefe}(S_n))$ :*

- Werte  $S_n$  durch eine in der Senke von  $S$  beginnende **Tiefensuche** aus.
- Benutze die Uniformität von  $\mathcal{S}$ , um die aktuell benötigte Information über  $S_n$  neu zu berechnen.
  - ▶ Der Weg der Tiefensuche wird durch die Bitfolge  $\vec{b}$  repräsentiert.
    - ★ Falls  $b_i = 1$  (bzw.  $b_i = 0$ ), ist der  $(i + 1)$ -te Knoten des Wegs der rechte (bzw. linke) Nachfolger des  $i$ -ten Knoten.
    - ★ Die Länge von  $\vec{b}$  ist höchstens proportional zu  $\text{Tiefe}(S_n)$ .
  - ▶ Der Speicherplatz ist durch  $s = O(\text{Tiefe}(S_n))$  beschränkt.

Als Konsequenz:

$$\text{DEPTH}_{\text{uniform}}(\mathbf{s}) \subseteq \text{DSPACE}(\mathbf{s})$$

und  $\text{NC}^1 \subseteq$

Für die uniforme Schaltkreisfamilie  $\mathcal{S} = (S_n)_{n \in \mathbb{N}}$  gelte  $\text{Tiefe}(S_n) = \Omega(\log_2 n)$ .

*Eine platz-effiziente Simulation von  $S_n$  in Platz  $O(\text{Tiefe}(S_n))$ :*

- Werte  $S_n$  durch eine in der Senke von  $S$  beginnende **Tiefensuche** aus.
- Benutze die Uniformität von  $\mathcal{S}$ , um die aktuell benötigte Information über  $S_n$  neu zu berechnen.
  - ▶ Der Weg der Tiefensuche wird durch die Bitfolge  $\vec{b}$  repräsentiert.
    - ★ Falls  $b_i = 1$  (bzw.  $b_i = 0$ ), ist der  $(i + 1)$ -te Knoten des Wegs der rechte (bzw. linke) Nachfolger des  $i$ -ten Knoten.
    - ★ Die Länge von  $\vec{b}$  ist höchstens proportional zu  $\text{Tiefe}(S_n)$ .
  - ▶ Der Speicherplatz ist durch  $s = O(\text{Tiefe}(S_n))$  beschränkt.

Als Konsequenz:

$$\text{DEPTH}_{\text{uniform}}(\mathbf{s}) \subseteq \text{DSPACE}(\mathbf{s})$$

und  $\text{NC}^1 \subseteq \text{DL}$  folgt.

Wir betrachten uniforme Schaltkreise mit **unbeschränktem** Fanin.

(a) Berechne das **Boolesche Matrizenprodukt**

$$(A \cdot B)[i, j] = \bigvee_{k=1}^n A[i, k] \wedge B[k, j]$$

in Tiefe



Wir betrachten uniforme Schaltkreise mit **unbeschränktem** Fanin.

(a) Berechne das **Boolesche Matrizenprodukt**

$$(A \cdot B)[i, j] = \bigvee_{k=1}^n A[i, k] \wedge B[k, j]$$

in Tiefe zwei mit  $O(n^3)$  Gattern.

Wir betrachten uniforme Schaltkreise mit **unbeschränktem** Fanin.

(a) Berechne das **Boolesche Matrizenprodukt**

$$(A \cdot B)[i, j] = \bigvee_{k=1}^n A[i, k] \wedge B[k, j]$$

in Tiefe zwei mit  $O(n^3)$  Gattern.

(b) Berechne die **transitive Hülle** eines gerichteten Graphen in Tiefe  $O(\log_2 n)$  und Größe  $O(n^3 \log_2 n)$ .

Wir betrachten uniforme Schaltkreise mit **unbeschränktem** Fanin.

- (a) Berechne das **Boolesche Matrizenprodukt**

$$(A \cdot B)[i, j] = \bigvee_{k=1}^n A[i, k] \wedge B[k, j]$$

in Tiefe zwei mit  $O(n^3)$  Gattern.

- (b) Berechne die **transitive Hülle** eines gerichteten Graphen in Tiefe  $O(\log_2 n)$  und Größe  $O(n^3 \log_2 n)$ .

- (a) Das Matrizenprodukt: Der Brute-Force Ansatz funktioniert. ✓

Wir betrachten uniforme Schaltkreise mit **unbeschränktem** Fanin.

- (a) Berechne das **Boolesche Matrizenprodukt**

$$(A \cdot B)[i, j] = \bigvee_{k=1}^n A[i, k] \wedge B[k, j]$$

in Tiefe zwei mit  $O(n^3)$  Gattern.

- (b) Berechne die **transitive Hülle** eines gerichteten Graphen in Tiefe  $O(\log_2 n)$  und Größe  $O(n^3 \log_2 n)$ .

- (a) Das Matrizenprodukt: Der Brute-Force Ansatz funktioniert. ✓
- (b) Für die Adjazenzmatrix  $A_G$  zeige durch Induktion über  $d$ :
- ▶  $A_G^d[i, j] = 1 \iff$

Wir betrachten uniforme Schaltkreise mit **unbeschränktem** Fanin.

- (a) Berechne das **Boolesche Matrizenprodukt**

$$(A \cdot B)[i, j] = \bigvee_{k=1}^n A[i, k] \wedge B[k, j]$$

in Tiefe zwei mit  $O(n^3)$  Gattern.

- (b) Berechne die **transitive Hülle** eines gerichteten Graphen in Tiefe  $O(\log_2 n)$  und Größe  $O(n^3 \log_2 n)$ .

- (a) Das Matrizenprodukt: Der Brute-Force Ansatz funktioniert. ✓
- (b) Für die Adjazenzmatrix  $A_G$  zeige durch Induktion über  $d$ :
- ▶  $A_G^d[i, j] = 1 \iff$  es gibt einen Weg der Länge  $d$  von  $i$  nach  $j$  in  $G$ .

Wir betrachten uniforme Schaltkreise mit **unbeschränktem** Fanin.

- (a) Berechne das **Boolesche Matrizenprodukt**

$$(A \cdot B)[i, j] = \bigvee_{k=1}^n A[i, k] \wedge B[k, j]$$

in Tiefe zwei mit  $O(n^3)$  Gattern.

- (b) Berechne die **transitive Hülle** eines gerichteten Graphen in Tiefe  $O(\log_2 n)$  und Größe  $O(n^3 \log_2 n)$ .

- (a) Das Matrizenprodukt: Der Brute-Force Ansatz funktioniert. ✓

- (b) Für die Adjazenzmatrix  $A_G$  zeige durch Induktion über  $d$ :

- ▶  $A_G^d[i, j] = 1 \iff$  es gibt einen Weg der Länge  $d$  von  $i$  nach  $j$  in  $G$ .
- ▶  $E_n$  sei die  $n \times n$  Einheitsmatrix: Es gibt einen Weg von  $i$  nach  $j \iff$

$$(A_G \vee$$

Wir betrachten uniforme Schaltkreise mit **unbeschränktem** Fanin.

- (a) Berechne das **Boolesche Matrizenprodukt**

$$(A \cdot B)[i, j] = \bigvee_{k=1}^n A[i, k] \wedge B[k, j]$$

in Tiefe zwei mit  $O(n^3)$  Gattern.

- (b) Berechne die **transitive Hülle** eines gerichteten Graphen in Tiefe  $O(\log_2 n)$  und Größe  $O(n^3 \log_2 n)$ .

- (a) Das Matrizenprodukt: Der Brute-Force Ansatz funktioniert. ✓

- (b) Für die Adjazenzmatrix  $A_G$  zeige durch Induktion über  $d$ :

- ▶  $A_G^d[i, j] = 1 \iff$  es gibt einen Weg der Länge  $d$  von  $i$  nach  $j$  in  $G$ .
- ▶  $E_n$  sei die  $n \times n$  Einheitsmatrix: Es gibt einen Weg von  $i$  nach  $j \iff$

$$(A_G \vee E_n)^n[i, j] = 1.$$

✓ Es gibt genau dann einen Weg von  $i$  nach  $j$ , wenn

$$(A_G \vee E_n)^n[i, j] = 1.$$

! Berechne  $(A_G \vee E_n)^n$ .



- ✓ Es gibt genau dann einen Weg von  $i$  nach  $j$ , wenn

$$(A_G \vee E_n)^n[i, j] = 1.$$

! Berechne  $(A_G \vee E_n)^n$ .

- Ein Boolesches Matrizenprodukt kann (mit unbeschränktem Fanin) in konstanter Tiefe und Größe  $O(n^3)$  ausgeführt werden.

- ✓ Es gibt genau dann einen Weg von  $i$  nach  $j$ , wenn

$$(A_G \vee E_n)^n[i, j] = 1.$$

! Berechne  $(A_G \vee E_n)^n$ .

- Ein Boolesches Matrizenprodukt kann (mit unbeschränktem Fanin) in konstanter Tiefe und Größe  $O(n^3)$  ausgeführt werden.
- Wenn  $N > n$  die nächstgrößere Zweierpotenz ist, dann benutze **iteriertes Quadrieren**

$$(A_G \vee E_n)^{2^{k+1}} = ((A_G \vee E_n)^{2^k})^2,$$

um  $(A_G \vee E_n)^N$  in **Tiefe**  $O(\log_2 n)$  und **Größe**  $O(n^3 \log_2 n)$  zu berechnen.

$s : \mathbb{N} \rightarrow \mathbb{N}$  mit  $s(n) = \Omega(\log_2 n)$  sei platz-konstruierbar. Dann ist

$$\text{DSPACE}(s) \subseteq \text{NSPACE}(s) \subseteq \text{DEPTH-SIZE}_{\text{uniform}}(s^2, 2^{O(s)}) \subseteq \text{DSPACE}(s^2).$$

(Wir erhalten den *Satz von Savitch* als Konsequenz!)

$s : \mathbb{N} \rightarrow \mathbb{N}$  mit  $s(n) = \Omega(\log_2 n)$  sei platz-konstruierbar. Dann ist

$$\text{DSPACE}(s) \subseteq \text{NSPACE}(s) \subseteq \text{DEPTH-SIZE}_{\text{uniform}}(s^2, 2^{O(s)}) \subseteq \text{DSPACE}(s^2).$$

(Wir erhalten den *Satz von Savitch* als Konsequenz!)

Zu zeigen  $\text{NSPACE}(s) \subseteq \text{DEPTH-SIZE}_{\text{uniform}}(s^2, 2^{O(s)})$ .

- Sei  $M$  eine nichtdeterministische TM mit  $\text{NSPACE}_M(w) = O(s(|w|))$ .

$s : \mathbb{N} \rightarrow \mathbb{N}$  mit  $s(n) = \Omega(\log_2 n)$  sei platz-konstruierbar. Dann ist

$$\text{DSPACE}(s) \subseteq \text{NSPACE}(s) \subseteq \text{DEPTH-SIZE}_{\text{uniform}}(s^2, 2^{O(s)}) \subseteq \text{DSPACE}(s^2).$$

(Wir erhalten den *Satz von Savitch* als Konsequenz!)

Zu zeigen  $\text{NSPACE}(s) \subseteq \text{DEPTH-SIZE}_{\text{uniform}}(s^2, 2^{O(s)})$ .

- Sei  $M$  eine nichtdeterministische TM mit  $\text{NSPACE}_M(w) = O(s(|w|))$ .
- Für Eingabe  $w$  ist zu entscheiden, ob es einen Weg vom Startknoten zu einem akzeptierenden Knoten im Berechnungsgraphen  $G_M(w)$  gibt.

$s : \mathbb{N} \rightarrow \mathbb{N}$  mit  $s(n) = \Omega(\log_2 n)$  sei platz-konstruierbar. Dann ist

$$\text{DSPACE}(s) \subseteq \text{NSPACE}(s) \subseteq \text{DEPTH-SIZE}_{\text{uniform}}(s^2, 2^{O(s)}) \subseteq \text{DSPACE}(s^2).$$

(Wir erhalten den *Satz von Savitch* als Konsequenz!)

Zu zeigen  $\text{NSPACE}(s) \subseteq \text{DEPTH-SIZE}_{\text{uniform}}(s^2, 2^{O(s)})$ .

- Sei  $M$  eine nichtdeterministische TM mit  $\text{NSPACE}_M(w) = O(s(|w|))$ .
- Für Eingabe  $w$  ist zu entscheiden, ob es einen Weg vom Startknoten zu einem akzeptierenden Knoten im Berechnungsgraphen  $G_M(w)$  gibt.
  - ▶ Man kann annehmen, dass es genau eine akzeptierende Konfiguration gibt.
  - ▶ Wir müssen D-REACHABILITY für  $G_M(w)$  lösen. D-REACHABILITY ist aber ein Spezialfall der Berechnung der transitiven Hülle.
  - ▶  $G_M(w)$  hat  $\leq N = 2^{O(s(|w|))}$  Knoten  $\implies$

$s : \mathbb{N} \rightarrow \mathbb{N}$  mit  $s(n) = \Omega(\log_2 n)$  sei platz-konstruierbar. Dann ist

$$\text{DSPACE}(s) \subseteq \text{NSPACE}(s) \subseteq \text{DEPTH-SIZE}_{\text{uniform}}(s^2, 2^{O(s)}) \subseteq \text{DSPACE}(s^2).$$

(Wir erhalten den *Satz von Savitch* als Konsequenz!)

Zu zeigen  $\text{NSPACE}(s) \subseteq \text{DEPTH-SIZE}_{\text{uniform}}(s^2, 2^{O(s)})$ .

- Sei  $M$  eine nichtdeterministische TM mit  $\text{NSPACE}_M(w) = O(s(|w|))$ .
- Für Eingabe  $w$  ist zu entscheiden, ob es einen Weg vom Startknoten zu einem akzeptierenden Knoten im Berechnungsgraphen  $G_M(w)$  gibt.
  - ▶ Man kann annehmen, dass es genau eine akzeptierende Konfiguration gibt.
  - ▶ Wir müssen D-REACHABILITY für  $G_M(w)$  lösen. D-REACHABILITY ist aber ein Spezialfall der Berechnung der transitiven Hülle.
  - ▶  $G_M(w)$  hat  $\leq N = 2^{O(s(|w|))}$  Knoten  $\implies$   
 D-REACHABILITY wird durch eine uniforme Schaltkreisfamilie vom Fanin 2
    - ★ in Tiefe  $O(\log_2^2 N) =$

$s : \mathbb{N} \rightarrow \mathbb{N}$  mit  $s(n) = \Omega(\log_2 n)$  sei platz-konstruierbar. Dann ist

$$\text{DSPACE}(s) \subseteq \text{NSPACE}(s) \subseteq \text{DEPTH-SIZE}_{\text{uniform}}(s^2, 2^{O(s)}) \subseteq \text{DSPACE}(s^2).$$

(Wir erhalten den *Satz von Savitch* als Konsequenz!)

Zu zeigen  $\text{NSPACE}(s) \subseteq \text{DEPTH-SIZE}_{\text{uniform}}(s^2, 2^{O(s)})$ .

- Sei  $M$  eine nichtdeterministische TM mit  $\text{NSPACE}_M(w) = O(s(|w|))$ .
- Für Eingabe  $w$  ist zu entscheiden, ob es einen Weg vom Startknoten zu einem akzeptierenden Knoten im Berechnungsgraphen  $G_M(w)$  gibt.
  - ▶ Man kann annehmen, dass es genau eine akzeptierende Konfiguration gibt.
  - ▶ Wir müssen D-REACHABILITY für  $G_M(w)$  lösen. D-REACHABILITY ist aber ein Spezialfall der Berechnung der transitiven Hülle.
  - ▶  $G_M(w)$  hat  $\leq N = 2^{O(s(|w|))}$  Knoten  $\implies$   
 D-REACHABILITY wird durch eine uniforme Schaltkreisfamilie vom Fanin 2
    - ★ in Tiefe  $O(\log_2^2 N) = O(s(|w|)^2)$  und Größe  $N^3 \log N =$



$s : \mathbb{N} \rightarrow \mathbb{N}$  mit  $s(n) = \Omega(\log_2 n)$  sei platz-konstruierbar. Dann ist

$$\text{DSPACE}(s) \subseteq \text{NSPACE}(s) \subseteq \text{DEPTH-SIZE}_{\text{uniform}}(s^2, 2^{O(s)}) \subseteq \text{DSPACE}(s^2).$$

(Wir erhalten den *Satz von Savitch* als Konsequenz!)

Zu zeigen  $\text{NSPACE}(s) \subseteq \text{DEPTH-SIZE}_{\text{uniform}}(s^2, 2^{O(s)})$ .

- Sei  $M$  eine nichtdeterministische TM mit  $\text{NSPACE}_M(w) = O(s(|w|))$ .
- Für Eingabe  $w$  ist zu entscheiden, ob es einen Weg vom Startknoten zu einem akzeptierenden Knoten im Berechnungsgraphen  $G_M(w)$  gibt.
  - ▶ Man kann annehmen, dass es genau eine akzeptierende Konfiguration gibt.
  - ▶ Wir müssen D-REACHABILITY für  $G_M(w)$  lösen. D-REACHABILITY ist aber ein Spezialfall der Berechnung der transitiven Hülle.
  - ▶  $G_M(w)$  hat  $\leq N = 2^{O(s(|w|))}$  Knoten  $\implies$   
 D-REACHABILITY wird durch eine uniforme Schaltkreisfamilie vom Fanin 2
    - ★ in **Tiefe**  $O(\log_2^2 N) = O(s(|w|)^2)$  und **Größe**  $N^3 \log N = 2^{O(s(|w|))}$  berechnet.

(a)  $NC^1 \subseteq DL$

(a)  $NC^1 \subseteq DL \subseteq NL$

(a)  $NC^1 \subseteq DL \subseteq NL \subseteq DEPTH-SIZE_{uniform}(\log_2^2 n, poly(n)) = NC^2$ .

- ▶ Alle Sprachen in NL sind parallelisierbar :-))

(a)  $NC^1 \subseteq DL \subseteq NL \subseteq DEPTH-SIZE_{\text{uniform}}(\log_2^2 n, \text{poly}(n)) = NC^2$ .

- ▶ Alle Sprachen in NL sind parallelisierbar :-))

(b) D-REACHABILITY liegt in NC<sup>2</sup>.

# D-REACHABILITY, DL, NL und NC<sup>1</sup>

(a)  $NC^1 \subseteq DL \subseteq NL \subseteq DEPTH-SIZE_{\text{uniform}}(\log_2^2 n, \text{poly}(n)) = NC^2$ .

▶ Alle Sprachen in NL sind parallelisierbar :-))

(b) D-REACHABILITY liegt in NC<sup>2</sup>.

(c) Wenn D-REACHABILITY in NC<sup>1</sup> liegen würde, dann

# D-REACHABILITY, $DL$ , $NL$ und $NC^1$

(a)  $NC^1 \subseteq DL \subseteq NL \subseteq DEPTH-SIZE_{uniform}(\log_2^2 n, poly(n)) = NC^2$ .

- ▶ Alle Sprachen in  $NL$  sind parallelisierbar :-))

(b) D-REACHABILITY liegt in  $NC^2$ .

(c) Wenn D-REACHABILITY in  $NC^1$  liegen würde, dann

- ▶ liegt D-REACHABILITY in  $DL$
- ▶ und  $DL = NL$  folgt.

# D-REACHABILITY, $DL$ , $NL$ und $NC^1$

(a)  $NC^1 \subseteq DL \subseteq NL \subseteq DEPTH-SIZE_{uniform}(\log_2^2 n, poly(n)) = NC^2$ .

- ▶ Alle Sprachen in  $NL$  sind parallelisierbar :-))

(b) D-REACHABILITY liegt in  $NC^2$ .

(c) Wenn D-REACHABILITY in  $NC^1$  liegen würde, dann

- ▶ liegt D-REACHABILITY in  $DL$
- ▶ und  $DL = NL$  folgt.

Also „wahrscheinlich“

$D-REACHABILITY \in DEPTH-SIZE(\log_2^2, poly(n)) \setminus NC^1$ .



# P-Vollständigkeit

Wir wählen die LOGSPACE-Reduzierbarkeit, um die Parallelisierbarkeit zweier Sprachen zu vergleichen. Warum?

(a) Es ist  $DL \subseteq NC^2$ .

# Die LOGSPACE-Reduktion

Wir wählen die LOGSPACE-Reduzierbarkeit, um die Parallelisierbarkeit zweier Sprachen zu vergleichen. Warum?

(a) Es ist  $DL \subseteq NC^2$ .

(b) Aus  $L_1 \leq_{\text{LOG}} L_2$  und  $L_2 \in NC$  folgt  $L_1 \in NC$ .

▶  $L_1 \leq_{\text{LOG}} L_2 \implies$  Es gibt eine TM mit  $DSPACE_M(w) = O(\log_2 |w|)$  und

$$w \in L_1 \iff M(w) \in L_2.$$

# Die LOGSPACE-Reduktion

Wir wählen die LOGSPACE-Reduzierbarkeit, um die Parallelisierbarkeit zweier Sprachen zu vergleichen. Warum?

(a) Es ist  $DL \subseteq NC^2$ .

(b) Aus  $L_1 \leq_{\text{LOG}} L_2$  und  $L_2 \in NC$  folgt  $L_1 \in NC$ .

▶  $L_1 \leq_{\text{LOG}} L_2 \implies$  Es gibt eine TM mit  $DSPACE_M(w) = O(\log_2 |w|)$  und

$$w \in L_1 \iff M(w) \in L_2.$$

▶ Jedes Ausgabebit von  $M$  kann durch eine uniforme Schaltkreisfamilie in polynomieller Größe und Tiefe  $O(\log_2^2 n)$  berechnet werden.

# Die LOGSPACE-Reduktion

Wir wählen die LOGSPACE-Reduzierbarkeit, um die Parallelisierbarkeit zweier Sprachen zu vergleichen. Warum?

(a) Es ist  $DL \subseteq NC^2$ .

(b) Aus  $L_1 \leq_{LOG} L_2$  und  $L_2 \in NC$  folgt  $L_1 \in NC$ .

▶  $L_1 \leq_{LOG} L_2 \implies$  Es gibt eine TM mit  $DSPACE_M(w) = O(\log_2 |w|)$  und

$$w \in L_1 \iff M(w) \in L_2.$$

- ▶ Jedes Ausgabebit von  $M$  kann durch eine uniforme Schaltkreisfamilie in polynomieller Größe und Tiefe  $O(\log_2^2 n)$  berechnet werden.
- ▶ Wenn  $L_2$  durch eine uniforme Schaltkreisfamilie (mit polynomieller Größe und polylogarithmischer Tiefe) erkannt wird, so ist auch  $L_1$  in polynomieller Größe und polylogarithmischer Tiefe erkennbar. Also

# Die LOGSPACE-Reduktion

Wir wählen die LOGSPACE-Reduzierbarkeit, um die Parallelisierbarkeit zweier Sprachen zu vergleichen. Warum?

(a) Es ist  $DL \subseteq NC^2$ .

(b) Aus  $L_1 \leq_{\text{LOG}} L_2$  und  $L_2 \in NC$  folgt  $L_1 \in NC$ .

▶  $L_1 \leq_{\text{LOG}} L_2 \implies$  Es gibt eine TM mit  $DSPACE_M(w) = O(\log_2 |w|)$  und

$$w \in L_1 \iff M(w) \in L_2.$$

- ▶ Jedes Ausgabebit von  $M$  kann durch eine uniforme Schaltkreisfamilie in polynomieller Größe und Tiefe  $O(\log_2^2 n)$  berechnet werden.
- ▶ Wenn  $L_2$  durch eine uniforme Schaltkreisfamilie (mit polynomieller Größe und polylogarithmischer Tiefe) erkannt wird, so ist auch  $L_1$  in polynomieller Größe und polylogarithmischer Tiefe erkennbar. Also

$$L_2 \in NC \implies L_1 \in NC.$$

- (a) Eine Sprache  $L$  heißt genau dann *P-hart*, wenn  $K \leq_{\text{LOG}} L$  für alle Sprachen  $K \in \text{P}$  gilt.
- (b) Eine Sprache  $L$  heißt genau dann *P-vollständig*, wenn  $L \in \text{P}$  und wenn  $L$  P-hart ist.

- (a) Eine Sprache  $L$  heißt genau dann  $P$ -hart, wenn  $K \leq_{\text{LOG}} L$  für alle Sprachen  $K \in P$  gilt.
- (b) Eine Sprache  $L$  heißt genau dann  $P$ -vollständig, wenn  $L \in P$  und wenn  $L$   $P$ -hart ist.

Wir wissen: Wenn  $L \leq_{\text{LOG}} K$  und wenn  $K \in \text{NC}$ , dann ist  $L \in \text{NC}$ .

Wenn irgendeine  $P$ -vollständige Sprache in  $\text{NC}$  liegt, dann folgt  $\text{NC} = P$ .

Es ist nicht zu erwarten, dass  $P$ -vollständige Sprachen parallelisierbar sind.



Circuit Value ist  $\mathcal{P}$ -vollständig

# Das Circuit Value Problem (CVP)

Wir nehmen an, daß der Fanin für alle zu betrachtenden Schaltkreise höchstens zwei ist und betrachten die folgenden Auswertungsprobleme:

**CVP** =  $\{ \langle S \rangle x \mid \text{der Schaltkreis } S \text{ akzeptiert Eingabe } x \}$

**M-CVP** =  $\{ \langle S \rangle x \mid \text{der monotone Schaltkreis } S \text{ akzeptiert Eingabe } x \}$

Ein Schaltkreis ohne Negationsgatter heißt monoton.

**NOR-CVP** =  $\{ \langle S \rangle x \mid \text{der NOR-Schaltkreis } S \text{ akzeptiert Eingabe } x \}$ .

# Das Circuit Value Problem (CVP)

Wir nehmen an, daß der Fanin für alle zu betrachtenden Schaltkreise höchstens zwei ist und betrachten die folgenden Auswertungsprobleme:

**CVP** =  $\{ \langle S \rangle x \mid \text{der Schaltkreis } S \text{ akzeptiert Eingabe } x \}$

**M-CVP** =  $\{ \langle S \rangle x \mid \text{der monotone Schaltkreis } S \text{ akzeptiert Eingabe } x \}$

Ein Schaltkreis ohne Negationsgatter heißt monoton.

**NOR-CVP** =  $\{ \langle S \rangle x \mid \text{der NOR-Schaltkreis } S \text{ akzeptiert Eingabe } x \}$ .

- **CVP** spielt als *generisches* Problem für die  $P$ -Vollständigkeit dieselbe Rolle wie **KNFSAT** für die  $NP$ -Vollständigkeit.
- **CVP**, **M-CVP** und **NOR-CVP** liegen alle in  $P$ .

Zeige  $L \leq_{\text{LOG}} \text{CVP}$  für eine beliebige Sprache  $L \in \mathbb{P}$ .

Zeige  $L \leq_{\text{LOG}} \text{CVP}$  für eine beliebige Sprache  $L \in P$ .

- $L = L(M)$  für eine deterministische Turingmaschine  $M$ , die  $L$  in höchstens  $\text{poly}(n)$  Schritten akzeptiert.
- Simuliere  $M$  für Eingaben der Länge  $n$  durch einen Schaltkreis  $S_n$ .

Zeige  $L \leq_{\text{LOG}} \text{CVP}$  für eine beliebige Sprache  $L \in \text{P}$ .

- $L = L(M)$  für eine deterministische Turingmaschine  $M$ , die  $L$  in höchstens  $\text{poly}(n)$  Schritten akzeptiert.
- Simuliere  $M$  für Eingaben der Länge  $n$  durch einen Schaltkreis  $S_n$ .
  - ▶  $S_n$  hat die Grobstruktur eines zwei-dimensionalen Gitters.
  - ▶ Die  $i$ -te „Zeile“ des Gitters gibt Bandinhalt, Kopfposition und Zustand von  $M$  zum Zeitpunkt  $i$  wieder.
    - ★ Die  $i$ -te Zeile von  $S$  ist aus kleinen Schaltkreisen  $S_{i,j}$  konstanter Größe aufgebaut, wobei  $S_{i,j}$  die Zelle  $j$  des Bandes zum Zeitpunkt  $i$  simuliert.
    - ★  $S_{i,j}$  muß den von Schaltkreis  $S_{i-1,j}$  berechneten Bandinhalt speichern können, falls der Kopf von  $M$  zum Zeitpunkt  $i$  die Zelle  $j$  **nicht** besucht,
    - ★ Sonst muß  $S_{i,j}$  den Bandinhalt verändern sowie den neuen Zustand und die neue Richtung festlegen, abhängig vom gegenwärtigen Zustand und Bandinhalt.

Zeige  $L \leq_{\text{LOG}} \text{CVP}$  für eine beliebige Sprache  $L \in P$ .

- $L = L(M)$  für eine deterministische Turingmaschine  $M$ , die  $L$  in höchstens  $\text{poly}(n)$  Schritten akzeptiert.
- Simuliere  $M$  für Eingaben der Länge  $n$  durch einen Schaltkreis  $S_n$ .
  - ▶  $S_n$  hat die Grobstruktur eines zwei-dimensionalen Gitters.
  - ▶ Die  $i$ -te „Zeile“ des Gitters gibt Bandinhalt, Kopfposition und Zustand von  $M$  zum Zeitpunkt  $i$  wieder.
    - ★ Die  $i$ -te Zeile von  $S$  ist aus kleinen Schaltkreisen  $S_{i,j}$  konstanter Größe aufgebaut, wobei  $S_{i,j}$  die Zelle  $j$  des Bandes zum Zeitpunkt  $i$  simuliert.
    - ★  $S_{i,j}$  muß den von Schaltkreis  $S_{i-1,j}$  berechneten Bandinhalt speichern können, falls der Kopf von  $M$  zum Zeitpunkt  $i$  die Zelle  $j$  **nicht** besucht,
    - ★ Sonst muß  $S_{i,j}$  den Bandinhalt verändern sowie den neuen Zustand und die neue Richtung festlegen, abhängig vom gegenwärtigen Zustand und Bandinhalt.
  - ▶ Das ist alles kein Problem, wenn wir die Ausgänge von  $S_{i-1,j-1}$ ,  $S_{i-1,j}$  und  $S_{i-1,j+1}$  zu Eingängen von  $S_{i,j}$  machen.

- Sämtliche Schaltkreise  $S_{i,j}$  können „baugleich“ gewählt werden, mit Ausnahme der Schaltkreise  $S_{0,j}$ ,
    - ▶ die entweder das Blank-Symbol als Wert erhalten ( $j \notin \{1, \dots, n\}$ )
    - ▶ oder an die Eingabe anzuschliessen sind ( $j \in \{1, \dots, n\}$ ).
- ⇒  $S_n$  kann durch eine logarithmisch-platzbeschränkte Turingmaschine konstruiert werden.



- Sämtliche Schaltkreise  $S_{i,j}$  können „baugleich“ gewählt werden, mit Ausnahme der Schaltkreise  $S_{0,j}$ ,
    - ▶ die entweder das Blank-Symbol als Wert erhalten ( $j \notin \{1, \dots, n\}$ )
    - ▶ oder an die Eingabe anzuschliessen sind ( $j \in \{1, \dots, n\}$ ).
- ⇒  $S_n$  kann durch eine logarithmisch-platzbeschränkte Turingmaschine konstruiert werden.
- Wir müssen das Gitter noch „auswerten“, d.h. wir müssen feststellen, ob der letzte Zustand akzeptierend ist.
    - ▶ Setze einen binären Auswertungsbaum „auf“ das Gitter.

- Sämtliche Schaltkreise  $S_{i,j}$  können „baugleich“ gewählt werden, mit Ausnahme der Schaltkreise  $S_{0,j}$ ,
    - ▶ die entweder das Blank-Symbol als Wert erhalten ( $j \notin \{1, \dots, n\}$ )
    - ▶ oder an die Eingabe anzuschliessen sind ( $j \in \{1, \dots, n\}$ ).
- ⇒  $S_n$  kann durch eine logarithmisch-platzbeschränkte Turingmaschine konstruiert werden.
- Wir müssen das Gitter noch „auswerten“, d.h. wir müssen feststellen, ob der letzte Zustand akzeptierend ist.
    - ▶ Setze einen binären Auswertungsbaum „auf“ das Gitter.

Mit ähnlichen Methoden zeigt man:

**M-CVP** und **NOR-CVP** sind  $P$ -vollständig.

Im Beweis der P-Vollständigkeit von CVP wird eine  $t$ -zeitbeschränkte Turingmaschine durch einen uniformen Schaltkreis der Größe  $\mathcal{O}(n^2)$  simuliert  $\implies$

$$\text{DTIME}(t) \subseteq \text{SIZE}_{\text{uniform}}(t^2).$$

Im Beweis der P-Vollständigkeit von CVP wird eine  $t$ -zeitbeschränkte Turingmaschine durch einen uniformen Schaltkreis der Größe  $\mathcal{O}(n^2)$  simuliert  $\implies$

$$\text{DTIME}(t) \subseteq \text{SIZE}_{\text{uniform}}(t^2).$$

Definiere

$$P/\text{poly} := \bigcup_{k \in \mathbb{N}} \text{SIZE}(n^k)$$

als die Klasse aller Sprachen mit (mgl. **nicht-uniformen**) Schaltkreisen polyn. Größe.

Es ist

$$P = \bigcup_{k \in \mathbb{N}} \text{SIZE}_{\text{uniform}}(n^k) \subset P/\text{poly}.$$

Im Beweis der P-Vollständigkeit von CVP wird eine  $t$ -zeitbeschränkte Turingmaschine durch einen uniformen Schaltkreis der Größe  $\mathcal{O}(n^2)$  simuliert  $\implies$

$$\text{DTIME}(t) \subseteq \text{SIZE}_{\text{uniform}}(t^2).$$

Definiere

$$P/\text{poly} := \bigcup_{k \in \mathbb{N}} \text{SIZE}(n^k)$$

als die Klasse aller Sprachen mit (mgl. **nicht-uniformen**) Schaltkreisen polyn. Größe.

Es ist

$$P = \bigcup_{k \in \mathbb{N}} \text{SIZE}_{\text{uniform}}(n^k) \subset P/\text{poly}.$$

Wenn  $L \notin P/\text{poly}$  für eine Sprache  $L \in \text{NP}$  gilt, dann folgt  $P \neq \text{NP}$ .

- Leider scheitert dieser Ansatz für den Beweis von  $P \neq \text{NP}$  bisher kläglich.

Die Lineare Programmierung ist  $\mathbb{P}$ -vollständig.

# Die Lineare Programmierung

$A$  ist eine  $m \times n$  Matrix von ganzen Zahlen,  $b \in \mathbb{Z}^m$  und  $c \in \mathbb{Z}^n$  sind Vektoren ganzer Zahlen.

- (a) Im *Problem der linearen Ungleichungen* ist zu entscheiden, ob es einen Vektor  $x \in \mathbb{Q}^n$  mit  $Ax \leq b$  gibt.
- (b) Im *Problem der linearen Programmierung* ist zu entscheiden, ob es einen Vektor  $x \in \mathbb{Q}^n$  mit  $Ax \leq b$  und  $c \cdot x \geq t$  gibt.

# Die Lineare Programmierung

$A$  ist eine  $m \times n$  Matrix von ganzen Zahlen,  $b \in \mathbb{Z}^m$  und  $c \in \mathbb{Z}^n$  sind Vektoren ganzer Zahlen.

- (a) Im *Problem der linearen Ungleichungen* ist zu entscheiden, ob es einen Vektor  $x \in \mathbb{Q}^n$  mit  $Ax \leq b$  gibt.
- (b) Im *Problem der linearen Programmierung* ist zu entscheiden, ob es einen Vektor  $x \in \mathbb{Q}^n$  mit  $Ax \leq b$  und  $c \cdot x \geq t$  gibt.

Das Problem der linearen Programmierung wird konventionell als Optimierungsproblem formuliert:

Maximiere  $c \cdot x$ , so daß  $Ax \leq b$  und  $x \geq 0$  gilt.



# Die Lineare Programmierung

$A$  ist eine  $m \times n$  Matrix von ganzen Zahlen,  $b \in \mathbb{Z}^m$  und  $c \in \mathbb{Z}^n$  sind Vektoren ganzer Zahlen.

- (a) Im *Problem der linearen Ungleichungen* ist zu entscheiden, ob es einen Vektor  $x \in \mathbb{Q}^n$  mit  $Ax \leq b$  gibt.
- (b) Im *Problem der linearen Programmierung* ist zu entscheiden, ob es einen Vektor  $x \in \mathbb{Q}^n$  mit  $Ax \leq b$  und  $c \cdot x \geq t$  gibt.

Das Problem der linearen Programmierung wird konventionell als Optimierungsproblem formuliert:

Maximiere  $c \cdot x$ , so daß  $Ax \leq b$  und  $x \geq 0$  gilt.

Zeige:

- (a) Das Problem der linearen Ungleichungen ist  $P$ -vollständig.
- (b) Das Problem der linearen Programmierung ist  $P$ -vollständig.

# Das Problem der linearen Ungleichungen ist $P$ -vollständig

- Wir zeigen die Reduktion  $M-CVP \leq_{LOG} \text{Lineare Ungleichungen}$ .
- Sei  $(S, x)$  eine Eingabe für  $M-CVP$ . Weise jedem Gatter von  $S$  Ungleichungen zu.
  - ▶ Beschreibe die Quelle zu Eingabe  $x_i$ :
    - ★ Falls  $x_i = 0$ , verwende die Ungleichungen

# Das Problem der linearen Ungleichungen ist $\mathcal{P}$ -vollständig

- Wir zeigen die Reduktion  $\text{M-CVP} \leq_{\text{LOG}} \text{Lineare Ungleichungen}$ .
- Sei  $(S, x)$  eine Eingabe für M-CVP. Weise jedem Gatter von  $S$  Ungleichungen zu.
  - ▶ Beschreibe die Quelle zu Eingabe  $x_i$ :
    - ★ Falls  $x_i = 0$ , verwende die Ungleichungen  $x_i \leq 0$  und  $-x_i \leq 0$ .

# Das Problem der linearen Ungleichungen ist $\mathcal{P}$ -vollständig

- Wir zeigen die Reduktion  $\text{M-CVP} \leq_{\text{LOG}} \text{Lineare Ungleichungen}$ .
- Sei  $(S, x)$  eine Eingabe für M-CVP. Weise jedem Gatter von  $S$  Ungleichungen zu.
  - ▶ Beschreibe die Quelle zu Eingabe  $x_i$ :
    - ★ Falls  $x_i = 0$ , verwende die Ungleichungen  $x_i \leq 0$  und  $-x_i \leq 0$ .
    - ★ Falls  $x_i = 1$ , verwende die Ungleichungen  $x_i \leq 1$  und  $-x_i \leq -1$ .

# Das Problem der linearen Ungleichungen ist P-vollständig

- Wir zeigen die Reduktion  $M-CVP \leq_{LOG} \text{Lineare Ungleichungen}$ .
- Sei  $(S, x)$  eine Eingabe für M-CVP. Weise jedem Gatter von  $S$  Ungleichungen zu.
  - ▶ Beschreibe die Quelle zu Eingabe  $x_i$ :
    - ★ Falls  $x_i = 0$ , verwende die Ungleichungen  $x_i \leq 0$  und  $-x_i \leq 0$ .
    - ★ Falls  $x_i = 1$ , verwende die Ungleichungen  $x_i \leq 1$  und  $-x_i \leq -1$ .
  - ▶ Für ein Gatter  $v \equiv u \wedge w$  verwende die Ungleichungen:

# Das Problem der linearen Ungleichungen ist $P$ -vollständig

- Wir zeigen die Reduktion  $M-CVP \leq_{LOG}$  Lineare Ungleichungen.
- Sei  $(S, x)$  eine Eingabe für M-CVP. Weise jedem Gatter von  $S$  Ungleichungen zu.
  - ▶ Beschreibe die Quelle zu Eingabe  $x_i$ :
    - ★ Falls  $x_i = 0$ , verwende die Ungleichungen  $x_i \leq 0$  und  $-x_i \leq 0$ .
    - ★ Falls  $x_i = 1$ , verwende die Ungleichungen  $x_i \leq 1$  und  $-x_i \leq -1$ .
  - ▶ Für ein Gatter  $v \equiv u \wedge w$  verwende die Ungleichungen:  
 $v \leq u$ ,  $v \leq w$ ,  $u + w - 1 \leq v$ ,  $0 \leq v$ .
  - ▶ Für ein Gatter  $v \equiv u \vee w$  verwende die Ungleichungen:

# Das Problem der linearen Ungleichungen ist $P$ -vollständig

- Wir zeigen die Reduktion  $M-CVP \leq_{LOG}$  Lineare Ungleichungen.
- Sei  $(S, x)$  eine Eingabe für  $M-CVP$ . Weise jedem Gatter von  $S$  Ungleichungen zu.
  - ▶ Beschreibe die Quelle zu Eingabe  $x_i$ :
    - ★ Falls  $x_i = 0$ , verwende die Ungleichungen  $x_i \leq 0$  und  $-x_i \leq 0$ .
    - ★ Falls  $x_i = 1$ , verwende die Ungleichungen  $x_i \leq 1$  und  $-x_i \leq -1$ .
  - ▶ Für ein Gatter  $v \equiv u \wedge w$  verwende die Ungleichungen:  
 $v \leq u$ ,  $v \leq w$ ,  $u + w - 1 \leq v$ ,  $0 \leq v$ .
  - ▶ Für ein Gatter  $v \equiv u \vee w$  verwende die Ungleichungen:  
 $u \leq v$ ,  $w \leq v$ ,  $v \leq u + w$ ,  $v \leq 1$ .

# Das Problem der linearen Ungleichungen ist P-vollständig

- Wir zeigen die Reduktion  $M-CVP \leq_{LOG}$  Lineare Ungleichungen.
- Sei  $(S, x)$  eine Eingabe für M-CVP. Weise jedem Gatter von  $S$  Ungleichungen zu.
  - ▶ Beschreibe die Quelle zu Eingabe  $x_i$ :
    - ★ Falls  $x_i = 0$ , verwende die Ungleichungen  $x_i \leq 0$  und  $-x_i \leq 0$ .
    - ★ Falls  $x_i = 1$ , verwende die Ungleichungen  $x_i \leq 1$  und  $-x_i \leq -1$ .
  - ▶ Für ein Gatter  $v \equiv u \wedge w$  verwende die Ungleichungen:  
 $v \leq u$ ,  $v \leq w$ ,  $u + w - 1 \leq v$ ,  $0 \leq v$ .
  - ▶ Für ein Gatter  $v \equiv u \vee w$  verwende die Ungleichungen:  
 $u \leq v$ ,  $w \leq v$ ,  $v \leq u + w$ ,  $v \leq 1$ .
  - ▶ Füge die Ungleichung  $-s \leq -1$  für die eindeutig bestimmte Senke  $s$  des Schaltkreises hinzu.



# Das Problem der linearen Ungleichungen ist $P$ -vollständig

- Wir zeigen die Reduktion  $M-CVP \leq_{LOG}$  Lineare Ungleichungen.
- Sei  $(S, x)$  eine Eingabe für  $M-CVP$ . Weise jedem Gatter von  $S$  Ungleichungen zu.
  - ▶ Beschreibe die Quelle zu Eingabe  $x_i$ :
    - ★ Falls  $x_i = 0$ , verwende die Ungleichungen  $x_i \leq 0$  und  $-x_i \leq 0$ .
    - ★ Falls  $x_i = 1$ , verwende die Ungleichungen  $x_i \leq 1$  und  $-x_i \leq -1$ .
  - ▶ Für ein Gatter  $v \equiv u \wedge w$  verwende die Ungleichungen:  
 $v \leq u$ ,  $v \leq w$ ,  $u + w - 1 \leq v$ ,  $0 \leq v$ .
  - ▶ Für ein Gatter  $v \equiv u \vee w$  verwende die Ungleichungen:  
 $u \leq v$ ,  $w \leq v$ ,  $v \leq u + w$ ,  $v \leq 1$ .
  - ▶ Füge die Ungleichung  $-s \leq -1$  für die eindeutig bestimmte Senke  $s$  des Schaltkreises hinzu.
- Zeige durch Induktion über die topologische Ordnung der Gatter:  
das lineare Ungleichungssystem ist lösbar  $\iff S$  akzeptiert  $x$ .

# Das Problem der linearen Programmierung ist P-vollständig

Zeige die Reduktion

Lineare Ungleichungen  $\leq_{\text{LOG}}$  Lineare Programmierung.

# Das Problem der linearen Programmierung ist P-vollständig

Zeige die Reduktion

Lineare Ungleichungen  $\leq_{\text{LOG}}$  Lineare Programmierung.

Wenn  $(A, b)$  eine Eingabe für das Problem der linearen Programmierung ist,

- dann übernahm  $A, b$  und
- setze  $c = \vec{0}$  und  $t = 0$ .

# Die Parallelisierbarkeit von Greedy-Algorithmen

# Eine Heuristik für das Independent-Set Problem

## Das Independent Set Problem

*Eingabe:*  $G = (V, E)$  ist ein ungerichteter Graph mit Knotenmenge  $V = \{1, \dots, n\}$ .

*Aufgabe:* Bestimme eine größtmögliche unabhängige Knotenmenge. (Eine Knotenmenge ist *unabhängig*, wenn keine zwei Knoten der Menge durch eine Kante verbunden sind.)

# Eine Heuristik für das Independent-Set Problem

## Das Independent Set Problem

*Eingabe:*  $G = (V, E)$  ist ein ungerichteter Graph mit Knotenmenge  $V = \{1, \dots, n\}$ .

*Aufgabe:* Bestimme eine größtmögliche unabhängige Knotenmenge. (Eine Knotenmenge ist *unabhängig*, wenn keine zwei Knoten der Menge durch eine Kante verbunden sind.)

## Die Independent Set Heuristik

- 1 Setze  $I(G) := \emptyset$ .
- 2 FOR  $v = 1$  TO  $n$  DO  
IF ( $v$  ist mit keinem Knoten in  $I(G)$  verbunden) THEN  
 $I(G) = I(G) \cup \{v\}$ .
- 3 Die Menge  $I(G)$  wird ausgegeben.

# Eine Heuristik für das Independent-Set Problem

## Das Independent Set Problem

**Eingabe:**  $G = (V, E)$  ist ein ungerichteter Graph mit Knotenmenge  $V = \{1, \dots, n\}$ .

**Aufgabe:** Bestimme eine größtmögliche unabhängige Knotenmenge. (Eine Knotenmenge ist *unabhängig*, wenn keine zwei Knoten der Menge durch eine Kante verbunden sind.)

## Die Independent Set Heuristik

- 1 Setze  $I(G) := \emptyset$ .
- 2 FOR  $v = 1$  TO  $n$  DO  
IF ( $v$  ist mit keinem Knoten in  $I(G)$  verbunden) THEN  
 $I(G) = I(G) \cup \{v\}$ .
- 3 Die Menge  $I(G)$  wird ausgegeben.

Lässt sich diese Heuristik parallelisieren?

Kann für jeden Knoten  $v$  super-schnell festgestellt werden, ob  $v \in I(G)$ ?

## Lexicographically-First-Maximal-Independent-Set-Problem (LFMIS)

*Eingabe:*  $G = (V, E)$  ist ein ungerichteter Graph mit  $V = \{1, \dots, n\}$  und  $v \in V$  ist ein ausgezeichneteter Knoten.

*Aufgabe:* Die Menge  $I = I(G)$  werden von der Independent Set Heuristik berechnet. Entscheide, ob  $v$  in der Menge  $I$  vorkommt.

LFMIS ist P-vollständig.



Zeige die Reduktion  $\text{NOR-CVP} \leq_{\text{LOG}} \text{LFMIS}$ .

- $(S, x)$  – mit Graphstruktur  $G = (V, E)$  – sei Eingabe von NOR-CVP.

Zeige die Reduktion  $\text{NOR-CVP} \leq_{\text{LOG}} \text{LFMIS}$ .

- $(S, x)$  – mit Graphstruktur  $G = (V, E)$  – sei Eingabe von NOR-CVP.
- Konstruiere einen **ungerichteten** Graphen  $G^* = (V^*, E^*)$  für LFMIS:

Zeige die Reduktion  $\text{NOR-CVP} \leq_{\text{LOG}} \text{LFMIS}$ .

- $(S, x)$  –mit Graphstruktur  $G = (V, E)$ – sei Eingabe von NOR-CVP.
- Konstruiere einen **ungerichteten** Graphen  $G^* = (V^*, E^*)$  für LFMIS:
  - ▶ Füge einen neuen Knoten  $v_0$  hinzu und setze genau dann eine Kante  $\{v_0, i\}$  zum Eingabeknoten  $i$  ein, wenn  $x_i = 0$  ist.
  - ▶ Es ist also  $V^* = V \cup \{v_0\}$  und  $E^* = \{\{v_0, i\} \mid x_i = 0\} \cup E$ .

Zeige die Reduktion  $\text{NOR-CVP} \leq_{\text{LOG}} \text{LFMIS}$ .

- $(S, x)$  –mit Graphstruktur  $G = (V, E)$ – sei Eingabe von NOR-CVP.
- Konstruiere einen **ungerichteten** Graphen  $G^* = (V^*, E^*)$  für LFMIS:
  - ▶ Füge einen neuen Knoten  $v_0$  hinzu und setze genau dann eine Kante  $\{v_0, i\}$  zum Eingabeknoten  $i$  ein, wenn  $x_i = 0$  ist.
  - ▶ Es ist also  $V^* = V \cup \{v_0\}$  und  $E^* = \{\{v_0, i\} \mid x_i = 0\} \cup E$ .

- $v_0$  erhält den Namen „1“. Der Namen eines jeden anderen Knotens entspricht der topologischen Nummer wie vom Schaltkreis vorgegeben.
- Zeige durch *Induktion* über die zugewiesenen Namen:

$$v \in I(G^*) \iff v = v_0 \text{ oder das Gatter } v \text{ hat den Wert } 1.$$

Zeige  $v \in I(G^*) \iff v = v_0$  oder das Gatter  $v$  hat den Wert 1  
durch *Induktion* über die Tiefe von  $v$ .

Zeige  $v \in I(G^*) \iff v = v_0$  oder das Gatter  $v$  hat den Wert 1 durch *Induktion* über die Tiefe von  $v$ .

- Die Basis der Induktion.
  - ▶ Die Heuristik wählt stets den Knoten 1:  $v_0 \in I(G^*)$ .

Zeige  $v \in I(G^*) \iff v = v_0$  oder das Gatter  $v$  hat den Wert 1 durch *Induktion* über die Tiefe von  $v$ .

- Die Basis der Induktion.
  - ▶ Die Heuristik wählt stets den Knoten 1:  $v_0 \in I(G^*)$ .
  - ▶ Erhält die Quelle  $i$  den Wert Null, ist

Zeige  $v \in I(G^*) \iff v = v_0$  oder das Gatter  $v$  hat den Wert 1 durch *Induktion* über die Tiefe von  $v$ .

- Die Basis der Induktion.

- ▶ Die Heuristik wählt stets den Knoten 1:  $v_0 \in I(G^*)$ .
- ▶ Erhält die Quelle  $i$  den Wert Null, ist  $i \notin I(G^*)$

Die Heuristik kann  $i$  wegen der Kante  $\{v_0, i\}$  nicht wählen.

Hat Quelle  $i$  den Wert Eins, ist



Zeige  $v \in I(G^*) \iff v = v_0$  oder das Gatter  $v$  hat den Wert 1 durch *Induktion* über die Tiefe von  $v$ .

- Die Basis der Induktion.

- ▶ Die Heuristik wählt stets den Knoten 1:  $v_0 \in I(G^*)$ .
- ▶ Erhält die Quelle  $i$  den Wert Null, ist  $i \notin I(G^*)$

Die Heuristik kann  $i$  wegen der Kante  $\{v_0, i\}$  nicht wählen.

Hat Quelle  $i$  den Wert Eins, ist  $i \in I(G^*)$ .

Zeige  $v \in I(G^*) \iff v = v_0$  oder das Gatter  $v$  hat den Wert 1 durch *Induktion* über die Tiefe von  $v$ .

- Die Basis der Induktion.
  - ▶ Die Heuristik wählt stets den Knoten 1:  $v_0 \in I(G^*)$ .
  - ▶ Erhält die Quelle  $i$  den Wert Null, ist  $i \notin I(G^*)$   
Die Heuristik kann  $i$  wegen der Kante  $\{v_0, i\}$  nicht wählen.  
Hat Quelle  $i$  den Wert Eins, ist  $i \in I(G^*)$ .
- Der Induktionsschritt: Knoten  $w$  entspreche dem Gatter  $\text{nor}(u, v)$ .
  - ▶ Die topologische Nummer der Knoten  $u$  und  $v$  ist kleiner als die topologische Nummer von  $w$ .
  - ▶ Es ist  $w = \neg(u \vee v) = 1 \iff (u = 0) \wedge (v = 0)$ .

Zeige  $v \in I(G^*) \iff v = v_0$  oder das Gatter  $v$  hat den Wert 1 durch *Induktion* über die Tiefe von  $v$ .

- Die Basis der Induktion.
  - ▶ Die Heuristik wählt stets den Knoten 1:  $v_0 \in I(G^*)$ .
  - ▶ Erhält die Quelle  $i$  den Wert Null, ist  $i \notin I(G^*)$   
 Die Heuristik kann  $i$  wegen der Kante  $\{v_0, i\}$  nicht wählen.  
 Hat Quelle  $i$  den Wert Eins, ist  $i \in I(G^*)$ .
- Der Induktionsschritt: Knoten  $w$  entspreche dem Gatter  $\text{nor}(u, v)$ .
  - ▶ Die topologische Nummer der Knoten  $u$  und  $v$  ist kleiner als die topologische Nummer von  $w$ .
  - ▶ Es ist  $w = \neg(u \vee v) = 1 \iff (u = 0) \wedge (v = 0)$ .
  - ▶ Nach Induktionsannahme:

$$u \text{ (bzw. } v) \text{ gehört zu } I(G^*) \iff u = 1 \text{ (bzw. } v = 1).$$

Zeige  $v \in I(G^*) \iff v = v_0$  oder das Gatter  $v$  hat den Wert 1  
durch *Induktion* über die Tiefe von  $v$ .

- Die Basis der Induktion.
  - ▶ Die Heuristik wählt stets den Knoten 1:  $v_0 \in I(G^*)$ .
  - ▶ Erhält die Quelle  $i$  den Wert Null, ist  $i \notin I(G^*)$   
Die Heuristik kann  $i$  wegen der Kante  $\{v_0, i\}$  nicht wählen.  
Hat Quelle  $i$  den Wert Eins, ist  $i \in I(G^*)$ .
- Der Induktionsschritt: Knoten  $w$  entspreche dem Gatter  $\text{nor}(u, v)$ .
  - ▶ Die topologische Nummer der Knoten  $u$  und  $v$  ist kleiner als die topologische Nummer von  $w$ .
  - ▶ Es ist  $w = \neg(u \vee v) = 1 \iff (u = 0) \wedge (v = 0)$ .
  - ▶ Nach Induktionsannahme:

$$u \text{ (bzw. } v) \text{ gehört zu } I(G^*) \iff u = 1 \text{ (bzw. } v = 1).$$

- Wegen der Kanten  $\{u, w\}$  und  $\{v, w\}$  nimmt die Heuristik  $w$  genau dann in  $I(G^*)$  auf, wenn  $u$  und  $v$  den Wert 0 annehmen, d.h. gdw.  $w$  den Wert Eins hat.

# Zusammenfassung

- Wir haben Schaltkreise als paralleles Rechnermodell gewählt und die Klasse  $NC$  aller parallelisierbaren Sprachen eingeführt.
  - ▶ Es besteht ein enger Zusammenhang zwischen den Komplexitätsmaßen Tiefe und Speicherplatz:

$$DEPTH(s) \subseteq DSPACE(s) \subseteq NSPACE(s) \subseteq DEPTH-SIZE_{\text{uniform}}(s^2, 2^{O(s)}).$$

- ▶ Insbesondere folgt

$$NC^1 \subseteq DL \subseteq NL \subseteq NC^2$$

- ★ Alle Sprachen in  $NL$  sind parallelisierbar.
- ★ Wenn D-REACHABILITY in  $NC^1$  liegt, dann ist  $DL = NL$ .
- ▶ Die *Parallel-Computation-Thesis* postuliert

für jedes vernünftige parallele Rechnermodell  
polynomielle Beziehungen zwischen Rechenzeit- und Speicherplatzklassen.

- Die  $P$ -*vollständigen* Sprachen sind die im Hinblick auf Parallelisierbarkeit schwierigsten Sprachen in  $P$ .
  - ▶ Das Circuit-Value Problem spielt als generisches Problem dieselbe Rolle für die  $P$ -Vollständigkeit wie KNFSAT für die  $NP$ -Vollständigkeit.
  - ▶ Weitere  $P$ -vollständige Sprachen sind:
    - ★ die Lineare Programmierung,
    - ★ die Bestimmung der lexikographisch ersten maximalen unabhängigen Menge
    - ★ und das Wortproblem für kontextfreie Sprachen.