

Begleitmaterial zur Vorlesung

Komplexitätstheorie II

(Version 0.7)

Wintersemester 2004/2005

Friedhelm Meyer auf der Heide

Universität Paderborn

Heinz Nixdorf Institut

und

Institut für Informatik

Stand 09/04

Vorwort

Dieses Begleitmaterial zur Vorlesung „Komplexitätstheorie II“ beschreibt in kurzer Form die Inhalte dieser Vorlesung.

Es ersetzt kein Lehrbuch, und erst recht nicht den regelmässigen Besuch der Vorlesung und der Übungen.

Eine Liste empfohlener Lehrbücher findet sich auf der letzten Seite dieses Skriptes. Einige davon sind im Semesterapparat zusammengetragen. Weitere Informationen finden sich auf der Internetseite zu dieser Vorlesung.

Inhaltsverzeichnis

1	Einleitung	4
2	Vergleich von Rechenzeit und Speicherplatz	4
3	Schaltkreise, uniforme Schaltkreisfamilien und Parallele Rechenmodelle	18
3.1	Asymptotische Resultate über Schaltkreisgrößen	21
3.2	Uniforme Schaltkreisfamilien und Turingmaschinen	23
3.3	Parallele Komplexitätsklassen	27
3.4	Eine untere Schranke für PARITY	33
	Literatur	40

1 Einleitung

Diese Vorlesung baut auf der Vorlesung Komplexitätstheorie I auf. Die dort erarbeiteten Inhalte werden vorausgesetzt, ebenso Inhalte der Grundvorlesungen über Theoretische Informatik im Grundstudium. Es wird insbesondere das Konzept der NP-Vollständigkeit (polynomielle Reduktion, NP-vollständige Probleme) sowie das der Unentscheidbarkeit (Reduktion, Diagonalisierung, Unentscheidbarkeit des Halteproblems) benötigt.

2 Vergleich von Rechenzeit und Speicherplatz

Wir wollen uns in diesem Kapitel die Frage stellen, ob wir eine $t(n)$ -zeitbeschränkte Turingmaschine grundsätzlich durch eine $o(t(n))$ -platzbeschränkte Turingmaschine simulieren können, falls wir sehr viel mehr Zeit erlauben. Wir werden feststellen, dass dies tatsächlich der Fall ist, denn es gilt der folgende Satz:

Satz 2.1 (Time versus Space) Ist $\frac{t(n)}{\log t(n)} = \Omega(n)$, so gilt:

$$\text{DTIME}(t(n)) \subseteq \text{DTAPE}\left(\frac{t(n)}{\log t(n)}\right) .$$

Vor dem Beweis dieses Satzes folgern wir noch ein wichtiges Korollar.

Korollar 2.2 Ist $s(n)$ bandkonstruierbar und $\frac{t(n)}{\log t(n)} = o(s(n))$, so gilt:

$$\text{DTIME}(t(n)) \subsetneq \text{DTAPE}(s(n)) .$$

Insbesondere gilt dann: $\text{DTIME}(s(n)) \subsetneq \text{DTAPE}(s(n))$.

Beweis:

- $\text{DTIME}(t(n)) \stackrel{\text{Satz 2.1}}{\subseteq} \text{DTAPE}\left(\frac{t(n)}{\log t(n)}\right) \stackrel{*}{\subsetneq} \text{DTAPE}(s(n))$.

* gilt wegen des in Komplexitätstheorie I gezeigten Bandhierarchiesatzes (Satz 4.3)!

- Die zweite Aussage folgt aus der ersten, da $\frac{s(n)}{\log s(n)} = o(s(n))$.

□ Korollar 2.2

Bevor wir Satz 2.1 beweisen, beginnen wir mit Betrachtungen auf gänzlich anderen Rechenmodellen, den *arithmetischen Schaltkreisen*. Der arithmetische Schaltkreis in Abbildung 1 berechnet: $(x_1 + x_2) * (x_2 + x_3) + (x_2 + x_3) * (x_4 + x_5)$.

Dabei ist der Schaltkreis effizienter im Auswerten als der darunter beschriebene arithmetische Ausdruck, da das Zwischenergebnis $(x_2 + x_3)$ nur einmal, im arithmetischen Ausdruck aber zweimal ausgewertet werden muß. Arithmetische Schaltkreise für arithmetische

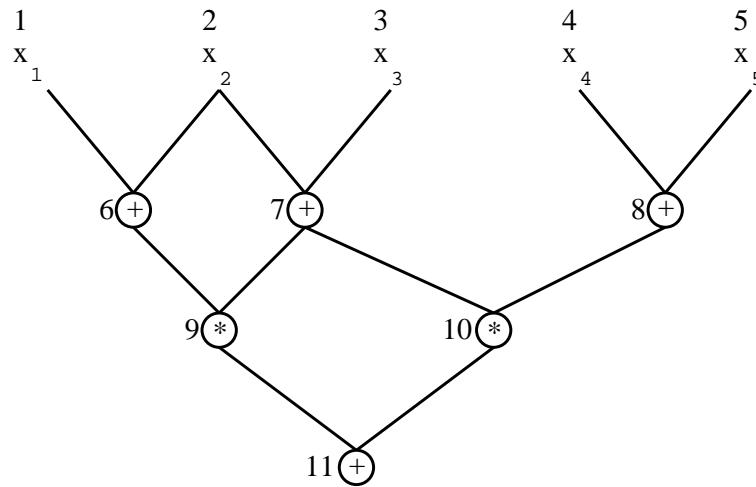


Abbildung 1: Arithmetischer Schaltkreis

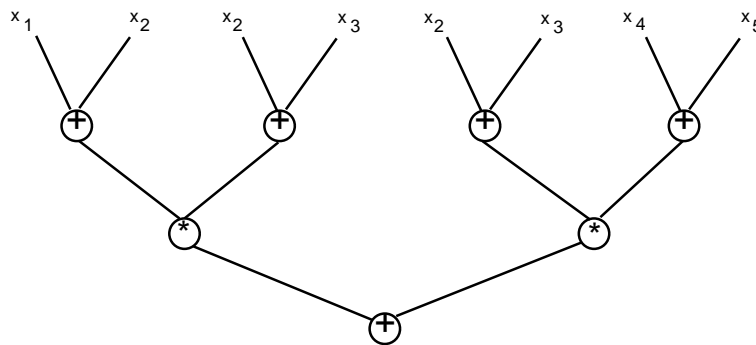


Abbildung 2: Arithmetischer Schaltkreis als Baum

Ausdrücke werden zu Bäumen, falls man die Inputs mehrfach hinschreiben darf. Für den Ausdruck aus Abbildung 1 ergibt das den Baum in Abbildung 2:

Arithmetische Schaltkreise lassen sich als Programme hinschreiben, indem man die Knoten topologisch sortiert, also so, dass Operanden einer Operation immer kleinere Nummern haben als die Operatoren selber.

In unserem Beispiel sieht das so aus:

- | | |
|-----------------------------------|---|
| 1) x_1 speichern in R_1 | 7) $[2] + [3]$ $R_7 := R_2 + R_3$ |
| 2) x_2 speichern in R_2 | 8) $[4] + [5]$ $R_8 := R_4 + R_5$ |
| 3) x_3 speichern in R_3 | 9) $[6] * [7]$ $R_9 := R_6 * R_7$ |
| 4) x_4 speichern in R_4 | 10) $[7] * [8]$ $R_{10} := R_7 * R_8$ |
| 5) x_5 speichern in R_5 | 11) $[9] + [10]$ $R_{11} := R_9 + R_{10}$ |
| 6) $[1] + [2]$ $R_6 := R_1 + R_2$ | |

In dieser Darstellungsform spricht man von *Straight Line Programmen* (Programme ohne Verzweigungen).

Ein Modell zur Auswertung von Straight Line Programmen:

- Wahlfreier Zugriff auf die Inputvariablen
- Speicherzellen können das Ergebnis einer Operation (also ein Zwischenergebnis) speichern
- In einem Schritt kann eine Operation ausgeführt und das Ergebnis gespeichert werden, falls die Operanden gespeichert sind, oder es kann eine beliebige Variable gelesen und abgespeichert werden. Speicherinhalte können jederzeit gelöscht werden, ohne dass dieses zur Laufzeit beiträgt. (Die Speicherzelle wird dann ‘unbenutzt’, bis sie wieder beschrieben wird).

Unter der *Laufzeit* eines Straight Line Programms verstehen wir die Anzahl der Programmschritte, und unter dem benötigten *Speicherplatz* die maximale Anzahl der gleichzeitig benutzten Zellen. Jedes Straight Line Programm der Länge n kann natürlich mit n Speicherzellen in n Schritten abgearbeitet werden, indem man die Operationsergebnisse in aufeinanderfolgenden Speicherzellen abspeichert. Es kann jedoch Speicherplatz gespart werden, wenn mehr als n Schritte durchgeführt werden. Siehe Abbildung 3.

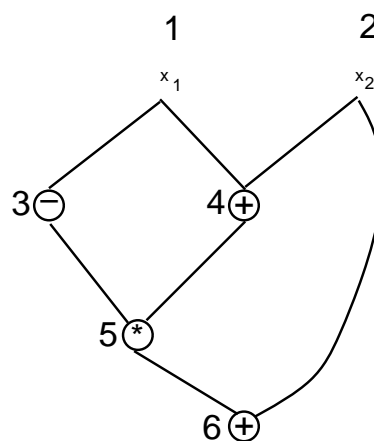


Abbildung 3: Berechnung von $(-x_1) \cdot (x_1 + x_2) + x_2$ in 6 Schritten

Behauptung: Um in 6 Schritten fertig zu werden, benötigt man genau 4 Zellen.

Beweis:

a) Es geht mit 4 Zellen, in 6 Schritten:

- | | |
|-----------------------------|-----------------------|
| 1) x_1 speichern in R_1 | 5) $R_1 := R_4 * R_3$ |
| 2) x_2 speichern in R_2 | R_3 löschen |
| 3) $R_3 := -R_1$ | 6) $R_3 := R_1 + R_2$ |
| 4) $R_4 := R_1 + R_2$ | |
| R_1 löschen | |

b) Man braucht 4 Zellen, wenn nur 6 Schritte erlaubt sind.

Wenn [4] berechnet wird, ist x_2 gespeichert und wird auch nicht mehr gelöscht, da es im letzten Schritt noch einmal gebraucht wird. Wenn [5] berechnet ist, sind somit [3], [4], [5] und x_2 gespeichert, d.h. es werden 4 Zellen benötigt.

□

Behauptung: In 7 Schritten kommt man mit 3 Zellen aus.

Beweis: Füge in der obigen Strategie hinter 3) „ R_3 löschen“ und vor 6) „ x_2 in R_2 speichern“ ein und benutze in den Schritten [4] und [5] das Register R_2 statt R_4 . □

Definition 2.3 Sei G ein gerichteter Graph ohne gerichtete Kreise (also ein directed acyclic Graph, DAG) und v_0 ein Knoten von G . Das pebble game ist ein Markierungsspiel, das die Knoten nach folgenden Regeln mit Marken (pebbles) belegen darf:

1. In einem Zug darf eine Marke auf einen Knoten v von G gelegt werden, falls alle Vorgänger von v in G schon Marken enthalten, insbesondere darf eine Marke jederzeit auf eine Quelle (das ist ein Knoten ohne Vorgänger) gelegt werden.
2. In einem Zug darf eine Marke entfernt werden.

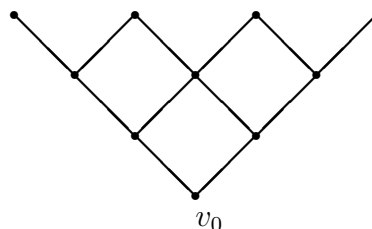
Das Spiel ist zu Ende, sobald v_0 eine Marke hat. Eine Folge von Regelanwendungen, die v_0 markiert, heißt dann Markierungsstrategie für v_0 in G . Ihre Kosten sind die maximale Zahl von Marken, die während der Strategie gleichzeitig auf dem Graphen liegen. Mit $K(G, v_0)$ bezeichnen wir das Minimum der Kosten aller möglichen Markierungsstrategien für v_0 in G .

Es besteht folgende Analogie zu dem obigen Modell der Straight Line Programme:

G	$\hat{=}$	arithmetischer Schaltkreis,
v_0	$\hat{=}$	Ergebnis,
Marke auf v	$\hat{=}$	Zwischenergebnis von v ist gespeichert.
Regel 1	$\hat{=}$	„Um eine Operation auszuwerten, müssen die Operanden gespeichert sein“.

Wir haben gesehen, dass Strategien mit minimalen Kosten manchmal mehr als n Schritte machen. Wir wollen uns jedoch nur um die Kosten kümmern.

Beispiel 2.4 Der Graph



heißt 4-Pyramide P_4 . Damit ist klar, was eine m -Pyramide P_m ist.

Behauptung: Es gilt: $K(P_m, v_0) = m + 1$ für $m \geq 2$.

Beweis:

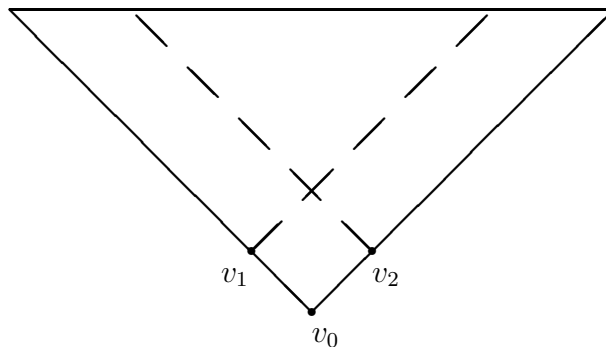
a) Wir zeigen: $K(P_m, v_0) \leq m + 1$ (obere Schranke).

Folgende rekursive Strategie benötigt $m + 1$ Marken.

$m = 2$:



$m > 2$: 2 Exemplare von P_{m-1}



1. Markiere v_1 (rekursiv).
2. Markiere v_2 (rekursiv).
3. Markiere v_0 .

Sei $S(m)$ die Zahl der Markierungen, die die obige Strategie benötigt; dann gilt:

$$\left. \begin{array}{l} S(2) = 3 \\ S(m) = S(m-1) + 1 \end{array} \right\} \Rightarrow S(m) = m + 1.$$

b) Wir zeigen: $K(P_m, v_0) \geq m + 1$ (untere Schranke).

Wir schauen uns in einer Markierungsstrategie den Zug A an, in dem gilt:

1. Vorher gab es einen Weg ohne Marken von einer Quelle zu v_0 (ein solcher Weg heißt *offener Weg*).
2. Ab Zug A gibt es nie wieder einen solchen Weg.

Bemerkung: In Zug A wird eine Marke auf eine Quelle q gelegt, wo vorher der offene Weg startete, denn sonst wäre auch vorher kein offener Weg da gewesen, denn die (markierten) Vorgänger des in A markierten Knotens v versperren alle Wege, die v versperren. Siehe Abbildung 4.

Auf jedem der $m - 1$ gestrichelten Wege muss vor Zug A ein Stein liegen. In Zug A wird ein weiterer auf q gelegt. Nach dem Zug A liegen also $\geq m$ Marken auf P_m . Falls im Zug $A + 1$ ein Stein entfernt wird, gab es schon vorher $\geq m + 1$ Marken, da auch hinterher alle Wege geschlossen sein müssen. Falls im Zug $A + 1$ ein Stein hingelegt wird, liegen nun $\geq m + 1$ Steine auf dem Graphen. \square

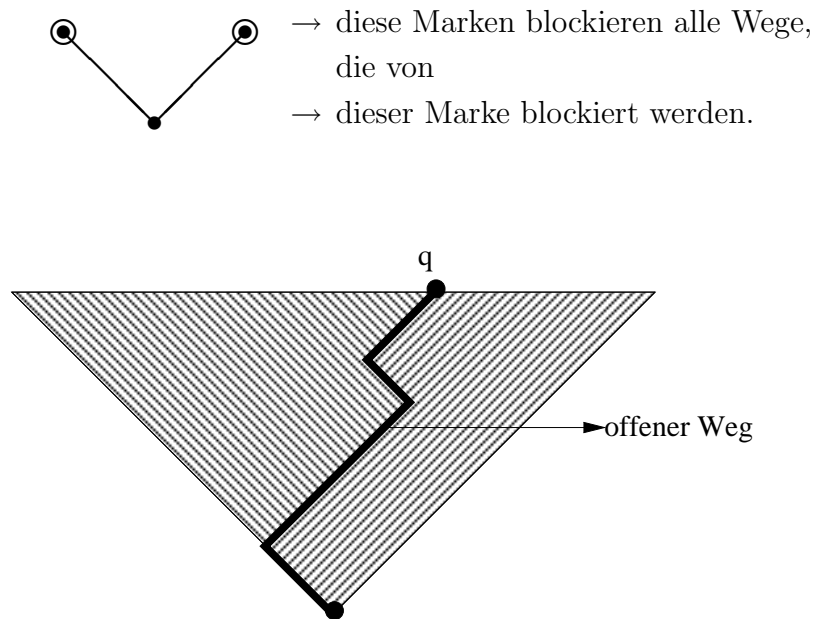


Abbildung 4: Zur Veranschaulichung der Markierungsstrategie

Jeder DAG mit n Knoten kann in höchstens 2^n Zügen markiert werden, da es nur 2^n verschiedene Muster gibt, in denen Marken auf den Knoten des Graphen liegen, und da in jeder Strategie, die mehr als 2^n Züge benötigt, sich ein Muster wiederholt, sodass sie deshalb verkürzt werden kann.

Zur Unterscheidung von $n = |V(G)|$ bezeichne im folgenden $N = |x|$ die Länge der Eingabe x an eine Turingmaschine. Stellen wir Graphen durch Adjazenzlisten dar, so hat ein Graph G mit n Knoten und m Kanten eine Adjazenzliste, die binär codiert eine Länge von $N = O(m \log n + n \log n)$ besitzt. Diese Darstellung von G bezeichnen wir mit $c(G)$.

Lemma 2.5 *Es gibt eine $O(N^2)$ - bandbeschränkte deterministische Turingmaschine M , die gestartet mit dem Tupel $(c(G), \text{bin}(i), \text{bin}(v))$ nacheinander (immer auf dem selben Platz) die Züge einer Markierungsstrategie für v in G generiert, die höchstens i Marken benutzt, falls eine solche Strategie existiert.*

Beweis: Seien $c(G), \text{bin}(i), \text{bin}(v)$ wie oben. Nach j Zügen liegen Marken auf v_1, \dots, v_s , $s \leq i$. A sei ein möglicher nächster Zug. Dann gibt es eine $O(N)$ - platzbeschränkte nichtdeterministische Turingmaschine M' , die bei Eingabe $c(G), \text{bin}(v), \text{bin}(v_1), \dots, \text{bin}(v_s), \text{bin}(A), \text{bin}(2^n - j), \text{bin}(i)$ entscheidet, ob es eine Strategie der Länge $2^n - j$ für v gibt, die nur i Marken benötigt und bei der Markierung v_1, \dots, v_s anschließend den Zug A ausführt.

Arbeitsweise von M' :

Band 1 enthält die Eingabe,

Band 2 enthält die zur Zeit markierten Knoten (zu Beginn v_1, \dots, v_s),

Band 3 enthält die noch zur Verfügung stehende Zahl von Schritten (zu Beginn $2^n - j$),

Band 4 enthält den jeweils geratenen Knoten.

Algorithmus für M' :

Führe A aus.

Solange der Inhalt von Band 3 größer als 0 ist, tue folgendes:

- Lösche Band 4.
- Rate Knoten \tilde{v} und schreibe ihn auf Band 4
- Falls \tilde{v} nicht auf Band 2 steht:
 - a) Falls alle Vorgänger von \tilde{v} auf Band 2 stehen (d.h. sie sind markiert) und falls auf Band 2 höchstens $i - 1$ Knoten stehen:
 - * Füge \tilde{v} zu Band 2 hinzu.
 - * Subtrahiere eine 1 von Band 3.
 - b) Falls es einen Vorgänger von \tilde{v} gibt, der nicht auf Band 2 steht, so akzeptiere nicht, stoppe.
- Falls \tilde{v} auf Band 2 steht:
 - Streiche \tilde{v} von Band 2.
 - Subtrahiere eine 1 von Band 3.

Akzeptiere sobald v markiert ist.

M' rät also einfach nichtdeterministisch Schritt für Schritt einen neuen Zug. M' kann nach [Komplexitätstheorie I, Satz 5.8] (Satz von Savitch) durch eine deterministische Turingmaschine M'' simuliert werden, die $O(N^2)$ - platzbeschränkt ist. Die im Lemma geforderte Turingmaschine M erzeugt nun einfach mit Hilfe von M'' Schritt für Schritt alle Züge der Strategie der Länge 2^n mit i Marken, falls eine solche Strategie existiert. M ist wie M'' $O(N^2)$ - bandbeschränkt. Die Länge 2^n ist keine Einschränkung, da sich in längeren Strategien Spielsituationen wiederholen, diese Strategien also verkürzt werden können. \square

Definition 2.6 Eine Zerlegung von $G = (V, E)$ in zwei Graphen $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ erfüllt:

- a) $V_1 \cap V_2 = \emptyset$, $V_1 \cup V_2 = V$,
- b) $E_1 = E|_{V_1}$, $E_2 = E|_{V_2}$,
- c) es gibt in G keine von V_2 nach V_1 gerichtete Kante.

Behauptung: Sei G ein DAG mit n Knoten, m Kanten und Ingrad k . Dann gibt es eine Zerlegung G_1, G_2 von G mit $\frac{m}{2} - k \leq |E_1| < \frac{m}{2}$.

Beweis: Erzeuge G_1 , indem solange Knoten von G in topologischer Reihenfolge hinzugefügt werden, bis zum ersten mal $|E_1| \geq \frac{m}{2} - k$ wird. Da in einem Schritt höchstens k Kanten zu E_1 hinzugefügt werden, ist $|E_1|$ auch $< \frac{m}{2}$. \square

Entscheidend für den Beweis von Satz 2.1 ist nun die folgende Aussage.

Satz 2.7 Sei G ein DAG mit n Knoten und Ingrad k , v sei ein Knoten von G . Dann gilt:

$$K(G, v) = O\left(k \cdot \log k \cdot \frac{n}{\log n}\right).$$

Beweis: $P_k(m) := \max\{K(G, v) \mid G \text{ hat } m \text{ Kanten und Ingrad } k, v \in V\}$, d.h. $P_k(m)$ ist die kleinste Anzahl von Marken, die man braucht, um in jedem DAG mit Ingrad k und m Kanten einen beliebigen Knoten zu markieren.

Wir zeigen folgende Beziehung: $P_k(m) = O(\log k \cdot \frac{m}{\log m})$ (Lemma 2.8).

Da jeder zusammenhängende Graph mit n Knoten und Ingrad k höchstens $k \cdot n$ und mindestens $n - 1$ viele Kanten hat, folgt daraus Satz 2.7: jede Zusammenhangskomponente läßt sich ‘pebbeln’ unabhängig von den anderen. \square

Lemma 2.8

$$P_k(m) \leq \max \left\{ P_k\left(\frac{m}{2} + k\right) + \frac{2m}{\log m}, P_k\left(\frac{m}{2}\right) + P_k\left(\frac{m}{2} + k - \frac{2m}{\log m}\right) + k \right\} .$$

Hieraus folgt durch einen technisch etwas aufwendigen Induktionsbeweis, den wir hier nicht durchführen wollen, die Beziehung $P_k(m) = O(\log k \cdot \frac{m}{\log m})$.

Beweis: G habe m Kanten und Ingrad k , v soll markiert werden. Zerlege G in G_1 und G_2 mit $\frac{m}{2} - k \leq |E_1| < \frac{m}{2}$ (siehe obige Behauptung). Bezeichne A die Menge der Kanten, die zwischen G_1 und G_2 verlaufen. Es gelten nun folgende Abschätzungen:

$$|E_1| < \frac{m}{2}, \quad |E_2| = m - |E_1| - |A| \leq \frac{m}{2} + k - |A| \leq \frac{m}{2} + k .$$

Fall 1: $v \in V_1 \Rightarrow$ es sind höchstens $P_k(|E_1|) \leq P_k(\frac{m}{2})$ Marken nötig.

Fall 2: $v \in V_2$. Wir betrachten zwei mögliche Strategien (siehe Abbildung 5):

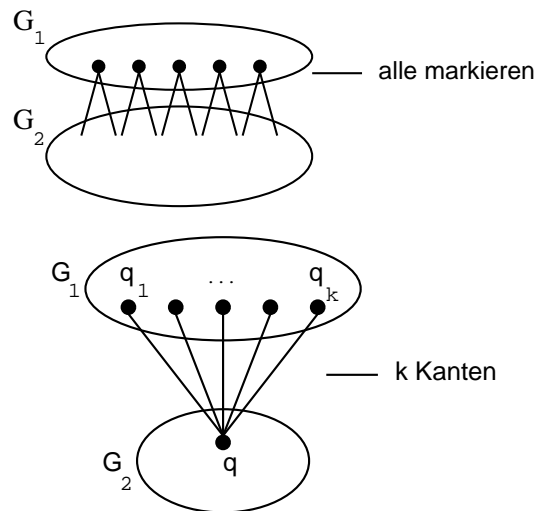


Abbildung 5: zum Beweis von Lemma 2.8

Strategie a) Markiere erst alle Quellen von A : da dies Knoten von G_1 sind, welcher maximalen Ingrad k hat, braucht dies für jede einzelne höchstens $P_k(|E_1|)$ Marken; eine bleibt jeweils liegen, während die nächste bearbeitet wird: macht insgesamt $P_k(|E_1|) + |A|$ Marken.

Lasse die $|A|$ Marken liegen und markiere nun, mit $P_k(|E_2|)$ weiteren Marken, G_2 : das ist möglich, da jede Quelle (von G_2) in der Tat alle Vorgänger (in G) bereits markiert hat.

$$P_k(m) \leq \max \left\{ P_k(|E_1|) + |A|, |A| + P_k(|E_2|) \right\} \leq P_k\left(\frac{m}{2} + k\right) + |A|$$

Strategie b) Bearbeite G_2 : Wann immer eine seiner Quellen q markiert werden soll, markiere alle seine (höchstens k) Vorgänger in G_1 . Letzteres geht mit $P_k(|E_1|) + k$ Marken, die Strategie für G_2 braucht maximal weitere $P_k(|E_2|)$ viele.

$$P_k(m) \leq P_k(|E_1|) + P_k(|E_2|) + k \leq P_k\left(\frac{m}{2}\right) + P_k\left(\frac{m}{2} + k - |A|\right) + k$$

Man sieht, dass Strategie a) für kleinere $|A|$ besser ist. Wir wählen Strategie a) bei $|A| < \frac{2m}{\log m}$, sonst Strategie b). Damit folgt Lemma 2.8 \square

Wir können nun das wichtigste Ergebnis dieses Abschnitts für die weiteren Betrachtungen formulieren:

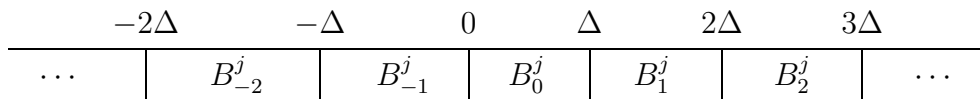
Korollar 2.9 *Es gibt eine deterministische $O(N^2)$ -bandbeschränkte Turingmaschine, die bei Eingabe $(c(G), v)$ nacheinander alle Züge einer Markierungsstrategie für v in G erzeugt und nur $O\left(k \cdot \log k \cdot \frac{n}{\log n}\right)$ Marken benutzt, falls G n Knoten und Ingrad k hat.*

Der Beweis folgt direkt aus Lemma 2.5 und Satz 2.7. \square

Nach all diesen Vorbereitungen können wir nun Satz 2.1 beweisen.

Beweis: (von Satz 2.1) Sei $L \in \text{DTIME}(t(n))$, M eine $c \cdot t(n)$ -zeitbeschränkte deterministische k -Band - Turingmaschine für L , x sei ein Input, $|x| = n$, $\Delta = \Delta(n) \in \mathbb{N}$ fest und $T := c \cdot t(n)$. Es folgt nun die Beschreibung eines Δ -Berechnungsgraphen von M gestartet mit x .

- Wir zerlegen jedes Band $j \in \{1, \dots, k\}$ in Blöcke B_i^j der Länge Δ , $i \in \mathbb{Z}$.

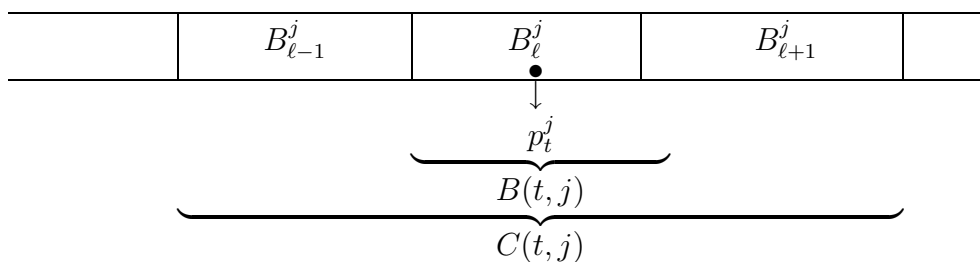


Band j

- Wir zerlegen die Rechnung in Zeitintervalle I_1, \dots, I_R . p_t^j sei die Kopfposition auf Band j zu Beginn von I_t .

$B(t, j) \hat{=} \text{Block auf Band } j, \text{ in dem } p_t^j \text{ liegt.}$

$C(t, j) \hat{=} B(t, j) \text{ zusammen mit den beiden Nachbarblöcken.}$

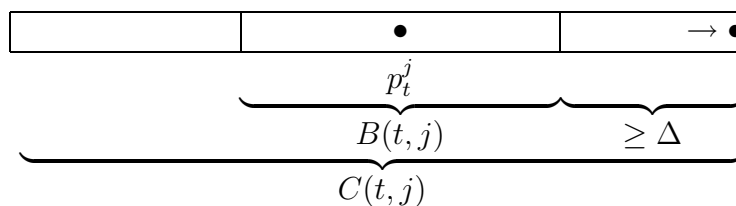


$B(t, j), C(t, j)$ sind definiert, sobald p_t^j definiert ist. Wir definieren den Beginn von I_t , sowie $p_t = (p_t^1, \dots, p_t^k)$, induktiv nach t .

$t = 1$: I_1 beginnt zum Zeitpunkt 1, $p_1^1 = \dots = p_1^k = 0$.

$t > 1$: I_t beginnt, sobald das erste Mal seit Beginn von I_{t-1} ein Kopf j den 3-er Block $C(t-1, j)$ verlassen hat. Die Kopfpositionen zu diesem Zeitpunkt heißen p_t^1, \dots, p_t^k .

Wir nennen p_1, \dots, p_R das Δ -Stenogramm (von M gestartet mit x). Jedes Intervall I_t (eventuell außer des letzten) hat eine Länge von $\geq \Delta$, da mindestens ein Kopf j von einer Zelle in $B(t, j)$ aus bis an den Rand von $C(t, j)$ laufen muss.



Also gibt es höchstens $R \leq \lceil \frac{T}{\Delta} \rceil$ viele Zeitintervalle. Wir können nun den Δ -Berechnungsgraphen definieren, auf dem wir später Markierungsstrategien laufen lassen werden. Die drei in $C(t, j)$ enthaltenen Blöcke bezeichnen wir als α -Blöcke von $C(t, j)$, für $\alpha \in \{-1, 0, 1\}$. Der Δ -Berechnungsgraph G von M gestartet mit x wird folgendermaßen definiert:

Knoten: I_1, \dots, I_R .

Kanten:

1. Für $t \in \{2, \dots, R\}$ hat I_t für jedes $(j, \alpha) \in \{1, \dots, k\} \times \{-1, 0, 1\}$ einen (j, α) -Vorgänger $I_{t'}$ mit:

$$t' = \begin{cases} \max\{t'' \mid t'' < t, \alpha\text{-Block von } C(t, j) \text{ ist in } C(t'', j) \text{ enthalten}\}, & \text{falls mindestens ein solches } t'' \text{ existiert} \\ 1, & \text{sonst} \end{cases}$$

2. der direkte Vorgänger von I_t ist I_{t-1} .

Es gilt nun:

1. Ingrad von $G \leq 3k + 1$.

2. Seien t, j fest, $I_{t(\alpha)}$, $\alpha \in \{-1, 0, 1\}$, die drei (j, α) -Vorgänger von I_t . Die Inschrift von $C(t, j)$ zu Beginn von I_t erhalten wir aus:

der Inschrift von $C(t_{(-1)}, j)$, am Ende von $I_{t_{(-1)}}$
 der Inschrift von $C(t_{(0)}, j)$, am Ende von $I_{t_{(0)}}$
 der Inschrift von $C(t_{(1)}, j)$, am Ende von $I_{t_{(1)}}$ } siehe hierzu auch Abbildung 6

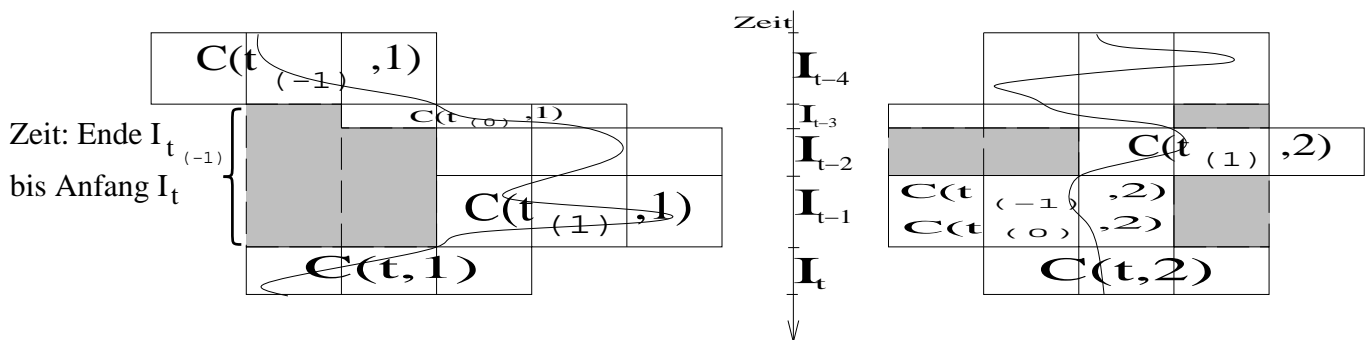


Abbildung 6: Zur Definition des Δ -Berechnungsgraphen

In den schraffierten Bereichen wird nach Definition von G die Inschrift des zugehörigen Blocks nicht verändert.

Bemerkung: Falls z.B. $t_{(1)} = 1$ ist, so steht in $C(t_{(-1)}, j)$ zu Beginn von I_1 der dementsprechende Block von $\dots B \dots BxB \dots B \dots$. Wir können also aus den Inschriften von $C(t_{(\alpha)}, j)$ zu Ende von $I_{t_{(\alpha)}}$ die Inschrift von $C(t, j)$ zu Beginn von I_t ablesen.

Wenn wir uns zu jedem $I_{t'}$ ausserdem den Zustand q_t von M am Ende von $I_{t'}$ merken, kennen wir über den direkten Vorgänger auch den Zustand zu Beginn von I_t . Die Kopfpositionen zu Beginn von I_t stehen im Δ -Stenogramm.

Lemma 2.10 *Seien G ein Graph mit Knoten I_1, \dots, I_R und Ingrad $3k+1$, $p = p_1, \dots, p_R$ eine Folge von k -Tupeln von Zahlen aus $\{-T, T\}$, x ein Input mit $|x| = n$ und $\Delta \in \mathbb{N}$, $\Delta \leq T$. Dann gibt es eine deterministische Turingmaschine \tilde{M} , die bei Eingabe $(c(G), p_1, \dots, p_R, \Delta, x)$ entscheidet, ob G der Δ -Berechnungsgraph und p_1, \dots, p_R das Δ -Stenogramm von M gestartet mit x ist, und, wenn ja, $f_M(x)$ berechnet. Die Turingmaschine \tilde{M} ist $O((R \log R)^2 + \Delta \cdot \frac{R}{\log R} + R \log T)$ - bandbeschränkt.*

Beweis: \tilde{M} arbeitet wie folgt:

Auf Band 1 wird die optimale Markierungsstrategie für G erzeugt, die nacheinander alle Züge generiert. Band 2 enthält $c(G), p_1, \dots, p_R, x, \Delta$. Falls die Markierungsstrategie auf Band 1 Marken auf I_{t_1}, \dots, I_{t_s} liegen hat, so soll auf Band 3 $C_e(t_1), \dots, C_e(t_s)$ gespeichert sein. Dabei ist $C_e(t) := (\text{Inhalte von } C(t, 1), \dots, C(t, k))$ am Ende von I_t und der Zustand von M am Ende von I_t .

Wir können diese Eigenschaft wie folgt erhalten: Seien A_1, \dots, A_ℓ die Züge der Markierungsstrategie auf Band 1.

Vor A_1 : Es liegt keine Marke auf G , d.h. Band 3 ist leer.

Seien nun $\ell \geq 0$ Züge A_1, \dots, A_ℓ ausgeführt.

Zug $A_{\ell+1}$:

a) *Entferne Marke von I_t :*

Streiche $C_e(t)$ von Band 3 und schiebe die Inschrift zusammen.

b) *Lege Marke auf I_t :*

Seien I_{t_1}, \dots, I_{t_q} die Vorgänger von I_t . Sie alle enthalten Marken. Auf Band 3 steht also unter anderem $C_e(t_1), \dots, C_e(t_q)$. $C_a(t)$ bezeichne (Inhalte von $C(t, 1), \dots, C(t, k)$) am Anfang von I_t und der Zustand von M am Anfang von I_t .

- Teste, ob $C_a(t)$ aus $C_e(t_1), \dots, C_e(t_q)$ erzeugt werden kann.

Falls nein: verwerfe, denn dann ist G nicht der Δ -Berechnungsgraph von M gestartet mit x .

Falls ja:

- Simuliere M , ausgehend vom Zustand und Bandinschrift aus $C_a(t)$, sowie Kopfposition aus p_t (von der Eingabe), solange, bis das erste mal ein Kopf j $C(t, j)$ verläßt. (Dann ist I_t vorbei.)
- Teste, ob die erhaltenen Kopfpositionen mit p_{t+1} übereinstimmen. Falls nein: verwerfe, denn dann ist p_1, \dots, p_R nicht das Δ -Stenogramm von M gestartet mit x . Falls ja:
- Füge das so berechnete $C_e(t)$ zu Band 3 hinzu.

Falls dieser Algorithmus während der gesamten Markierungsstrategie nie verwirft, hat er schließlich $C_e(R)$ berechnet, also insbesondere $f_M(x) \in \{0, 1\}$. Dieses steht dann nämlich unter der Kopfposition von Band 1 in $C_e(R)$.

Die Korrektheit ist durch die Kommentare während der Beschreibung gezeigt.

Bandverbrauch:

Band 1: $O((R \log R)^2)$ wegen Korollar 2.9
(G hat R Knoten und $\leq (3k + 1)R = O(R)$ Kanten).

Band 2: $O(R \log T) + n + \log \Delta + O(R \log R) = O(R \log T)$.

Band 3: Da auf Spur 1 eine optimale Markierungsstrategie für G erzeugt wird, werden nach Korollar 2.9 höchstens $O(\frac{R}{\log R})$ Marken benutzt. Für jede Marke wird auf Band 3 $C_e(t)$ notiert, also Platz $O(\Delta)$ benötigt. Die Simulation benötigt jeweils $O(\Delta)$ Platz. Insgesamt benötigt Band 3 also Platz $O(\Delta \cdot \frac{R}{\log R})$.

Gesamtplatz: $O((R \log R)^2 + R \log T + \Delta \cdot \frac{R}{\log R})$. □

Beschreibung des Simulationsalgorithmus:

Vorbemerkung: Wir wollen Δ so wählen, dass der Gesamtplatzbedarf der obigen Turingmaschine

$$O(R^2(\log R)^2 + R \log T + \Delta \cdot \frac{R}{\log R})$$

minimal wird. Dabei wissen wir: $T = \Delta \cdot R$.

Beobachtung: $R^2(\log R)^2$ wächst mit R , wogegen $\Delta \cdot \frac{R}{\log R} = \frac{T}{\log R}$ mit R fällt. Wir wählen Δ so, daß die beiden Terme gleich groß werden, also daß $R^2(\log R)^2 = \frac{T}{\log R}$ gilt. Da T polynomiell in R ist, ist $\log R = O(\log T)$.

Also: Bis auf einen konstanten Faktor gilt: $R^2(\log T)^3 = T$ also $R = \left\lfloor \frac{T^{1/2}}{(\log T)^{3/2}} \right\rfloor$.

Dann ist $\Delta = \frac{T}{R} = T^{1/2} \cdot (\log T)^{3/2}$, und für den Bandverbrauch ergibt sich:

$$\begin{aligned} & O(R^2(\log R)^2 + R \log T + \Delta R / \log R) \\ &= O\left(\frac{T}{(\log T)^3} \cdot (\log T)^2 + \frac{T^{1/2}}{(\log T)^{3/2}} \cdot \log T + \frac{T}{\log T}\right) \\ &= O\left(\frac{T}{\log T}\right). \end{aligned}$$

Folgende deterministische Turingmaschine M' simuliert M mit Eingabe x . (Wir nehmen vorerst an: $t(|x|)$ ist aus x auf Band $O(\frac{t(|x|)}{\log t(|x|)})$ berechenbar.)

1. Berechne $T = c \cdot t(|x|)$.
2. Berechne $\Delta = \lceil T^{1/2} \cdot (\log T)^{3/2} \rceil$.

3. Berechne $R = \left\lceil \frac{T^{1/2}}{(\log T)^{3/2}} \right\rceil = \left\lceil \frac{T}{\Delta} \right\rceil$.
4. Erzeuge auf Band 2 hintereinander (und immer auf demselben Platz) alle Worte vom Typ $(c(G), p_1, \dots, p_R, \Delta, x)$ wie in Lemma 2.10, wobei x die Eingabe, Δ, R wie in 2) und 3), $p_t \in \{-T, \dots, T\}^k$, $t = 1, \dots, R$, und $c(G)$ die Codierung eines gerichteten Graphen ohne Kreise mit Ingrad $3k + 1$, der alle Kanten $(t, t - 1)$ enthält, seien. Wende darauf jedesmal die Turingmaschine aus Lemma 2.10 an. Falls diese Turingmaschine für eine auf Band 2 erzeugte Eingabe entscheidet, daß G der Δ -Berechnungsgraph und p_1, \dots, p_R das Δ -Stenogramm von M gestartet mit x ist, erzeugt die Simulation aus dem Lemma $f_M(x)$ und die Rechnung ist fertig.

Bandverbrauch:

- 1) $O\left(\frac{t(|x|)}{\log t(|x|)}\right)$ wegen der Annahme.
- 2) und 3) $O\left(\frac{t(|x|)}{\log t(|x|)}\right)$.
- 4) $O\left(\frac{t(|x|)}{\log t(|x|)}\right)$ nach Lemma 2.10 und wegen der Wahl von Δ .

Insgesamt: $O\left(\frac{t(|x|)}{\log t(|x|)}\right)$.

Ein Trick, um die Berechnung von $T = c \cdot t(|x|)$ zu umgehen: Arbeite 2) – 4) hintereinander ab, jeweils unter der Annahme $T = 2, 4, 8, \dots, 2^i$, bis irgendwann $f_M(|x|)$ berechnet ist. $f_M(|x|)$ wird berechnet, falls $T = 2^{i_0}$ benutzt wurde mit $i_0 = \lceil \log(c \cdot T(|x|)) \rceil$.

Da jedes 2^i auf Platz i berechnet wird und das größte i ($= i_0$) $O(\log(c \cdot T(|x|)))$ groß ist, benötigt auch 1) nur $O(\log T(|x|)) = O\left(\frac{T(|x|)}{\log T(|x|)}\right)$, egal wieviel Platzaufwand zur Berechnung von $T(|x|)$ notwendig wäre. □ Satz 2.1

3 Schaltkreise, uniforme Schaltkreisfamilien und Parallele Rechenmodelle

Boole'sche Schaltkreise modellieren die Hardware von Rechnern. Sie berechnen nicht Funktionen $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ sondern nur endliche Funktionen $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ für feste $n, m \in \mathbb{N}$. Sie sind ein sogenanntes *nichtuniformes Rechenmodell*. In Rechnern enthält die ALU Schaltkreise (allerdings im Gegensatz zu unseren, mit Rückkopplung \rightarrow sog. Schaltwerke) für logische und arithmetische Operationen (z. B. Addition, Subtraktion, Multiplikation auf je zwei 32-Bit Worten in Zweierkomplementdarstellung).

Wir werden sehen, dass sog. uniforme Schaltkreisfamilien ein Rechenmodell definieren, in dem wir sequentielle und parallele Laufzeit messen können. Wir werden die sequentielle Laufzeit zu der Laufzeit von TM's in Relation setzen. Zudem werden wir eine enge Beziehung zwischen paralleler Laufzeit und Speicherplatz herleiten.

Definition 3.1 B_n^m bezeichnet die Menge aller Booleschen Funktionen mit n Inputs und m Outputs, d. h. $B_n^m = \{f : \{0, 1\}^n \rightarrow \{0, 1\}^m\}$. Für B_n^1 schreiben wir B_n . Eine Basis ist eine Menge $\Omega \subseteq B_2 \cup B_1$.

Bemerkung: $|B_n^m| = 2^{m \cdot 2^n}$.

Definition 3.2 Sei Ω eine Basis. Ein Ω -Schaltkreis S mit n Eingängen und m Ausgängen ist ein gerichteter azyklischer Graph mit n Quellen Q_1, \dots, Q_n , den Eingaben, speziellen Quellen E, N , und m ausgezeichneten Knoten S_1, \dots, S_m , den Ausgaben. Der Graph hat Ingrad 2 und jedem Knoten mit Ingrad $i, i \in \{1, 2\}$, ist ein Baustein $b \in B_i$ zugeordnet. Bei Input $\bar{x} = (x_1, \dots, x_n)$ wird an jedem Knoten v eine Funktion g_v berechnet. An Q_i wird $g(\bar{x}) = x_i$, an E wird $g_E(\bar{x}) \equiv 1$, an N wird $g_N(\bar{x}) \equiv 0$ berechnet, an Knoten v mit Baustein $b \in \Omega$ und Vorgängern v' und v'' (bzw. nur v' , falls $b \in B_1$) wird $g_v(\bar{x}) = b(g_{v'}(\bar{x}), g_{v''}(\bar{x}))$ (bzw. $g_v(\bar{x}) = b(g_{v'}(\bar{x}))$) berechnet. Die von S berechnete Funktion $f_S : \{0, 1\}^n \rightarrow \{0, 1\}^m$ ist $f_S(\bar{x}) = (g_{S_1}(\bar{x}), \dots, g_{S_m}(\bar{x}))$.

Einige Bausteine: NOT(\neg) $\in B_1$ und AND(\wedge), OR(\vee), XOR(\oplus) $\in B_2$

Inputs	0	1		0	0	0	1	1	0	1	1
NOT, \neg	1	0	AND, \wedge	0	0	0	0	0	0	1	1
			OR, \vee	0	1	1	1	1	1	1	1
			XOR, \oplus	0	1	1	1	1	0	0	0

Definition 3.3 S sei ein Schaltkreis.

$C(S) :=$ Anzahl der Bausteine aus B_2 in S ist die Schaltkreisgröße.

$D(S) :=$ Länge eines längsten gerichteten Weges von einer Quelle zu einer Senke (ohne Bausteine aus B_1) ist die Schaltkreistiefe.

Sei $f \in B_n^m$ durch einen Ω -Schaltkreis berechenbar, dann ist

$$C_\Omega(f) := \min\{C(S), S \text{ ist } \Omega\text{-Schaltkreis für } f\}$$

$$D_\Omega(f) := \min\{D(S), S \text{ ist } \Omega\text{-Schaltkreis für } f\}.$$

Bemerkungen:

- Intuitiv kann jeder Schaltkreis S (etwa von einer RAM) in Zeit $O(C(S))$ ausgewertet werden. (Berechne die Werte der Bausteine nacheinander in topologischer Reihenfolge.) $\longrightarrow C(S) \approx$ sequentielle Laufzeit.
- Intuitiv kann jeder Schaltkreis S mit $C(S)$ Prozessoren in Zeit $O(D(S))$ parallel ausgewertet werden. (Jeder Prozessor wertet einen Baustein aus, und zwar, sobald seine Vorgänger ausgewertet sind.) $\longrightarrow D(S) \approx$ parallele Laufzeit (mit $C(S)$ Prozessoren).
- Jeder Schaltkreis S kann mit $D(S)$ Speicherzellen (+ Eingabe) ausgewertet werden (benutze Markierungsstrategie, $T(d)$ = maximale Zahl von Marken, die für Schaltkreise der Tiefe d notwendig ist. $T(1) = 1, d > 1 : T(d) \leq T(d-1) + 1 \implies T(d) = d \longrightarrow D(S) \approx$ Speicherbedarf).

Definition 3.4 Eine Basis Ω heißt vollständig, falls es für jedes $f \in B_n$ einen Ω -Schaltkreis gibt.

Bemerkung: $\{\wedge, \vee, \neg\}, \{\wedge, \neg\}, \{\vee, \neg\}, \{\oplus, \wedge\}$ sind vollständige Basen.

Beweis: $\Omega = \{\wedge, \vee, \neg\}$. Seien $\bar{x} = (x_1, \dots, x_n)$ Bool'sche Variablen. Ein Minterm $m_{\bar{a}}$ für $\bar{a} \in \{0, 1\}^n$ ist definiert als $m_{\bar{a}} = x_1^{a(1)} \wedge \dots \wedge x_n^{a(n)}$. Dabei ist $x^0 := \neg x, x^1 := x$. Sei $f \in B_n$. Dann läßt sich f durch folgenden Schaltkreis beschreiben: $f(x) = \bigvee_{a \in f^{-1}(1)} m_{\bar{a}}(x)$ (Disjunktive Normalform)(DNF).

Beweis:

(i) $f(\bar{x}) = 1 \implies \bar{x} \in f^{-1}(1)$ und $m_{\bar{x}}(\bar{x}) = 1 \implies \bigvee_{a \in f^{-1}(1)} m_{\bar{a}}(x) = 1$

(ii) $\bigvee_{a \in f^{-1}(1)} m_{\bar{a}}(x) = 1 \implies \exists a \in f^{-1}(1) : m_{\bar{a}}(x) = 1 \implies x \in f^{-1}(1) \implies f(x) = 1.$

– $\{\wedge, \neg\}$: stelle \vee durch $\{\wedge, \neg\}$ dar: $x \vee y = \neg(\neg x \wedge \neg y)$

– $\{\vee, \neg\}$: stelle \wedge durch $\{\vee, \neg\}$ dar: $x \wedge y = \neg(\neg x \vee \neg y)$

– $\{\oplus, \wedge\}$: stelle \vee, \neg durch $\{\oplus, \wedge\}$ dar: $\neg x = x \oplus 1, x \vee y = \neg(\neg x \wedge \neg y)$

□

Exkurs: Effiziente Schaltkreise

DNF ist eine einfache Darstellung, aber oft ineffizient. Beispiel Parity:

$$\text{Parity}(x) = 1 \iff \sum_{i=1}^n x_i \text{ ist ungerade.}$$

Man berechnet $\text{Parity}(x)$ etwa mit \oplus -Bausteinen:

$$\begin{aligned} \text{Parity}(x) = x_1 \oplus \dots \oplus x_n, \quad & \text{also } C_{\{\oplus\}}(\text{Parity}) \leq n - 1, \text{ insbesondere} \\ & C(\text{Parity}) = O(n) \text{ über der Standardbasis } \{\wedge, \vee, \neg\}. \end{aligned}$$

Aber die Disjunktive Normalform für Parity: $\text{Parity}(x) = \bigvee_{a \in \text{Parity}^{-1}(1)} m_{\bar{a}}(x)$ hat wegen $|\text{Parity}^{-1}(1)| = 2^{n-1}$ exponentielle Länge!

Polynomdarstellung

Zu jeder Boole'schen Variablen x_i gibt es die zwei *Literale* x_i (die Variable) und \bar{x}_i (ihre Negation). Ein *Monom* $m = y_1 \wedge \dots \wedge y_l$ ist die Konjunktion von Literalen y_1, \dots, y_l .

Ein *Polynom* ist die Disjunktion von Monomen $\bigvee_{i=1}^l m_i$, m_i Monome.

Jede Boole'sche Funktion $f \in B_n$ hat eine Polynomdarstellung, nämlich die *DNF*. Als ein *Minimalpolynom* von f ($MP(f)$) bezeichnet man ein Polynom für f mit kürzester Länge. Ein solches $MP(f)$ kann für jedes f aus seiner *DNF* berechnet werden. Es ist jedoch $MP(\text{Parity}) = DNF(\text{Parity})$ und damit z.B. das Minimalpolynom für Parity wiederum exponentiell lang. Polynome lassen sich unmittelbar als zweistufige (d.h. Tiefe=2) Schaltkreise realisieren. Somit kann man also direkt aus der *DNF* oder der Minimalpolynomdarstellung einer Funktion f einen Schaltkreis konstruieren.

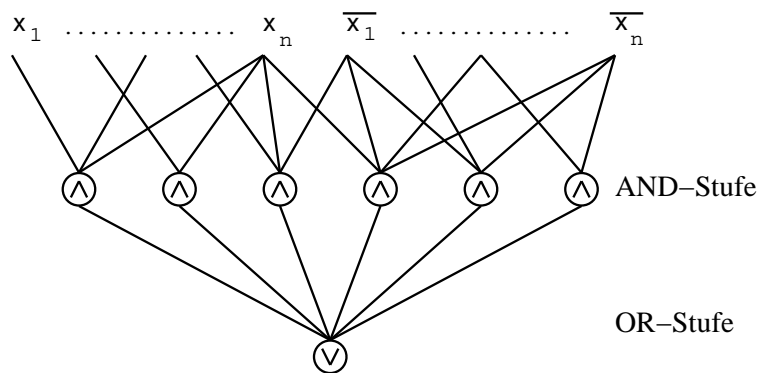


Abbildung 7: Aus einem Minimalpolynom konstruierter Schaltkreis

Die AND-Stufe enthält für jedes Monom y_i der (Minimal-) Polynomdarstellung genau einen AND-Baustein. Die so konstruierten zweistufigen Schaltkreise lassen sich auch sehr einfach realisieren (als PLA's, Programmable Logical Arrays), sind jedoch wie oben gesehen häufig sehr ineffizient.

Der Entwurf effizienter Schaltkreise benötigt – ebenso wie der Entwurf effizienter Algorithmen – viel Intuition, Erfahrung und Verständnis für das zu lösende Problem.

3.1 Asymptotische Resultate über Schaltkreisgrößen

Wir wollen hier untersuchen, wie klein man einen Schaltkreis für eine beliebige Funktion $f \in B_n$ machen kann, und wir wollen zeigen, dass es „schwere“ Funktionen gibt, für die ein Schaltkreis nicht klein sein kann.

Satz 3.5

- a) Für jede Funktion $f \in B_n$ gilt: $C(f) \leq \frac{2^n}{n} + o(\frac{2^n}{n})$ (mit der Standardbasis).
- b) Es gibt Funktionen $f \in B_n$, für die gilt: $C(f) \geq \frac{2^n}{n} - o(\frac{2^n}{n})$ (fast alle Funktionen sind so „schwer“).

Beweis:

- a) Wir zeigen hier eine etwas einfachere Konstruktion, die $2 \frac{2^n}{n} + o(\frac{2^n}{n})$ Bausteine benötigt. Sei $f \in B_n$; $f_0 := f|_{x_n=0}$ und $f_1 := f|_{x_n=1}$, also $f_0, f_1 \in B_{n-1}$. Es gilt:

$$f(x) = (x_n \wedge f_1(x)) \vee (\neg x_n \wedge f_0(x)) \quad . \quad (1)$$

Diese Konstruktion liefert uns ein Schema zur Berechnung von f . Die Größe des so konstruierten Schaltkreises ergibt sich durch die Rekursion mit $T(2) = 1; T(n) = 2T(n - 1) + 3$:

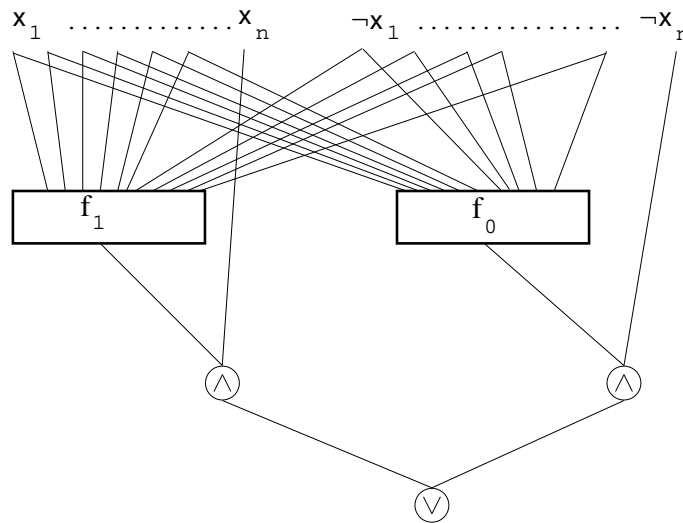


Abbildung 8: Diese Konstruktion liefert das Schema zur Berechnung von f

Durch Auflösen folgt: $T(n) = 2^n - 3$.

Wir entwickeln eine Idee, die die Konstruktion verkleinert:

Beobachtung: Obiger Schaltkreis enthält 2^{n-k} viele disjunkte Subschaltkreise für Funktionen aus B_k . Falls nun $2^{n-k} \gg |B_k| = 2^{2^k}$ ist, so werden sehr viele Funktionen aus B_k mehrfach berechnet.

Sei $T^*(k)$ die Anzahl der Bausteine, um alle Funktionen aus B_k zu berechnen. Wir konstruieren für ein gewisses k die Schaltkreise für Funktionen aus B_k , indem wir *alle* Funktionen aus B_{k-1} berechnen und aus diesen mit der Regel (1) die Funktionen aus B_k zusammensetzen. Es ist

$$\begin{aligned} T^*(2) &= 16 && \text{(wegen } \Omega = B_2 \text{ und } |B_2| = 2^{2^2} = 16) \\ T^*(k) &\leq T^*(k-1) + 2^{2^k} \cdot 3 && (|B_k| = 2^{2^k} \text{ und die 3 aus Regel (1)) \end{aligned}$$

und damit folgt $T^*(k) \leq 3 \cdot 2^{2^k} + 6 \cdot 2^{2^{k-1}}$.

Wir bauen nun den gesamten Schaltkreis wie folgt auf:

Für f_n benutzen wir bis zu einer Ebene k die Rekursion (1). Auf dieser Ebene brechen wir die Rekursion ab, und wir berechnen *alle* Funktionen $f \in B_k$. Dies bedeutet, daß wir in der Rekurrenz $T(n)$ den Aufwandsausdruck für die Rekursion streichen und durch T^* mit dem entsprechenden Parameter ersetzen.

$$\begin{aligned} T(n) &= 2^{n-k} \cdot T(k) + 3 \cdot 2^{n-k} - 3, \text{ für } 2 \leq k \leq n \\ \tilde{T}(n) &= T^*(k) + 3 \cdot 2^{n-k} - 3 \\ &\leq [3 \cdot 2^{n-k} - 3] + [3 \cdot 2^{2^k} + 6 \cdot 2^{2^{k-1}}] . \end{aligned}$$

Wählen wir nun $k := \lfloor \log n \rfloor - 1$, so ergibt sich:

$$\begin{aligned} \tilde{T}(n) &\leq 3 \cdot 2^{n-\lfloor \log n \rfloor+1} + 3 \cdot 2^{2^{\lfloor \log n \rfloor-1}} + 6 \cdot 2^{2^{\lfloor \log n \rfloor-2}} \\ &\leq 3 \cdot 2^{n-\log n+2} + 3 \cdot 2^{2^{\log n-1}} + 6 \cdot 2^{2^{\log n-2}} \\ &= 12 \cdot \frac{2^n}{n} + 3 \cdot 2^{n/2} + 6 \cdot 2^{n/4} \\ &= 12 \cdot \frac{2^n}{n} + o\left(\frac{2^n}{n}\right) . \end{aligned}$$

b) Der Beweis beruht auf einem Zählargument:

Sei $SK(b)$ die Anzahl der Schaltkreise mit b Bausteinen aus B_2 . Pro Baustein gibt es höchstens $(n+b-1)^2$ Möglichkeiten, die zwei Eingänge zu belegen. Jeder Baustein hat eine der 16 möglichen Funktionen aus B_2 .

$$\Rightarrow [(n+b-1)^2 \cdot 16]^b .$$

Außerdem kann jeder der Bausteine als Outputbaustein des Schaltkreises gewählt werden. Also gibt es höchstens

$$(n+b-1)^{2b} \cdot 16^b \cdot b$$

„verschiedene“ Schaltkreise mit genau b Bausteinen. In dieser Zählung ist jeder Schaltkreis (wegen der $b!$ möglichen Numerierungen der Bausteine) $b!$ - mal enthalten.

Es folgt

$$\begin{aligned} SK(b) &\leq [(n+b-1)^{2b} \cdot 16^b \cdot b]/b! \\ &\leq (n+b)^{2b} . \end{aligned}$$

Damit nun jede Funktion $f \in B_n$ mit einem Schaltkreis aus b Bausteinen berechnet werden kann, muß gelten:

$$2^{2^n} = |B_n| \leq SK(b) \leq (n+b)^{2b} \text{ bzw. } 2^n \leq 2b \cdot \log(n+b).$$

Aber für $b := \frac{2^{n-1}}{n} = \frac{1}{2} \frac{2^n}{n}$ folgt:

$$\begin{aligned} 2b \cdot \log(n+b) &< 2b \cdot \log(2b) \\ &= \frac{2^n}{n} \cdot \log\left(\frac{2^n}{n}\right) \\ &= \frac{2^n}{n} \cdot (n - \log n) \\ &= 2^n \cdot \left(1 - \frac{\log n}{n}\right) < 2^n . \end{aligned}$$

Somit gibt es Funktionen in B_n die mehr als $\frac{1}{2} \frac{2^n}{n}$ Bausteine aus B_2 benötigen.

□

3.2 Uniforme Schaltkreisfamilien und Turingmaschinen

Im Folgenden wollen wir Schaltkreise mit TM's vergleichen. TM's berechnen Funktionen $f : \{0,1\}^* \rightarrow \{0,1\}$. Um f mit Schaltkreisen zu berechnen, benötigen wir Familien von Schaltkreisen (SKF) $S := (S_n)_{n \in \mathbb{N}}$, so dass S_n gerade $f_n \equiv f|_{\{0,1\}^n}$ berechnet. Derartige Familien sind jedoch zu mächtig, denn

- Jede (auch nicht berechenbare) Funktion kann durch eine Schaltkreisfamilie mit $C(S_n) \leq \frac{2^n}{n}$ berechnet werden.
- Es gibt nicht berechenbare Funktionen, die durch SKF mit $C(S_n) = 1$ berechnet werden.

Beispiel: Sei $f : \mathbb{N} \rightarrow \{0,1\}$ nicht berechenbar, dann ist $g : \{0,1\}^* \rightarrow \{0,1\}$ mit $g(x) = f(|x|)$ nicht berechenbar. Es gibt auf der anderen Seite natürlich eine Schaltkreisfamilie $(S_n)_{n \in \mathbb{N}}$, die g berechnet, und jeder Schaltkreis S_n erfüllt $C(S_n) = 1$.

Wir müssen daher fordern, daß S_n aus n effizient berechenbar ist.

Bemerkung: Man kann zeigen, daß jeder Schaltkreis S simuliert werden kann durch einen Schaltkreis S' mit $C(S') = O(C(S))$, $D(S') = O(D(S))$, der Outgrad 2 hat.

Wir werden im Folgenden daher stets davon ausgehen, dass Schaltkreise Ingrad und Outgrad 2 haben. Wir wollen einen Schaltkreis kodieren, indem wir für jeden Baustein im Schaltkreis seine Funktion (aus B_2), seine beiden Vorgänger und seine beiden Nachfolger angeben.

Definition 3.6 Eine SKF $S = (S_n)_{n \in \mathbb{N}}$ heißt uniform, falls es eine $O(C(S_n) \cdot \log C(S_n))$ zeitbeschränkte deterministische TM und eine $O(D(S_n))$ platzbeschränkte deterministische TM gibt, die bei Eingabe von 1^n eine Kodierung von S_n erzeugt.

Komplexitätsklassen:

$\text{SIZE}(C(n)) = \{L \subseteq \{0, 1\}^* \mid \text{es gibt uniforme } O(C(n)) \text{ großbeschränkte SKF für } L\}$

$\text{DEPTH}(D(n)) = \{L \subseteq \{0, 1\}^* \mid \text{es gibt uniforme } O(D(n)) \text{ tiefenbeschränkte SKF für } L\}$

Zeiteffiziente Simulation von Schaltkreisen durch Turingmaschinen

Satz 3.7 $\text{SIZE}(C(n)) \subseteq \text{DTIME}(C(n) \cdot \log(C(n))^2)$.

Beweis: Wir zeigen hier nur: $\text{SIZE}(C(n)) \subseteq \text{DTIME}(C(n)^2 \cdot \log(C(n)))$.

Sei $(S_n)_{n \in \mathbb{N}}$ eine Schaltkreisfamilie und $x_1 \dots x_n$ der Input.

1. Berechne Schaltkreis S_n (in Zeit $O(C(n) \cdot \log C(n))$).
2. Werte S_n mit Input $x_1 \dots x_n$ aus. Durchlaufe die Bausteine in topologischer Reihenfolge und ersetze jeden durch das Ergebnis der zugehörigen Operation. Um die Operanden zu finden, muß einmal über die Eingabe gelaufen werden, d.h. die Kosten pro Operation betragen $O(C(n) \cdot \log(C(n)))$. Da $C(n)$ Bausteine ausgewertet werden müssen, folgt die Behauptung.

□

Platzeffiziente Simulation von Schaltkreisen durch Turingmaschinen

Satz 3.8 $\text{DEPTH}(d(n)) \subseteq \text{DTAPE}(d(n))$.

Beweis: Sei S Schaltkreis der Tiefe $d(n)$ und Größe $c(n)$. Beachte: $c(n) \leq 2^{d(n)}$. Man verwende die rekursive Prozedur aus der Bemerkung unter Definition 3.3.

Bandbedarf:

Beschreibung einer Marke benötigt Platz $O(\log c(n))$.

Es werden bis zu $d(n)$ Marken gespeichert.

Insgesamt wird $O(d(n) \cdot \log c(n)) = O(d(n)^2)$ benötigt.

Man kann den Platzbedarf auf $O(d(n))$ reduzieren, dieses Verfahren soll hier aber nicht erläutert werden. Es kann in [1] auf Seite 83 nachgelesen werden. □

Nachdem wir in den Sätzen 3.7 und 3.8 gesehen haben, dass Schaltkreisfamilien im wesentlichen zeit- und platzeffizient durch deterministische Turingmaschinen simuliert werden können, wollen wir in den folgenden Abschnitten umgekehrt Turingmaschinen durch Schaltkreisfamilien simulieren.

Tiefeneffiziente Simulation von Turingmaschinen durch Schaltkreise

Satz 3.9 $\text{DTAPE}(S(n)) \subseteq \text{NTAPE}(S(n)) \subseteq \text{DEPTH}(S(n)^2)$.

Beweis: Sei M eine nichtdeterministische 1-Band Turingmaschine mit Zeitbeschränkung $\alpha \cdot S(n)$. Sei x der Input mit $|x| = n$. M macht $t \leq 2^{\beta \cdot S(n)} = r$ Schritte. Seien C_1, \dots, C_r die Konfigurationen der Länge $\alpha \cdot S(n)$. Sei A die Boole'sche $r \times r$ Matrix mit

$$a_{ij} = \begin{cases} 1 & \text{falls } i = j \text{ oder } C_i \vdash C_j \\ 0 & \text{sonst.} \end{cases}$$

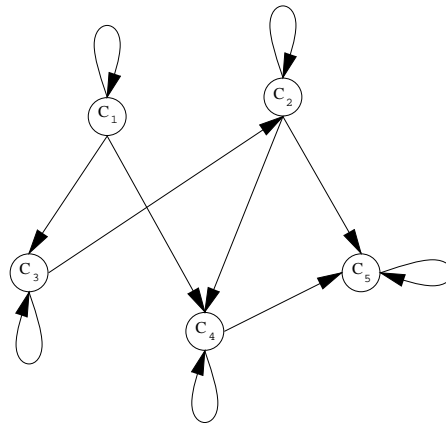


Abbildung 9:

C_1 sei die Startkonfiguration mit x , C_r sei die einzige akzeptierende Enkonfiguration.

M akzeptiert $x \iff$ es gibt einen gerichteten Weg von C_1 nach C_r
 \iff der Eintrag $a_{1,r}^r$ in A^r , dem r -fachen Boole'schen Produkt von A , ist 1.

Das Produkt zweier $r \times r$ Matrizen kann (mit der Schulmethode) in Tiefe $O(\log(r))$ berechnet werden, also A^r durch iteriertes quadrieren in Tiefe $O(\log(r)^2) = O(S(n)^2)$. \square

$A \circ B$ Boole'sches Matrizenprodukt. Es gilt $A \circ B = C \rightsquigarrow C_{ij} = \bigvee_{k=1}^r a_{ik} \wedge b_{kj}$.

Größeneffiziente Simulation von Turingmaschinen durch Schaltkreise

Satz 3.10 Sei $t(n) \geq n$ zeitkonstruierbar, dann gilt:

$$\text{DTIME}(t(n)) \subseteq \text{SIZE}(t(n) \cdot \log t(n)) .$$

Definition 3.11 Eine deterministische k -Band TM heißt oblivious (dt. bewegungsuniform), falls zu jedem Zeitpunkt t die Position der k Köpfe für alle Inputs aus $\{0,1\}^n$ die gleichen sind.

Beweis: (von Satz 3.10)

Der Beweis des Satzes 3.10 ergibt sich direkt aus der Anwendung der folgenden Lemmata über die Simulation einer k -Band TM durch eine oblivious k -Band TM und einer oblivious k -Band TM durch eine Schaltkreisfamilie.

Lemma 3.12 Jede $t(n)$ zeitbeschränkte k -Band TM kann durch eine $O(t(n) \cdot \log t(n))$ zeitbeschränkte oblivious k -Band TM simuliert werden.

Lemma 3.13 Jede $t(n)$ zeitbeschränkte oblivious k -Band TM kann durch eine uniform $O(t(n))$ größenbeschränkte Schaltkreisfamilie simuliert werden.

Beweis: (von Lemma 3.12)

Wir zeigen nur ein etwas schwächeres Resultat: Jede $t(n)$ zeitbeschränkte k -Band TM M kann durch eine $O(t(n)^2)$ zeitbeschränkte oblivious k -Band TM simuliert werden.

Beschreibung der Simulation eines Schrittes für ein Band: Wir wollen die in Bild 10 beschriebene Darstellung von Konfigurationen von M benutzen.

- Laufe von Endmarker \otimes zu \otimes , führe bei $\$$ den Übergang von M aus (evtl. Inschrift modifizieren), falls Kopfbewegung N oder R war. Schreibe rechtes \otimes eine Stelle weiter nach rechts.
- Laufe zurück zum linken \otimes , führe bei $\$$ Übergang von M aus; falls Kopfbewegung L war, verschiebe linkes \otimes eine Stelle weiter nach links.

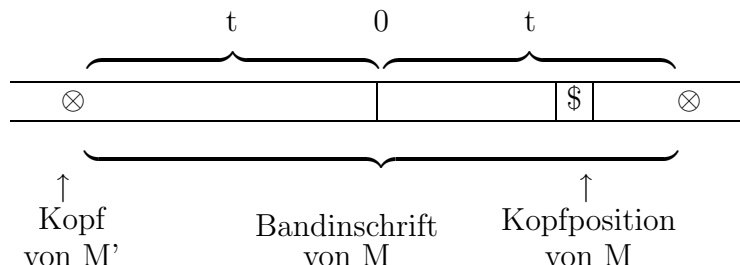


Abbildung 10: Konstruierte oblivious TM

Daraus ergibt sich folgende Laufzeit: $\sum_{t=1}^{t(n)} (4t + c) = O(t(n)^2)$ □ Lemma 3.12

Beweis: (von Lemma 3.13)

Sei $\delta : Q \times \Gamma^k \rightarrow Q \times A^k \times \{L, N, R\}^k$ die Übergangsfunktion von M . Kodiere Q und Γ und L, N, R jeweils in $\{0, 1\}^b$ (b konstant, genügend groß), so ergibt sich eine Boole'sche Version δ' von $\delta : \delta' : \{0, 1\}^b \times (\{0, 1\}^b)^k \rightarrow \{0, 1\}^b \times (\{0, 1\}^b)^k \times \{0, 1\}^{2 \cdot k}$. S_δ sei der Schaltkreis für δ' . (Dieser hat konstante Größe und Tiefe.)

Der Schaltkreis für die Simulation von M bei Eingabelänge n setzt sich zusammen aus $t(n)$ vielen Kopien von S_δ , die gemäß der (inputunabhängigen) Kopfbewegungen hintereinander geschaltet sind.

Als Beispiel für eine Konfiguration nach t Schritten betrachte Abbildung 11, und für den daraus sich ergebenden Teil des Schaltkreises Abbildung 12.

Die Größe ist $O(t(n))$, da der Schaltkreis aus $t(n)$ Exemplaren von S_δ besteht.

□ Lemma 3.13

Hintereinanderschalten der Simulationen beendet den Beweis von Satz 3.10. □ Satz 3.10

3.3 Parallele Komplexitätsklassen

In diesem Abschnitt sollen einige der wichtigsten parallelen Komplexitätsklassen mit Hilfe von Schaltkreisen definiert werden. Wir verallgemeinern dazu das Schaltkreiskonzept.

Definition 3.14 *Ein Schaltkreis mit unbeschränkten Ingrad (unbounded fan-in), ub-SK, mit n Eingaben und m Ausgaben hat $2n$ Quellen für $x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n$ und Bausteine \wedge, \vee mit beliebiger Zahl von Operanden. Der Schaltkreis berechnet $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$, wobei nur \wedge und \vee Bausteine erlaubt sind. Familien von ub-SK, uniforme ub-SKF, sind analog zu Def 3.6 definiert.*

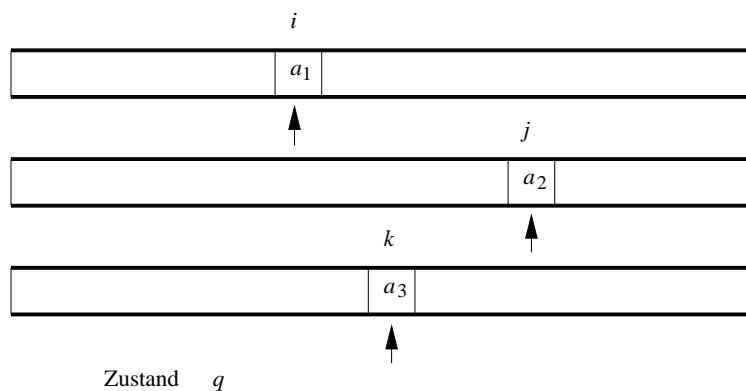


Abbildung 11: Konfiguration nach t Schritten

Definition 3.15 (Parallele Komplexitätsklassen)

1. $NC^k = \{L \in \{0, 1\}^* \mid \text{es gibt uniforme SKF für } L \text{ mit Größe } \text{poly}(n) \text{ und Tiefe } O(\log(n)^k)\}$
2. $AC^k = \{L \in \{0, 1\}^* \mid \text{es gibt uniforme ub-SKF für } L \text{ mit Größe } \text{poly}(n) \text{ und Tiefe } O(\log(n)^k)\}$
3. $NC = \bigcup_{k \geq 1} NC^k$
4. $AC = \bigcup_{k \geq 1} AC^k$

Dabei steht NC für „Nick’s Class“. (Geprägt von Steve Cook zur Würdigung von Nick Pippenger. Dieser hat seinerseits später die Klasse SC („Steve’s Class“) definiert.) AC steht für „Alternation Class“.

Bemerkung:

$$AC^0 \subseteq NC^1 \subset AC^1 \subset NC^2 \subset \dots \subseteq AC^k \subseteq NC^{k+1} \subseteq AC^{k+1} \subseteq \dots \subseteq AC = NC$$

d.h. bei der Simulation des unbounded fan-in durch bounded fan-in wächst die Tiefe höchstens um den Faktor $O(\log n)$.

Beweis: Wir müssen nur zeigen, dass

1. $NC^k \subseteq AC^k$, was klar ist, da ein Schaltkreis für NC ein unbounded fan-in Schaltkreis ist, und
2. $AC^k \subseteq NC^{k+1}$. Man beachte, dass jeder Baustein eines AC^k - Schaltkreises nur höchstens $\text{poly}(n)$ Ingrad hat (mehr Bausteine kann der Schaltkreis nicht besitzen). Der Ingrad wird durch einen binären Baum der Tiefe $O(\log \text{poly}(n)) = O(\log n)$ und Größe $O(\text{poly}(n))$ ersetzt. Dann folgt, dass die Tiefe nur um den Faktor $O(\log n)$ größer wird, also von $\log(n)^k$ auf $\log(n)^{k+1}$ steigt.

□

Beispiele:

Probleme aus NC :

1. Addition zweier n-Bit Zahlen $\in NC^1$
2. Multiplikation zweier n-Bit Zahlen $\in NC^1$
3. Division zweier n-Bit Zahlen $\in NC^2$
4. Matrix Multiplikation ($n \times n$ Matrix mit ℓ -Bit Zahlen, ℓ konstant) $\in NC^1$
5. Berechnung einer Determinante $\in NC^2$
6. Berechnung des transitiven Abschlusses eines Graphen $\in NC^2$

$$\delta(q, a_1, a_2, a_3) = (q', a'_1, a'_2, a'_3, R, N, L)$$

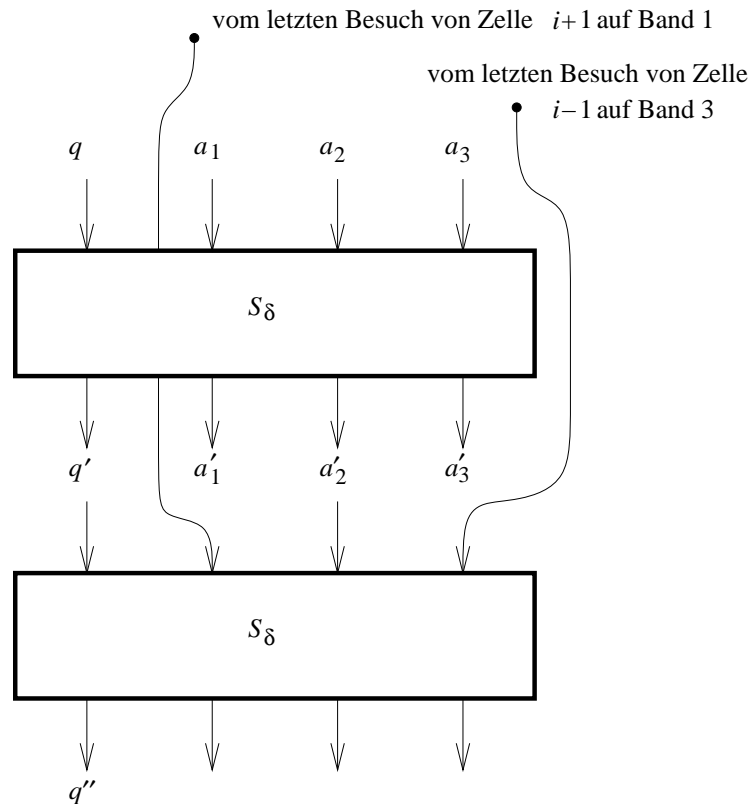


Abbildung 12: Schaltkreis für die Simulation

Probleme aus AC^0 :

1. Addition zweier n-Bit Zahlen
2. Vergleich zweier n-Bit Zahlen
3. Maximumbestimmung von m ℓ -Bit Zahlen

Probleme aus $AC^1 \setminus AC^0$:

1. Multiplikation zweier n-Bit Zahlen
2. Sortieren
3. Parity

Satz 3.16 Falls $P \neq NC^k$ für $k \geq 2$, so sind P -vollständige Probleme nicht in NC^k .

Man kann NC auffassen als die Teilklasse von P , deren Probleme nicht nur in polynomieller Zeit gelöst werden können, sondern zudem noch effizient parallelisierbar sind, d.h. die mit $poly(n)$ großen Schaltkreisen der Tiefe $poly(\log n)$ berechenbar sind. Insbesondere folgt

aus dem Satz 3.16, dass aus $P \neq NC$ folgt, dass P -vollständige Probleme nicht effizient parallelisierbar sind.

Beispiele: Falls $P \neq NC$ ist, so sind (die P -vollständigen Probleme) Circuit Value (gegeben sind ein Schaltkreis und eine Eingabe; werte den SK aus), Flußproblem und Lineare Programmierung nicht effizient parallelisierbar.

Parallele Rechenmodelle

Unter einem Parallelrechner versteht man allgemein eine gewisse Anzahl von (gleichen) Rechnern (oder Prozessoren), die *gemeinsam* ein Problem lösen, d.h. eine Funktion $f : \{0, 1\}^* \rightarrow \{0, 1\}$ berechnen. Daraus ergibt sich die Konsequenz, dass die Prozessoren miteinander kommunizieren müssen (Datenaustausch). Schematisch kann dies wie in Abbildung 3.3 dargestellt werden:

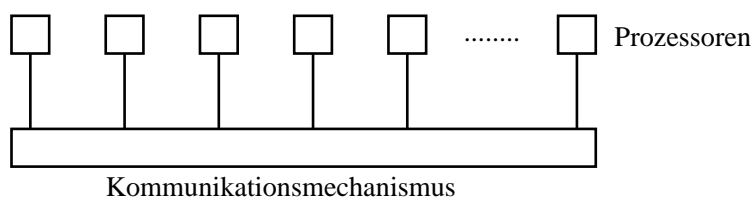


Abbildung 13: Paralleles Rechenmodell

Die einzelnen Prozessoren wollen wir als RAM auffassen. Die verschiedenen Rechenmodelle unterscheiden sich nach ihren Kommunikationsmechanismen.

- Typ 1: *Netzwerke*: In diesem Rechenmodell sind jeweils zwei Prozessoren durch *Links* verbunden. Beispiele sind z.B. das Cliquennetzwerk (vollständiges Netzwerk), der Hypercube oder das Gitter.
- Typ 2: *Parallele Registermaschine* (PRAM): In diesem Rechenmodell kommunizieren die einzelnen Prozessoren über einen gemeinsamen Speicher.

Zwischen den Rechenmodellen bestehen (informell) folgende Beziehungen:

- t Schritte eines beliebigen n -Netzwerks (Netzwerk mit n Prozessoren) können durch t Schritte eines vollständigen n -Netzwerks simuliert werden.
- t Schritte eines vollständigen n -Netzwerks können in $O(t \cdot \log(n))$ Schritten auf geeigneten n -Netzwerken mit beschränktem Grad simuliert werden (z.B. Butterfly- und de Bruijn- Netzwerk).
- t Schritte eines n -Netzwerks können in $O(t)$ Schritten auf n -PRAM simuliert werden (\leadsto PRAM ist das „stärkste“ parallele Rechenmodell).
- t Schritte einer n -PRAM können in $O(t \cdot \log(n))$ Schritten auf einem vollständigen n -Netzwerk simuliert werden (sehr aufwendige Simulation).

In PRAMs kann es beim Lesen und Schreiben zu Konflikten kommen, die durch verschiedene Mechanismen gelöst werden können:

Speicherzugriffskonflikte:

1. CR: mehrere Prozessoren wollen gleichzeitig in einer Zelle lesen (concurrent read)
2. CW: mehrere Prozessoren wollen gleichzeitig in eine Zelle schreiben (concurrent write)

Lösung:

1. CR: alle lesenden Prozessoren erhalten den Inhalt der Speicherzelle (kein Problem)
2. common CW: nur erlaubt, wenn alle Prozessoren das gleiche schreiben
priority CW: der Prozessor mit der größten Nummer gewinnt
arbitrary CW: *irgendein* unvorhersehbarer Prozessor gewinnt

Definition 3.17 Eine PRAM besteht aus unendlich vielen Prozessoren P_1, P_2, \dots und einer unendlichen Folge von Registern M_0, M_1, \dots (M_i ist dabei eine globale Speicherzelle). Jeder Prozessor ist eine RAM (Operationen $+, -, \text{shift}$) mit folgenden Zusätzen:

- Jeder Prozessor P_i hat ein spezielles read-only Register PID mit Inhalt i
- Die PRAM hat ein endliches Programm (wie eine RAM), welches jedem Prozessor bekannt ist. Dabei gibt es folgende zusätzlichen Befehle:
 - read PID
 - direkter und indirekter Zugriff auf gemeinsamen Speicher
 - aktiviere k : im nächsten Schritt beginnt der Prozessor $P_{c_i(k)}$ das Programm abzuarbeiten, aber nur dann, wenn er noch nicht arbeitet
- Zugriffskonfliktregelung: priority CRCW
- Die Arbeitsweise bei Eingabe $(x_1, \dots, x_n) \in \{0, 1\}^*$ ist wie folgt:
 - Zu Beginn steht x_i in M_i und n steht in M_0 . Alle anderen globalen und lokalen Register sind undefiniert, nur P_1 ist aktiv.
 - In einem Schritt arbeiten alle Prozessoren gleichzeitig (synchron) ihren jeweils nächsten Schritt ab.
 - Sobald alle Prozessoren END erreicht haben, hält die PRAM an.
- Zeitaufwand: $T_M(\bar{x}) = \text{Anzahl der Schritte bei Input } x \text{ (uniforme Kosten)}$
- Prozessoraufwand: $P_M(\bar{x}) = \text{maximale Anzahl der aktiven Prozessoren bei Eingabe } x$
- $t(n)$ -Zeitbeschränkung und $s(n)$ -Platzbeschränkung definiert man analog zu RAMs (maximiert über alle Eingaben der Länge n)

Über die Ressourcen Zeit- und Prozessorzahl definieren wir die folgenden Komplexitätsklassen für PRAMs.

Definition 3.18 (Komplexitätsklassen für PRAMs)

$\text{PRAM}(t(n), p(n)) = \{L \in \{0, 1\}^* \mid \text{es gibt } O(t(n)) \text{ zeit- und } O(p(n)) \text{ prozessorbeschränkte PRAM für } L\}$

$\text{PRAM}(t(n)) = \{L \in \{0, 1\}^* \mid \text{es gibt } O(t(n)) \text{ zeitbeschränkte PRAM für } L\}$,
also keine Prozessorbeschränkung.

Bemerkung: $\text{PRAM}(t(n)) = \text{PRAM}(t(n), 2^{t(n)})$, denn in jedem Schritt kann sich die Zahl der aktiven Prozessoren höchstens verdoppeln.

Satz 3.19 (Stockmeyer/Vishkin 1984)

- a) Jede uniform $c(n)$ -größen- und $d(n)$ -tiefenbeschränkte ub-SKF kann durch $O(d(n))$ -zeit- und $O(c(n))$ -prozessorbeschränkte PRAM simuliert werden.
- b) Jede $t(n)$ -zeit- und $p(n)$ -prozessorbeschränkte PRAM kann durch eine $O(t(n))$ -tiefen- und $O((n + p(n) \cdot t(n))^4)$ -größenbeschränkte ub-SKF simuliert werden.

Korollar 3.20

- $\text{PRAM}((\log n)^k, \text{poly}(n)) = AC^k$
- $\text{PRAM}(\text{poly } \log n, \text{poly}(n)) = AC$
- $\text{PRAM}(\text{poly}(n)) = PSPACE$

Beweis: (von Satz 3.19)

- a) Reserviere für jeden Baustein v des SK einen Prozessor P_v und eine Speicherzelle M_v . Der Inhalt von M_v ist der Wert, der in v berechnet wird.
 - P_v kennt $x_v \in \{0, 1, *\}$ (aktueller Wert von M_0 , * steht für „noch nicht berechnet“).
 - P_v kennt seine Nachfolger.
 - Ziel: falls alle Vorgänger v_1, \dots, v_l von v' ein $x_i \in \{0, 1\}$ haben, soll $x_{v_1} \wedge \dots \wedge x_{v_l}$ in $M_{v'}$ berechnet werden.
Algorithmus:
 - * $P_{v'}$ schreibt 1 nach $M_{v'}$.
 - * Falls $M_{v_i} = 0$ ist, schreibt P_{v_i} 0 nach $M_{v'}$.
 - Das logische ODER wird analog berechnet.

b) *Idee:* Realisiere PRAM in Hardware, beachte:

ALU: Addition, Subtraktion, Shift (kann in konstanter Tiefe realisiert werden)

Speicherzugriff: Multiplexer, Demultiplexer (kann in konstanter Tiefe realisiert werden)

Somit ergibt sich ein ub-Schaltkreis der Tiefe $O(d(n))$. Für die Analyse der Größe werden die Größen von Addierern, Multiplexern etc. benötigt. Beachte: Es werden nur Zahlen der Länge $O(t(n) + \log(p(n)))$ erzeugt, also müssen nur Addierer etc. mit so großen Eingaben gebaut werden.

□

3.4 Eine untere Schranke für PARITY

In diesem Kapitel betrachten wir eine Verallgemeinerung von ub-SKen, in der zusätzlich zu den Operationen \vee, \wedge, \neg auch die Operation

$$\boxed{\text{mod}_3}(x_1, x_2, \dots, x_n) := \begin{cases} 0 : \sum x_i \equiv 0 \pmod{3} \\ 1 : \sum x_i \not\equiv 0 \pmod{3} \end{cases}$$

erlaubt ist. Falls wir $\boxed{\text{mod}_2}$, also \oplus erlauben würden, könnte *Parity* (x_1, \dots, x_n) in Tiefe 1 und Größe 1 berechnet werden. Im Gegensatz dazu zeigen wir:

Satz 3.21 (Razbarov, Smolenski) *Für genügend große n hat jeder ub-SK (mit zusätzlichen $\boxed{\text{mod}_3}$ Operationen) der Tiefe d für Parity_n mindestens Größe $0.075 \cdot (\sqrt{2})^{n^{1/(2d)}}$. Insbesondere ist für polynomielle Größe Tiefe $\Omega(\log(n)/\log \log(n))$ nötig, $\text{Parity}_n \notin AC^0$.*

Beweis:

Wir zeigen folgende zwei Lemmata:

Lemma 3.22 *$f \in B_n$ werde von einem Schaltkreis S wie oben beschrieben berechnet in Tiefe d und Größe s . Dann gibt es für jedes $\varepsilon > 0$ ein multilineares Polynom $p(x_1, \dots, x_n)$ über GF_3 vom Grad $(2\lceil \log(\frac{s}{\varepsilon}) \rceil)^d$, so dass $f(x_1, \dots, x_n) \neq p(x_1, \dots, x_n)$ für höchstens $\varepsilon \cdot 2^n$ viele Inputs aus $\{0, 1\}^n$ gilt.*

Lemma 3.23 *Jedes multilineare Polynom $p(x_1, \dots, x_n)$ über GF_3 mit Grad höchstens \sqrt{n} erfüllt: $p(x_1, \dots, x_n) \neq \text{Parity}_n(x_1, \dots, x_n)$ für mindestens $0.15 \cdot 2^n$ Inputs.*

Daraus können wir nun einfach den Satz folgern:

Der Schaltkreis aus Lemma 3.22 berechne $f = \text{Parity}_n$, das ε aus Lemma 3.23 erfülle $\varepsilon = 0.15$.

Dann folgt:

$$(2\lceil \log(\frac{s}{\varepsilon}) \rceil)^d \geq \sqrt{n} .$$

Dieses ist äquivalent zu

$$\lceil \log\left(\frac{s}{\varepsilon}\right) \rceil \geq \frac{1}{2} \cdot n^{1/(2d)} .$$

Somit ist

$$\log\left(\frac{s}{\varepsilon}\right) \geq \frac{1}{2} \cdot n^{1/(2d)} - 1 ,$$

also

$$s \geq \varepsilon \cdot 2^{\frac{1}{2} \cdot n^{1/(2d)} - 1} = 0.075 \cdot (\sqrt{2})^{n^{1/(2d)}} .$$

□ Satz 3.21

Beweis: (von Lemma 3.22)

Wir zeigen zuerst:

Behauptung 3.24 Für jede Wahrscheinlichkeitsverteilung w auf $\{0, 1\}^n$ und jedes $\delta > 0$ gibt es ein Polynom $p(x_1, \dots, x_n)$ über GF_3 vom Grad $2\lceil \log(\frac{1}{\delta}) \rceil$, so dass gilt: $\text{Prob}_w [\text{OR}(x_1, \dots, x_n) \neq p(x_1, \dots, x_n)] \leq \delta$. (Analoges gilt für AND).

Beweis: (von Behauptung 3.24)

Für $S \subseteq \{1, \dots, n\}$ sei $f_S(x_1, \dots, x_n) = (\sum_{i \in S} x_i)^2 \pmod 3$. Folgendes gilt:

- i) $f_S(x_1, \dots, x_n) \in \{0, 1\}$
- ii) Sei $\text{OR}(x_1, \dots, x_n) = 0$. Dann gilt $f_S(0, \dots, 0) = 0$.
- iii) Sei $x_{i_0} = 1$ für $i_0 \in \{1, \dots, n\}$, $S' \subseteq \{1, \dots, n\} \setminus \{i_0\}$. Dann $f_{S'}(x_1, \dots, x_n) = 1$ oder $f_{S' \cup \{i_0\}}(x_1, \dots, x_n) = 1$.
- iv) Sei $\text{OR}(x_1, \dots, x_n) = 1$.
Dann gilt für zufälliges $S \subseteq \{1, \dots, n\}$: $\text{Prob}_S [f_S(x_1, \dots, x_n) = 1] \geq \frac{1}{2}$.
- v) $\text{grad}(f_S) \leq 2$

(i), (ii), (v) sind klar. (iii) rechnen wir einfach nach durch Fallunterscheidung (im Folgenden sind alle Summen $\pmod 3$ zu verstehen):

$$R := \sum_{i \in S'} x_i, \quad f_{S'}(x_1, \dots, x_n) = R^2, \quad f_{S' \cup \{i_0\}}(x_1, \dots, x_n) = R^2 + 2R + 1$$

- $R = 0 \Rightarrow f_{S' \cup \{i_0\}} = R^2 + 2R + 1 = 1 \quad \checkmark$
- $R = 1 \Rightarrow f_{S'} = R^2 = 1, \quad f_{S' \cup \{i_0\}} = 1 \quad \checkmark$
- $R = 2 \Rightarrow f_{S'} = R^2 = 1 \quad \checkmark$

Zum Beweis von (iv) beachte: Auf Grund der Gleichverteilung der $S \subseteq \{1, \dots, n\}$ ist

$$\text{Prob}_S [f_S(x) = 1] = \frac{1}{2^n} \sum_{S \subseteq \{1, \dots, n\}} f_S(x) .$$

Wegen $\text{OR}(x) = 1$ existiert $i_0 \in \{1, \dots, n\}$ mit $x_{i_0} = 1$.

Wir unterteilen den Laufbereich der Summe in

1. diejenigen $S \subseteq \{1, \dots, n\}$ mit $i_0 \notin S$; d.h. $S = S' \subseteq \{1, \dots, n\} \setminus \{i_0\}$.
2. diejenigen $S \subseteq \{1, \dots, n\}$ mit $i_0 \in S$; d.h. $S = S' \cup \{i_0\}$, $S' \subseteq \{1, \dots, n\} \setminus \{i_0\}$.

Weil diese Unterteilung disjunkt ist, gilt:

$$\sum_{S \subseteq \{1, \dots, n\}} f_S(x) = \sum_{S' \subseteq \{1, \dots, n\} \setminus \{i_0\}} (f_{S'}(x) + f_{S' \cup \{i_0\}}(x)) \stackrel{(iii)}{\geq} \sum_{S' \subseteq \{1, \dots, n\} \setminus \{i_0\}} 1 = 2^{n-1} .$$

Eigenschaften (ii) und (iv) bedeuten einen *einseitigen Fehler mit beschränkter Wahrscheinlichkeit*. Wir wenden nun die Technik der *Wahrscheinlichkeitsverstärkung* an, indem wir $l = \lceil \log(\frac{1}{\delta}) \rceil$ viele Mengen S_1, \dots, S_l zufällig und unabhängig in $\{1, \dots, n\}$ wählen und betrachten:

$$f_{S_1, \dots, S_l}(x) := 1 - \prod_{i=1}^l (1 - f_{S_i}(x)) .$$

Auch für dieses f_{S_1, \dots, S_l} gilt (i), (ii); (v) lautet jetzt: $\text{grad}(f_{S_1, \dots, S_l}) \leq 2l = 2 \lceil \log(\frac{1}{\delta}) \rceil$; und anstelle von (iv) haben wir für alle $x = (x_1, \dots, x_n)$ mit $\text{OR}(x) = 1$:

$$\begin{aligned} \text{Prob} [f_{S_1, \dots, S_l}(x) = 1] &= 1 - \text{Prob} [\forall i \in \{1, \dots, l\} : f_{S_i}(x) = 0] \\ &= 1 - \prod_{i=1}^l \text{Prob} [f_{S_i}(x) = 0] \\ &\stackrel{(iv)}{\geq} 1 - (1/2)^l \geq 1 - \delta . \end{aligned}$$

Also folgt für jedes $x \in \{0, 1\}^n$: $\text{Prob} [f_{S_1, \dots, S_l}(x) \neq \text{OR}(x)] \leq \delta$.

Um nun die Behauptung zu beweisen, betrachte eine beliebige Wahrscheinlichkeitsverteilung w auf $\{0, 1\}^n$ und beliebiges $\delta > 0$. Sei wie oben $l = \lceil \log(\frac{1}{\delta}) \rceil$.

Wir zeigen, dass die Behauptung gilt, falls wir für p ein f_{S_1, \dots, S_l} für passende S_1, \dots, S_l wählen. Diese passenden S_1, \dots, S_l können wir nicht explizit zu w konstruieren, aber wir werden durch ein einfaches Abzählargument ihre Existenz nachweisen.

Für $x \in \{0,1\}^n, S_1, \dots, S_l \subseteq S$ sei $\lambda(x; S_1, \dots, S_l) \in \{0,1\}$ genau dann 1, wenn $f_{S_1, \dots, S_l}(x) \neq \text{OR}(x_1, \dots, x_n)$ gilt. Damit ist

$$\begin{aligned} \sum_{S_1, \dots, S_l} \left(\sum_{x \in \{0,1\}^n} w(x) \cdot \lambda(x; S_1, \dots, S_l) \right) &= \sum_{x \in \{0,1\}^n} w(x) \cdot \sum_{S_1, \dots, S_l} \lambda(x; S_1, \dots, S_l) \\ &\leq \sum_{x \in \{0,1\}^n} w(x) \cdot 2^{nl} \cdot \delta = 2^{n \cdot l} \cdot \delta . \end{aligned}$$

Somit gibt es S_1, \dots, S_l mit $\sum_{x \in \{0,1\}^n} w(x) \cdot \lambda(x; S_1, \dots, S_l) \leq \delta$, d.h. mit $\text{Prob}_w [f_{S_1, \dots, S_l}(x) \neq \text{OR}(x)] \leq \delta$. f_{S_1, \dots, S_l} ist das gesuchte Polynom p .

□ Behauptung 3.24

Betrachte einen Schaltkreis der Größe s und Tiefe d für *Parity*. Für Wahrscheinlichkeitsverteilung w seien g_w, h_w die Polynome aus der Behauptung für OR bzw. AND, wobei die Fehlerschranke δ als ε/s gewählt ist. g_w, h_w haben somit Grad $2 \lceil \log(s/\varepsilon) \rceil =: D$.

Wir werden nun jedem Baustein des Schaltkreises ein Polynom zuordnen, und zwar Level für Level. Bausteine auf Level i erhalten Polynome vom Grad $\leq D^i$.

Level 0:

$$\left. \begin{array}{l} x_i \text{ bekommt Polynom } g(x) = x_i \\ \bar{x}_i \text{ bekommt Polynom } g(x) = 1 - x_i \end{array} \right\} \text{ beide Polynome haben Grad } 1 = D^0 .$$

Level $i > 0$:

Gatter Δ habe Inputs (von Level $i - 1$), denen die Polynome p_1, \dots, p_k zugeordnet sind, jeweils mit Grad $\leq D^{i-1}$. Wir definieren das Polynom g für Δ abhängig vom Typ von Δ .

$$\Delta \stackrel{\wedge}{=} \boxed{\text{mod } 3} : g(x) = \left(\sum_{i=1}^k p_i(x) \right)^2 \\ \longrightarrow g(x) \text{ hat Grad } \leq 2D^{i-1} \leq D^i .$$

$$\Delta \stackrel{\wedge}{=} \text{OR} : \text{ Betrachte auf } \{0,1\}^k \text{ die Wahrscheinlichkeitsverteilung } w(y) = \frac{1}{2^n} \cdot \\ \#\{x \in \{0,1\}^n : p_1(x) = y_1, \dots, p_k(x) = y_k\} . \text{ Das zu } \Delta \text{ gehörige} \\ \text{Polynom ist dann } g(x) = g_w(p_1(x), \dots, p_k(x)) . \\ \longrightarrow g(x) \text{ hat Grad } \leq D \cdot D^{i-1} = D^i .$$

$$\Delta \stackrel{\wedge}{=} \text{AND} : \text{ Wähle } w \text{ wie oben, und } g(x) = h_w(p_1(x), \dots, p_k(x)) . \\ \longrightarrow h(x) \text{ hat ebenfalls Grad } \leq D^i .$$

Für wieviele Inputs $x \in \{0,1\}^n$ macht das Polynom g zu Baustein Δ einen Fehler, d.h. berechnet einen anderen Wert als der Baustein?

Auf Level 0 werden offensichtlich keine Fehler gemacht.

Betrachte Δ auf Level $i > 0$, Vorgänger $\Delta_1, \dots, \Delta_k$ mit Polynomen p_1, \dots, p_k , an denen auf Inputmenge $U \subseteq \{0, 1\}^n$ keine Fehler gemacht werden.

$\Delta \stackrel{\wedge}{=} \boxed{\text{mod } 3}$: g macht auf U keine Fehler, da

$$g(x) = \left(\sum_{i=1}^k p_i(x) \right)^2 \pmod{3} = \begin{cases} 0 & \Sigma p_i(x) = 0 \pmod{3} \\ 1 & \text{sonst.} \end{cases}$$

$\Delta \stackrel{\wedge}{=} \text{OR}$: Sei $g_w(y_1, \dots, y_k) \neq \text{OR}(y_1, \dots, y_k)$ für alle $y \in T \subseteq \{0, 1\}^k$. Dann ist $w(T) \leq \delta$, d.h. nach Konstruktion von w :

$$\begin{aligned} \#\{x \in \{0, 1\}^n \mid p_1(x) = y_1, \dots, p_k(x) = y_k \text{ für ein } (y_1, \dots, y_k) \in T\} \\ = 2^n \cdot \sum_{y \in T} w(y) = 2^n \cdot w(T) \leq 2^n \cdot \delta. \end{aligned}$$

Also macht das zu p gehörige Polynom höchstens $2^n \cdot \delta$ Fehler auf U .

Analoges gilt für $\Delta \stackrel{\wedge}{=} \text{AND}$.

Resultat:

An allen s Bausteinen zusammen machen die Polynome höchstens $s \cdot \delta \cdot 2^n = \varepsilon \cdot 2^n$ Fehler. Insbesondere macht das zum Output gehörige Polynom p höchstens $\varepsilon \cdot 2^n$ Fehler. p hat Grad D^d , ist allerdings noch nicht multilinear. Das kann aber einfach gemacht werden: Da uns p nur für Inputs aus $\{0, 1\}^n$ interessiert, ändert sich der Funktionswert nicht, wenn wir in der Standarddarstellung immer x_i^a für $a > 1$ durch x_i ersetzen. Das so erzeugte multilineare Polynom erfüllt alle Anforderungen von Lemma 3.22.

□ Lemma 3.22

Beweis: (von Lemma 3.23)

Der Einfachheit halber transformieren wir den Inputbereich von $\{0, 1\}^n$ nach $\{-1, 1\}^n$ durch $x \mapsto 1 - 2x$. D.h.: 0 wird durch 1 ersetzt, 1 durch -1. Anstatt Parity berechnen wir entsprechend *Parity**: $\{-1, 1\}^n \rightarrow \{-1, 1\}$ mit $\text{Parity}^*(x_1, \dots, x_n) = \prod_{i=1}^n x_i$. Da wir nur eine lineare Transformation vornehmen, hat sie keinen Einfluß auf die Grade von Polynomen.

Behauptung 3.25 Sei $p : \{-1, 1\}^n \rightarrow \{-1, 1\}$ ein beliebiges multilineares Polynom vom Grad $\leq \sqrt{n}$ über GF_3 . Dann gilt: $p(x) = \text{Parity}^*(x)$ für höchstens $0.85 \cdot 2^n$ viele Inputs.

Daraus folgt sofort das Lemma.

Beweis: (von Behauptung 3.25)

Sei p wie in der Behauptung, $A = \{x \in \{-1, 1\}^n : p(x) = \text{Parity}^*(x)\}$. Sei $F = \{f : A \rightarrow \{-1, 0, 1\}\}$, $f \in F$. Sei $q(x) = \sum_{i=1}^r a_i \prod_{j \in S_i} x_j$ eine Darstellung von f als multilineares Polynom über GF_3 , mit $a_i \in \{0, 1, 2\}$.

(Beachte: $-1 = 2 \pmod 3$; jede Funktion über einem endlichen Körper kann als Polynom dargestellt werden; falls der Definitionsbereich auf eine Teilmenge $A \subseteq \{-1, 1\}^n$ eingeschränkt wird, kann es multilinear gemacht werden.)

Wir werden q jetzt so umbauen, dass es zwar weiterhin auf A die Funktion f darstellt, aber Grad $\leq \frac{1}{2}(n + \sqrt{n})$ bekommt. Dazu betrachte ein S_i mit $|S_i| > \frac{1}{2}(n + \sqrt{n})$. Dann gilt: $p(x) = \prod_{j=1}^n x_j = \prod_{j \in S_i} x_j \cdot \prod_{j \notin S_i} x_j = \prod_{j \in S_i} x_j / \prod_{j \notin S_i} x_j$ für alle $x \in A$, da für $a, b \in \{-1, 1\}$ in GF_3 gilt: $a \cdot b = a/b$.

Somit können wir für $x \in A$ das Produkt $\prod_{j \in S_i} x_j$ als $\left(\prod_{j \notin S_i} x_j \right) \cdot p(x)$ darstellen, also als Polynom vom Grad $(n - |S_i|) + \text{Grad}(p) \leq n - \frac{1}{2}(n + \sqrt{n}) + \sqrt{n} = \frac{1}{2}(n + \sqrt{n})$.

Wir erhalten für f auf A eine Polynomdarstellung in GF_3 vom Grad $\frac{1}{2}(n + \sqrt{n})$. Folglich gilt:

$$\begin{aligned} 3^{\#A} &= \#F \\ &\leq \# \left\{ \begin{array}{l} \text{multilineare Polynome mit } n \text{ Inputs, Koeffizienten aus } \{0, 1, 2\} \\ \text{und Grad } \leq \frac{1}{2}(n + \sqrt{n}) \end{array} \right\} \\ &= 3^{\sum_{i=0}^r \binom{n}{i}} \end{aligned}$$

für $r = \frac{1}{2}(n + \sqrt{n})$. Also folgt: $\#A \leq \sum_{i=0}^r \binom{n}{i}$.

Lemma 3.26 impliziert somit die Behauptung.
Damit ist das Lemma gezeigt.

□ Behauptung 3.25
□ Lemma 3.23

Lemma 3.26 $S := \sum_{i=0}^{\frac{1}{2}(n+\sqrt{n})} \binom{n}{i} \leq 0.85 \cdot 2^n$.

Beweis: (von Lemma 3.26)

Wir machen Anleihen in der Stochastik und werden den Grenzwertsatz von Moivre-Laplace benutzen:

Sei $X_{p,n}$ die Zufallsvariable, die für n unabhängige Experimente, die jeweils mit Wahrscheinlichkeit p erfolgreich sind, angibt, wieviele Experimente erfolgreich waren. Die Wahrscheinlichkeit für höchstens k Erfolge ist durch die kumulierte Binomialverteilung gegeben:

$$F_{p,n}(k) = \sum_{i=0}^k \binom{n}{i} p^i (1-p)^{n-i} .$$

Unsere gesuchte Zahl S erfüllt somit

$$\frac{1}{2^n} S = F_{\frac{1}{2},n}\left(\frac{1}{2}(n + \sqrt{n})\right) .$$

Wir betrachten die zentrierte, normierte Version $X_{p,n}^*$ von $X_{p,n}$:

$$X_{p,n}^* = \frac{X_{p,n} - p \cdot n}{\sqrt{p(1-p)n}} .$$

Somit gilt:

$$\text{Prob} (a \leq x_{p,n} \leq b) = \text{Prob} \left(\underbrace{\frac{a - pn}{\sqrt{p(1-p)n}}}_{= a^*} \leq X_{p,n}^* \leq \underbrace{\frac{b - pn}{\sqrt{p(1-p)n}}}_{= b^*} \right) .$$

Für unsere Situation, $p = \frac{1}{2}$, $a = 0$, $b = \frac{1}{2}(n + \sqrt{n})$, ergibt sich:

$$\frac{1}{2^n} S = \text{Prob} (-\sqrt{n} \leq X_{\frac{1}{2},n}^* \leq 1) .$$

Wir nutzen nun den Grenzwertsatz von Moivre-Laplace:

Satz 3.27 $\lim_{n \rightarrow \infty} (\text{Prob} (\alpha \leq X_{p,n}^* \leq \beta)) = \frac{1}{2\pi} \cdot \int_{\alpha}^{\beta} e^{-x^2/2} dx =: \Phi_0(\beta) - \Phi_0(\alpha).$

D.h.: Die normierte, zentrierte Binomialverteilung konvergiert mit wachsendem n gegen die Standard-Normalverteilung. Für uns heißt das:

$$\begin{aligned} & \frac{1}{2^n} \cdot S = \text{Prob} (-\sqrt{n} \leq X_{\frac{1}{2},n}^* \leq 1) \\ & \leq \text{Prob} (-\infty < X_{\frac{1}{2},n}^* \leq 0) + \text{Prob} (0 \leq X_{\frac{1}{2},n}^* \leq 1) \approx \frac{1}{2} + 0.3413\dots \leq 0.85 . \end{aligned}$$

□ Lemma 3.26

Literatur

- [1] **K. R. Reischuk:** *Einführung in die Komplexitätstheorie*, Teubner, Stuttgart, 1990
- [2] **W. J. Paul:** *Komplexitätstheorie*, Teubner, Stuttgart, 1978
- [3] **J. E. Hopcroft, J. D. Ullman:** *Introduction to Automata Theory, Languages and Computation*, Addison/Wesley, Reading, 1979
- [4] **M. R. Garey, D.S. Johnson:** *A Guide to the Theory of NP-completeness*, Freeman, San Francisco, 1979
- [5] **H. Hermes:** *Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit*, Springer, Berlin, 1971
- [6] **K. Wagner, G. Wechsung:** *Computational Complexity*, VEB Deutscher Verlag der Wissenschaften, Berlin(Ost), 1986
- [7] **J. E. Savage:** *Models of Computation: Exploring the Power of Computing*, Addison Wesley, 1998
- [8] **B. Marel:** *The Theory of Computation*, Addison Wesley, 1997