Vorlesungsskript

Schaltkreiskomplexität

Sommersemester 2009

Prof. Dr. Johannes Köbler
Humboldt-Universität zu Berlin
Lehrstuhl Komplexität und Kryptografie

18. Mai 2009

Inhaltsverzeichnis

1 Einführung

1.1 Addition von Binärzahlen

$$\begin{aligned} & \mathrm{ADD}^{2n} : \{0,1\}^{2n} \mapsto \{0,1\}^{n+1} \\ \text{mit:} \quad & \mathrm{ADD}^{2n}(a_{n-1}\dots a_0 b_{n-1}\dots b_0) = s_n \dots s_0 \quad , \\ \text{wobei:} \quad & \sum_{i=0}^{n} s_i 2^i = \sum_{i=0}^{n-1} a_i 2^i + \sum_{i=0}^{n-1} b_i 2^i \end{aligned}$$

Für $k \in \mathbb{N}$ sei

$$bin(k) = \begin{cases} 0, & k = 0 \\ a_{n-1}\dots a_0, & k = \sum\limits_{i=0}^{n-1} a_i 2^i > 0, a_{n-1} = 1 \end{cases}$$

Bezeichnen wir die Länge $|bin(k)|$ der Binärdarstellung von k mit $len(k)$, so gilt:

$$len(k) = \begin{cases} 1, & k = 0, \\ \lfloor \log_2 k \rfloor, & k \geq 1. \end{cases}$$

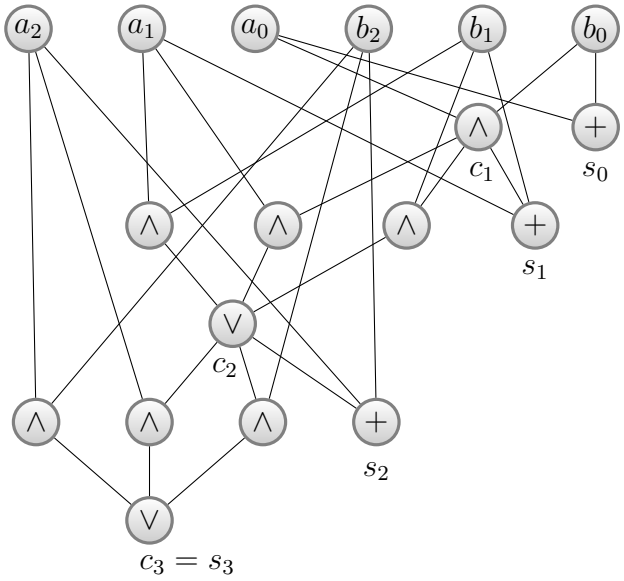
k	$bin(k)$	$\lvert bin(k) \rvert$
0	0	1
1	1	1
2	10	2
3	11	2
4	100	3
5	101	3

1.1.1 Schulmethode

Nach der Schulmethode lassen sich zwei Binärzahlen wie folgt addieren.

$$\begin{aligned} & s_0 = a_0 \oplus b_0, \\ & c_1 = a_0 \wedge b_0, \\ & s_i = a_i \oplus b_i \oplus c_i, && i = 1, \dots, n-1, \\ & c_i = (a_{i-1} \wedge b_{i-1}) \vee (a_{i-1} \wedge c_{i-1}) \vee (b_{i-1} \wedge c_{i-1}), && i = 2, \dots, n, \\ & s_n = c_n. \end{aligned}$$

Beispiel 1. *Für $n = 3$ führt diese Methode auf folgenden Schaltkreis.*



Größe (Anzahl der Gatter):

$$\underbrace{1}_{s_0} + \underbrace{1}_{c_1} + (n-1)(\underbrace{4}_{c_i} + \underbrace{1}_{s_i}) = 5n - 3.$$

Tiefe (maximale Pfadlänge): $2n - 1$.

1.1.2 Carry-lookahead Methode

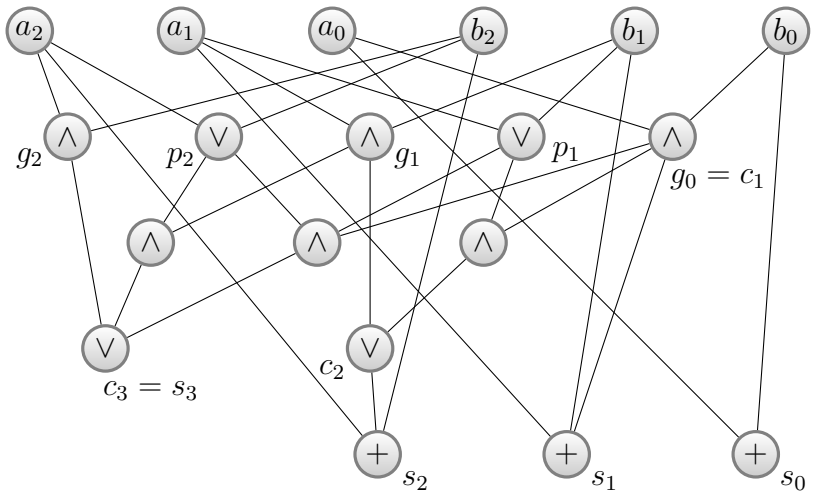
Durch parallele Berechnung der Bits g_i (Position i generiert ein Carry) und p_i (Position i propagiert Carry),

$$\begin{aligned} g_i &= a_i \wedge b_i, && i = 0, \ldots, n-1, \\ p_i &= a_i \vee b_i, && i = 1, \ldots, n-1, \end{aligned}$$

lassen sich die Carry- und Summenbits auch wie folgt berechnen:

$$\begin{aligned} c_i &= \bigvee_{j=0}^{i-1} \Big(g_j \wedge \bigwedge_{k=j+1}^{i-1} p_k \Big), && i = 1, \ldots, n, \\ s_i &= a_i \oplus b_i \oplus c_i, && i = 1, \ldots, n, \\ s_0 &= a_0 \oplus b_0. \end{aligned}$$

Beispiel 2. *Für $n = 3$ erhalten wir nach dieser Methode folgenden Schaltkreis.*



Größe:

$$\underbrace{n}_{g_i} + \underbrace{n-1}_{p_i} + \underbrace{(2+3+\ldots+n)}_{c_i} + \underbrace{n}_{s_i} = \binom{n+1}{2} + 3n - 2 = n^2/2 + \mathcal{O}(n).$$

Tiefe: Da die Tiefe für die Carry-Berechnung durch 3 begrenzt ist, ist die Gesamttiefe ≤ 4.

Definition 3. *Sei $n \in \mathbb{N}$. Eine **n-stellige boolesche Funktion** ist eine Funktion*

$$f : \{0,1\}^n \mapsto \{0,1\}.$$

Wir bezeichnen die Menge aller n-stelligen booleschen Funktionen mit $\mathbb{B}^n$.

Beispiel 4.

- $\neg : \{0,1\} \mapsto \{0,1\}$.
- $\wedge, \vee, \oplus : \{0,1\}^2 \mapsto \{0,1\}$.
- $0, 1 : \{0,1\}^0 \mapsto \{0,1\}$. *"Konstanten"*
- $\bigwedge^m : \{0,1\}^m \mapsto \{0,1\}$ *mit* $\bigwedge^m(a_1 \ldots a_m) = \prod_{i=1}^{m} a_i$.

Definition 5. *Eine **Familie von booleschen Funktionen** ist eine Folge $f = (f^n)_{n\in\mathbb{N}}$, wobei f^n eine n-stellige boolesche Funktion ist. Für $f^n(a_1 \ldots a_n)$ schreiben wir auch einfach $f(a_1 \ldots a_n)$.*

Beispiel 6. $\bigwedge = (\bigwedge^m)_{m\in\mathbb{N}}$.

1.2 Boolesche Schaltkreise

Definition 7. *Eine **Basis** ist eine (i.a. endliche) Menge von booleschen Funktionen und von Familien boolescher Funktionen.*

Beispiel 8.

- $B = \{\wedge^2, \vee^2, \oplus^2, \oplus^3\}$.
- $B_0 = \{0, 1, \neg, \wedge^2, \vee^2\}$ *heißt Standardbasis mit beschränktem Fanin.*
- $B_1 = \{0, 1, \neg, (\wedge^m)_{m\in\mathbb{N}}, (\vee^m)_{m\in\mathbb{N}}\}$ *heißt Standardbasis mit unbeschränktem Fanin.*

Definition 9. *Ein **boolescher Schaltkreis** (kurz SK) über einer Basis B mit n Ein- und m Ausgängen ist ein Tupel $C = (V, E, \alpha, \beta, \omega)$, wobei (V, E) ein endlicher, azyklischer Digraph mit Mehrfachkanten (also ein gerichteter Multigraph) ist und α, β und ω Funktionen der Form*

$$\begin{aligned}\alpha &: E \to \mathbb{N}\\ \beta &: V \to B \cup \{x_1 \dots x_n\}\\ \omega &: V \to \{y_1 \dots y_m\} \cup \{\star\}\end{aligned}$$

mit folgenden Eigenschaften sind.

1. *α ist injektiv (ordnet die eingehenden Kanten eines Gatters).*
2. *Für $i = 1, \dots, n$ existiert höchstens ein Knoten $v \in V$ mit $\beta(v) = x_i$. Diese Knoten werden als **Eingabegatter** bezeichnet.*
3. *Für $i = 1, \dots, m$ existiert genau ein Knoten $v \in V$ mit $\omega(v) = y_i$. Diese Knoten werden als **Ausgabegatter** bezeichnet.*
4. *Für alle Knoten $v \in V$ mit Eingangsgrad (Fanin) 0 ist*

   $$\beta(v) \in \{x_1 \dots x_n\} \cup (B \cap \{0, 1\}),$$

   d.h. v ist ein Eingabegatter oder eine Konstante in B.
5. *Falls $v \in V$ Fanin $k > 0$ hat, so ist $\beta(v)$ eine k-stellige boolesche Funktion oder eine Familie in B.*

Die Kanten $(u, v) \in E$ werden auch als **Drähte** bezeichnet. Ist $(u, v) \in E$, so heißt u **Vorgänger** von v. Jedem Knoten $v \in V$ ordnen wir induktiv eine boolesche Funktion $val_v \in \mathbb{B}^n$ wie folgt zu.

1. Falls v Fanin 0 hat, so ist

   $$val_v(a_1, \dots, a_n) = \begin{cases} a_i, & \beta(v) = x_i,\\ b, & \beta(v) = b \in \{0, 1\}.\end{cases}$$

2. Hat v einen Fanin $k > 0$ und sind $(v_1, v), \dots, (v_k, v)$ die in v mündenden Kanten, geordnet gemäß α (d.h. $\alpha(v_1, v) < \dots < \alpha(v_k, v)$) und ist $\beta(v)$ eine k-stellige Funktion f oder eine Familie von booleschen Funktionen f in B, so ist

   $$val_v(a_1 \dots a_n) = f(val_{v_1}(a_1 \dots a_n), \dots, val_{v_k}(a_1 \dots a_n)).$$

Falls die Basis B nur symmetrische Funktionen und Familien f enthält (d.h. $f(a_1 \dots a_n)$ hängt nur von $a_1 + \dots + a_n$ ab), kann auf die Angabe von α natürlich verzichtet werden.

Seien $v_1, \ldots, v_m$ die Knoten in V mit $\omega(v_i) = y_i$, so **berechnet** C die Funktion

$$f_C : \{0,1\}^n \to \{0,1\}^m$$

mit

$$f_C(a_1, \ldots, a_n) = val_{v_1}(a_1 \ldots a_n) \ldots val_{v_m}(a_1 \ldots a_n).$$

Wir fassen alle Funktionen f der Form $f : \{0,1\}^n \to \{0,1\}^m$ in der Klasse $\mathbb{B}^{n,m}$ zusammen. Die **Größe** eines Schaltkreises C ist

$$size(C) = \|\{v \in V | \beta(v) \in B\}\|$$

und seine **Tiefe** ist

$$depth(C) = \text{die maximale Länge eines gerichteten Pfades in } (V, E).$$

1.3 Schaltkreis-Familien

Eine **Schaltkreis-Familie** über B ist eine Folge $\mathsf{C} = (C_n)_{n \in \mathbb{N}}$ von Schaltkreisen C_n mit n Eingängen. C **berechnet** die Funktion $f_{\mathsf{C}} : \{0,1\}^\star \to \{0,1\}^\star$ mit

$$f_{\mathsf{C}}(a_1 \ldots a_n) = f_{C_n}(a_1 \ldots a_n).$$

Ist f_{C} $\{0,1\}$-wertig (d.h. die Schaltkreise C_n besitzen jeweils genau ein Ausgabegatter), so berechnet C die **charakteristische Funktion** der Sprache

$$L(\mathsf{C}) = \{x \in \{0,1\}^\star | f_{\mathsf{C}}(x) = 1\}.$$

Wir werden im Folgenden die charakteristische Funktion einer Sprache L mit dieser identifizieren.
Seien $s, d : \mathbb{N} \to \mathbb{N}$. C hat die **Größe** s (**Tiefe** d), falls gilt:

$$size(\mathsf{C}_n) = s(n) \quad \text{bzw.} \quad depth(\mathsf{C}_n) = d(n).$$

Hierfür schreiben wir kurz $size(\mathsf{C}) = s$ und $depth(\mathsf{C}) = d$. Mit

$$\mathsf{Size}_B(s) = \left\{ f_{\mathsf{C}} \,\middle|\, \mathsf{C} \text{ ist eine Schaltkreisfamilie über } B \text{ mit } size(\mathsf{C}) = \mathcal{O}(s) \right\},$$

bezeichnen wir die Klasse aller Funktionen $f : \{0,1\}^\star \to \{0,1\}^\star$, für die eine Schaltkreisfamilie C über B der Größe $\mathcal{O}(s)$ existiert, welche f_{C} berechnet. Weiter definieren wir

$$\mathsf{Depth}_B(d) = \left\{ f_{\mathsf{C}} \,\middle|\, \mathsf{C} \text{ ist eine Schaltkreisfamilie über } B \text{ mit } depth(\mathsf{C}) = \mathcal{O}(d) \right\}$$

und

$$\mathsf{Size\text{-}Depth}_B(s,d) = \left\{ f_{\mathsf{C}} \,\middle|\, \begin{array}{c} \mathsf{C} \text{ ist eine Schaltkreisfamilie über } B \text{ mit} \\ size(\mathsf{C}) = \mathcal{O}(s) \text{ und } depth(\mathsf{C}) = \mathcal{O}(d) \end{array} \right\}.$$

Bemerkung 10.

- *Im Fall* $B = B_0 = \{0, 1, \neg, \wedge^2, \vee^2\}$ *lassen wir den Index* B_0 *weg (d.h.* $\mathsf{Size}(s)$ *steht für* $\mathsf{Size}_{B_0}(s)$*).*
- *Im Fall* $B = B_1 = \{0, 1, \neg, (\wedge^n)_{n\in\mathbb{N}}, (\vee^n)_{n\in\mathbb{N}}\}$ *lassen wir den Index ebenfalls weg, benutzen aber das Präfix "*Unb*" (d.h.* $\mathsf{UnbSize}(s)$ *steht für* $\mathsf{Size}_{B_1}(s)$*).*

Definition 11. *Eine Basis, die mindestens eine Familie von booleschen Funktionen enthält, heißt von* ***unbeschränktem Fanin****, andernfalls von* ***beschränktem Fanin****.*

Proposition 12. *Sei* B *von beschränktem Fanin und seien* $s, d : \mathbb{N} \to \mathbb{N}$*. Dann gilt:*

1. $\mathsf{Size\text{-}Depth}_{B\cup B_0}(s, d) = \mathsf{Size\text{-}Depth}(s, d)$,
2. $\mathsf{Size\text{-}Depth}_{B\cup B_1}(s, d) = \mathsf{UnbSize\text{-}Depth}(s, d)$.

Beweis. Ersetze jedes Gatter, das eine Funktion $f \in B$ berechnet, durch einen Schaltkreis C_f über der Basis B_0. ■

Proposition 13. $\mathsf{UnbSize\text{-}Depth}(s, d) \subseteq \mathsf{Size\text{-}Depth}((s + n)s, d \log(s + n))$.

Beweis. Ersetze jedes $\wedge^k$- bzw. $\vee^k$- Gatter, $k \leq s + n$, durch einen Binärbaum von $\wedge^2$- bzw. $\vee^2$- Gattern der Größe k und Tiefe $\log k$. ■

1.4 Die NC-Hierarchie

Neben $\wedge^n$- und $\vee^n$- Gattern spielen auch die Majoritätsgatter MAJ^n eine wichtige Rolle. Diese berechnen die Funktion $\mathrm{MAJ}^n : \{0, 1\}^n \to \{0, 1\}$ mit

$$\mathrm{MAJ}^n(a_1, \ldots, a_n) = \begin{cases} 1, & \sum_{i=1}^n a_i \geq n/2, \\ 0, & \text{sonst.} \end{cases}$$

Zusammen mit den Konstanten $0, 1$ und dem Negationsgatter bilden diese die Basis $B_2 = \{0, 1, \neg, (\mathrm{MAJ}^n)\}$ mit unbeschränktem Fanin.
Für eine Klasse $\mathcal{F}$ von Funktionen $f : \mathbb{N} \to \mathbb{N}$ ist

$$\mathsf{Size}(\mathcal{F}) = \bigcup_{f\in\mathbb{F}} \mathsf{Size}(f)$$

und analog für die anderen Schaltkreisklassen. Die wichtigsten Schaltkreisklassen sind

$$\begin{aligned} \mathsf{NC}^i &= \mathsf{Size\text{-}Depth}(n^{O(1)}, \log^i n), \\ \mathsf{AC}^i &= \mathsf{UnbSize\text{-}Depth}(n^{O(1)}, \log^i n), \\ \mathsf{TC}^i &= \mathsf{Size\text{-}Depth}_{B_2}(n^{O(1)}, \log^i n). \end{aligned}$$

Die Inklusion $\mathsf{NC}^i \subseteq \mathsf{AC}^i$ folgt direkt aus der Definition. Da sich $\wedge^n$- und $\vee^n$-Gatter durch Majoritätsgatter simulieren lassen, d.h. es gilt

$$\wedge^n(x_1, \ldots, x_n) = \mathrm{MAJ}(x_1, \ldots, x_n, \underbrace{0, \ldots, 0}_{(n-1)\text{-mal}})$$

und

$$\vee^n(x_1, \ldots, x_n) = \mathrm{MAJ}(x_1, \ldots, x_n, \underbrace{1, \ldots, 1}_{(n-1)\text{-mal}}),$$

ist AC^i eine Teilklasse von TC^i. Wir werden später sehen, dass TC^i in NC^{i+1} enthalten ist. Daher bilden diese Klassen eine Hierarchie (die so genannte NC-Hierarchie):

$$\mathsf{NC}^0 \subseteq \mathsf{AC}^0 \subseteq \mathsf{TC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{AC}^1 \subseteq \mathsf{TC}^1 \subseteq \mathsf{NC}^2 \subseteq \cdots \subseteq \mathsf{NC},$$

wobei $\mathsf{NC} = \bigcup_{i \geq 0} \mathsf{NC}^i = \bigcup_{i \geq 0} \mathsf{AC}^i = \bigcup_{i \geq 0} \mathsf{TC}^i$ ist.
Wie wir gesehen haben, ist $\mathrm{ADD} \in \mathsf{Size\text{-}Depth}_B(n^2, 1)$ für $B = \{\oplus^2, \oplus^3, (\wedge^n), (\vee^n)\}$. Da sich die Parity-Gatter $\oplus^2$ und $\oplus^3$ durch Schaltkreise konstanter Größe über der Basis B_1 (sogar über B_0) berechnen lassen, folgt $\mathrm{ADD} \in \mathsf{Size\text{-}Depth}_{B_1}(n^2, 1) \subseteq \mathsf{AC}^0$.

1.5 Multiplikation von Binärzahlen

Als nächstes betrachten wir die Multiplikation von Binärzahlen.

Definition 14. $\mathrm{MULT}^{2n} : \{0,1\}^{2n} \to \{0,1\}^{2n}$ *mit*

$$\mathrm{MULT}(a_{n-1} \ldots a_0 b_{n-1} \ldots b_0) = d_{2n-1} \ldots d_0,$$

wobei

$$\sum_{i=0}^{2n-1} d_i \cdot 2^i = \left(\sum_{i=0}^{n-1} a_i 2^i\right)\left(\sum_{i=0}^{n-1} b_i 2^i\right).$$

Mithilfe der Schulmethode lässt sich die Multiplikation von 2 n-Bit Zahlen auf die Addition von n n-Bit Zahlen zurückführen.

Definition 15. $\mathrm{ITADD}^{n^2} : \{0,1\}^{n^2} \to \{0,1\}^{2n}$ *mit*

$$\mathrm{ITADD}(a_{1,n-1} \ldots a_{1,0} \ldots a_{n,n-1} \ldots a_{n,0}) = s_{2n-1} \ldots s_0,$$

wobei

$$\sum_{i=0}^{2n-1} s_i 2^i = \sum_{i=1}^{n} \sum_{j=0}^{n-1} a_{i,j} \cdot 2^j.$$

Addieren wir die Summanden baumartig auf, so erhalten wir einen AC^1 Schaltkreis für ITADD. Tatsächlich können wir ITADD sogar in NC^1 berechnen (wir werden später sehen, dass es sogar in TC^0 geht).

Wir können die Addition von 3 Binärzahlen $a, b, c \in \{0,1\}^n$ auf die Addition von 2 Binärzahlen $d, e \in \{0,1\}^{n+1}$ wie folgt reduzieren:

$$d_i = \begin{cases} a_i \oplus b_i \oplus c_i & i = 0, \ldots, n-1 \\ 0 & i = n \end{cases}$$

$$e_i = \begin{cases} 0 & i = 0 \\ (a_{i-1} \wedge b_{i-1}) \vee (a_{i-1} \wedge c_{i-1}) \vee (b_{i-1} \wedge c_{i-1}) & i = 0, \ldots, n-1 \end{cases}$$

$$\begin{array}{cccccccc} & a_{n-1} & \ldots & a_{i+1} & a_i & \ldots & a_0 \\ & b_{n-1} & \ldots & b_{i+1} & b_i & \ldots & b_0 \\ + & c_{n-1} & \ldots & c_{i+1} & c_i & \ldots & c_0 \\ \hline d_n & d_{n-1} & \ldots & d_{i+1} & d_i & \ldots & d_0 \\ e_n & e_{n-1} & \ldots & e_{i+1} & e_i & \ldots & e_0 \end{array}$$

Nun ist leicht zu sehen, dass $a_i + b_i + c_i = d_i + 2e_{i+1}$ und daher $a + b + c = d + e$ ist.

Sei $R(a, b, c) = (d, e)$ ein Schaltkreis, der diese Funktion berechnet, dann hat R die Größe $O(n)$ und die Tiefe $O(1)$ über der Basis B_0. Durch eine Schicht von R-Schaltkreisen können wir die Addition von $m \geq 3$ k-Bit Zahlen auf die Addition von m' $(k+1)$-bit Zahlen reduzieren, wobei

$$m' \leq \frac{2}{3}(m-2) + 2 \leq \frac{8}{9}m$$

ist. Daher reichen $O(\log n)$ Schichten aus, um die Addition von n n-Bit Zahlen auf die Addition von 2 $(n+O(\log n))$-Bit Zahlen zu reduzieren. Diese lassen sich durch einen ADD-Schaltkreis über B_0 der Größe $(n+O(\log n))^{O(1)} = n^{O(1)}$ und Tiefe $O(\log(n+O(\log n))) = O(\log n)$ addieren. Dies zeigt ITADD $\in \mathsf{NC}^1$.

Ein Spezialfall von ITADD ist die Addition von n Bits. Hierzu betrachten wir die Funktion $\text{BCount}^n : \{0,1\}^n \to \{0,1\}^{len(n)}$ mit

$$\text{BCount}^n(a_1, \ldots, a_n) = b_{len(n)-1} \cdots b_0,$$

wobei $\sum_{i=1}^{n} a_i = \sum_{j=0}^{len(n)-1} b_j 2^j$ ist. Wir werden sehen, dass BCount und ITADD vergleichbare Schaltkreiskomplexität haben (formal: ITADD ist in konstanter Tiefe auf BCount reduzierbar; siehe Abschnitt 2).

Auch die Majoritätsfunktion MAJ hat ähnliche Komplexität. Um $\text{MAJ}(a_1 \ldots a_n)$ in NC^1 zu berechnen, berechnet man zuerst $b = \text{BCount}^n(a_1, \ldots, a_n)$ und vergleicht dann diesen Wert mit dem Wert $m = \lceil n/2 \rceil$. Die Ausgabe ist genau dann 1, wenn $b \geq m$ ist. Da die Funktion $\text{LEQ}^{2n} : \{0,1\}^{2n} \to \{0,1\}$ mit

$$\text{LEQ}^{2n}(a_{n-1} \cdots a_0 b_{n-1} \cdots b_0) = \begin{cases} 1, & \sum_{i=0}^{n-1} a_i 2^i \leq \sum_{i=0}^{n-1} b_i 2^i, \\ 0, & \text{sonst} \end{cases}$$

in AC^0 berechenbar ist (siehe Übungen), folgt MAJ $\in \mathsf{NC}^1$ und somit $\mathsf{TC}^i \subseteq \mathsf{NC}^{i+1}$.

Korollar 16.

1. MULT $\in \mathsf{NC}^1$,
2. BCount $\in \mathsf{NC}^1$,
3. MAJ $\in \mathsf{NC}^1$.

Korollar 17. *Für $i \geq 0$ gilt $\mathsf{TC}^i \subseteq \mathsf{NC}^{i+1}$.*

1.6 Addition von logarithmisch vielen Binärzahlen

Zwei n-Bit-Zahlen können also in AC^0 und n n-Bit-Zahlen in NC^1 addiert werden. In welcher Tiefe lassen sich logarithmisch viele n-Bit-Zahlen mit polynomiell vielen Gattern addieren? Hierzu betrachten wir die Funktion LOGITADD : $\{0,1\}^{nlen(n)} \to \{0,1\}^{n+len(n)}$, die bei Eingabe von $len(n)$ n-Bit Zahlen die Binärdarstellung der Summe berechnet.

Wenden wir die zum Nachweis von ITADD $\in \mathsf{NC}^1$ verwendete Technik an, so erhalten wir einen Schaltkreis der Tiefe $O(\log\log n)$ für LOGITADD. Tatsächlich lässt sich LOGITADD in AC^0 berechnen.

Satz 18. LOGITADD $\in \mathsf{AC}^0$.

Beweis. Gegeben sind $len(n)$ Binärzahlen $a_1, \ldots, a_{len(n)}$, wobei $a_i = a_{i,n-1}, \ldots, a_{i,0}$ für $i = 1, \ldots, len(n)$ ist. Für $k = 0, \ldots, n-1$ berechnen wir die Spaltensumme $s_k = \sum_{i=1}^{len(n)} a_{i,k}$ in Binärdarstellung. Dann hat jedes s_k die Länge $len(len(n)) = len^{(2)}(n)$ und es gilt $s_k = \sum_{i=1}^{len(n)} a_{i,k} = \sum_{j=0}^{len^{(2)}(n)-1} s_{k,j}2^j$. Nun folgt

$$\begin{aligned}\sum_{i=1}^{len(n)} a_i &= \sum_{i=1}^{len(n)} \sum_{k=0}^{n-1} a_{i,k}2^k = \sum_{k=0}^{n-1} \sum_{i=1}^{len(n)} a_{i,k}2^k = \sum_{k=0}^{n-1} s_k 2^k = \sum_{k=0}^{n-1} \sum_{j=0}^{len^{(2)}(n)-1} s_{k,j}2^j 2^k \\ &= \sum_{j=0}^{len^{(2)}(n)-1} \sum_{k=0}^{n-1} s_{k,j}2^{j+k}\end{aligned}$$

Es genügt also, die $len^{(2)}(n)$ Zahlen $s_j^2 = \sum_{k=0}^{n-1} s_{k,j}2^{j+k}$ mit je $n + len^{(2)}(n)$ Bits zu addieren. Man beachte, dass jedes Bit dieser Zahlen nur von $len(n)$ Eingabebits abhängt und daher in Tiefe 3 mit $\mathcal{O}(2^{len(n)}) = \mathcal{O}(n)$ Gattern berechenbar ist.

Wiederholen wir obiges Verfahren, so erhalten wir $len^{(3)}(n)$ Zahlen s_j^3 mit $n + len^{(2)}(n) + len^{(3)}(n)$ Bits usw. bis 2 Zahlen s_j^k mit $n + len^{(2)}(n) + \ldots len^{(k)}(n)$ Bits übrig bleiben, wobei $k = \min\{l \geq 0 \mid len^{(l)}(n) \leq 2\}$ ist. Da $n + len^{(2)}(n) + \ldots len^{(k)}(n) = n + o(\log n)$ ist (siehe Übungen), lassen sich die beiden Zahlen s_0^k und s_1^k in AC^0 addieren.

Um zu zeigen, dass sich s_0^k und s_1^k aus den $len^{(2)}(n)$ Zahlen $s_j^2 = \sum_{k=0}^{n-1} s_{k,j}2^{j+k}$, $j = 1, \ldots, len^{(2)}(n)$, in konstanter Tiefe berechnen lassen, machen wir folgende Beobachtung: Für $i = 3, \ldots, k$ hängt jedes Bit der Zahlen s_j^i, $j = 1, \ldots, len^{(i)}(n)$, in Stufe i nur von $len^{(i-1)}(n)$ Bits der Zahlen s_j^{i-1}, $j = 1, \ldots, len^{(i-1)}(n)$, in Stufe $i-1$ ab. Folglich hängt

jedes Bit der Zahlen s_0^k und s_1^k nur von $len^{(k-1)}(n) \cdot len^{(k-2)}(n) \cdot \cdots \cdot len^{(2)}(n)$ Bits der Zahlen s_j^2, $j = 1, \ldots, len^{(2)}(n)$, ab. Da $len^{(k-1)}(n) \cdot len^{(k-2)}(n) \cdot \cdots \cdot len^{(2)}(n) = \mathcal{O}(\log n)$ ist (siehe Übungen), lassen sich s_0^k und s_1^k aus den Zahlen $s_j^2 = \sum_{k=0}^{n-1} s_{k,j} 2^{j+k}$, $j = 1, \ldots, len^{(2)}(n)$, in AC^0 berechnen. ■

Die Funktion LOGITADD wird sich noch bei der Berechnung von ITADD (und damit auch von MULT und BCount) in TC^0 als nützlich erweisen.

2 AC^0-Reduktionen

In diesem Abschnitt führen wir den Begriff der AC^0-Reduktion, den wir im Beweis von Korollar 16 bereits implizit benutzt haben, formal ein.

Definition 19.

1. *Eine Funktion $g : \{0,1\}^* \to \{0,1\}^+$ heißt **längenverträglich**, falls $|g(x)| = r(|x|)$ für eine Funktion $r : \mathbb{N} \to \mathbb{N}^+$ ist.*
2. *Für eine längenverträgliche Funktion g sei $bits(g) = \{g_i^n \mid n \geq 0, i = 1, \ldots, r(n)\}$ die Menge der booleschen Funktionen g_i^n, die das i-te Bit von $g^n(x)$ berechnen, d.h. im Falle $g^n(x) = a_1 \ldots a_{r(n)}$ ist $g_i^n(x) = a_i$ für $i = 1, \ldots, r(n)$. Im Fall $r(n) = 1$ (d.h. g ist eine Familie von booleschen Funktionen) gilt also $bits(g) = \{g^n \mid n \geq 0\}$.*
3. *Entsprechend definieren wir für eine Menge $\mathcal{F}$ von längenverträglichen Funktionen die Menge $bits(\mathcal{F}) = \bigcup_{f \in \mathcal{F}} bits(f)$.*
4. *$\mathcal{F}$ heißt auf $\mathcal{G}$ **in konstanter Tiefe reduzierbar** (kurz: $\mathcal{F} \leq_{cd} \mathcal{G}$), falls es Konstanten c und d gibt, so dass für alle $f \in \mathcal{F}$ gilt:*

   f^n ist durch einen Schaltkreis über der Basis $B_1 \cup bits(\mathcal{G})$ der Größe $n^c + c$ und Tiefe d berechenbar.

   Hierbei trägt jedes Gatter aus $bits(\mathcal{G})$ mit Fanin k genau k zur Größe des Schaltkreises für f^n bei.

Gilt $\mathcal{F} \leq_{cd} \mathcal{G}$ und $\mathcal{G} \leq_{cd} \mathcal{F}$, so schreiben wir $\mathcal{F} \equiv_{cd} \mathcal{G}$. Bestehen $\mathcal{F}$ und $\mathcal{G}$ jeweils nur aus einer längenverträglichen Funktion f bzw. g, so schreiben wir auch einfach $f \leq_{cd} g$ bzw. $f \equiv_{cd} g$.
Für $f \leq_{cd} g$ wird oft auch $f \leq_{\mathsf{AC}^0} g$ oder $f \in \mathsf{AC}^0(g)$ geschrieben.

Proposition 20.

1. $\text{MAJ} \leq_{cd} \text{BCount} \leq_{cd} \text{ITADD}$,
2. $\text{MULT} \leq_{cd} \text{ITADD}$.

Lemma 21. *Seien f, g, h längenverträgliche und polynomiell längenbeschränkte (d.h. $|f(x)| = |x|^{O(1)}$) Funktionen. Dann gilt:*

i) $f \leq_{cd} f$, *d.h.* $\leq_{cd}$ *ist reflexiv.*

ii) $f \leq_{cd} g$ *und* $g \leq_{cd} h \Rightarrow f \leq_{cd} h$, *d.h.* $\leq_{cd}$ *ist transitiv.*

iii) Für $i \geq 0$ *gilt:* $f \leq_{cd} g$, $g \in \mathsf{AC}^i \Rightarrow f \in \mathsf{AC}^i$.

iv) Für $i \geq 0$ *gilt:* $f \leq_{cd} g$, $g \in \mathsf{TC}^i \Rightarrow f \in \mathsf{TC}^i$.

v) Für $i \geq 1$ *gilt:* $f \leq_{cd} g$, $g \in \mathsf{NC}^i \Rightarrow f \in \mathsf{NC}^i$.

vi) $f \in \mathsf{AC}^0 \Rightarrow f \leq_{cd} g$.

2.1 Addieren von Bits

Um BCount $\leq_{cd}$ MAJ zu zeigen, betrachten wir als Verallgemeinerung von Majoritätsgattern so genannte Threshold-Gatter

$$T_m^n(a_1 \dots a_n) = \begin{cases} 1, & \sum a_i \geq m, \\ 0, & \text{sonst.} \end{cases}$$

Lemma 22. $\{T_m \mid m \in \mathbb{N}\} \leq_{cd}$ MAJ.

Beweis. Wir betrachten zuerst den Fall $n/2 \leq m \leq n$. Durch Anfügen von $2m-n$ Nullen erhalten wir

$$T_m^n(a_1 \dots a_n) = \mathrm{MAJ}^{2m}(a_1 \dots a_n \underbrace{0 \dots 0}_{2m-n}).$$

Die Größe des Reduktionsschaltkreises ist also $2m - n$ (Anzahl der Nullen) plus $2m$ (Fanin des Majority-Orakelgatters) gleich $4m - n \leq 3n$. Im Fall $1 \leq m < n/2$ fügen wir $n - 2m$ Einsen an:

$$T_m^n(a_1 \dots a_n) = \mathrm{MAJ}^{2(n-m)}(a_1 \dots a_n \underbrace{1 \dots 1}_{n-2m}).$$

Die Größe der Reduktionsschaltkreise ist nun $n - 2m$ (Anzahl der Einsen) plus $2(n - m)$ (Fanin des Majority-Orakelgatters) gleich $3(n - m) \leq 3n$. ■

Satz 23. BCount $\leq_{cd}$ MAJ.

Beweis. Unter Verwendung des vorigen Lemmas genügt es, BCount $\leq_{cd} \{T_m \mid m \geq 1\}$ zu zeigen. Sei $s_{l-1} \dots s_0$ die Binärdarstellung von $s = \sum_{i=1}^{n} a_i$. Dann gilt

$$\begin{aligned} s_0 = 1 &\Leftrightarrow s \text{ ist ungerade} \\ &\Leftrightarrow \text{ es existiert ein } k \in \{0, \dots, n\},\ k \text{ ungerade mit } \sum a_i = k \\ &\Leftrightarrow \bigvee_{k \in S_0^n} T_k^n(a_1 \dots a_n) \wedge \neg T_{k+1}^n(a_1 \dots a_n), \end{aligned}$$

wobei S_0^n alle ungeraden $k \in \{0, \dots, n\}$ enthält. Allgemeiner ist $s_j = 1$, wenn ein $k \in S_j^n$ ex. mit $\sum a_i = k$, wobei

$$S_j^n = \{k \leq n \mid \mathit{bin}(k) \text{ hat an der } j\text{-ten Stelle eine Eins}\}$$

ist. ■

Korollar 24. BCount $\in \mathsf{TC}^0$.

2.2 Multiplikation von Binärzahlen

Als nächstes zeigen wir die Reduktion ITADD $\leq_{cd}$ BCount und damit MULT $\in \mathsf{TC}^0$.

Satz 25. ITADD $\leq_{cd}$ BCount.

Beweis. Um die Summe s von n n-Bit-Zahlen $a_i = a_{i,n-1} \dots a_{i,0}, 1 \leq i \leq n$, zu berechnen, berechnen wir zunächst die Spaltensummen $s_k = \sum_{i=1}^{n} a_{i,k}$, $k = 0, \dots, n-1$, mit BCount-Gattern. Die Bits dieser Zahlen lassen sich wie im Beweis von Satz 18 auf $len(n)$ Zahlen mit je $n + len(n)$ Bits verteilen, deren Summe s ergibt. Nach Satz 18 können diese in AC^0 aufaddiert werden. ■

Korollar 26. MULT, ITADD $\in \mathsf{TC}^0$.

Umgekehrt lässt sich BCount auch auf MULT reduzieren.

Satz 27. BCount $\leq_{cd}$ MULT.

Beweis. Um die Binärdarstellung der Summe s von n Bits $a_{n-1}, \dots, a_0$ zu erhalten, setzen wir $k = len(n)$ und multiplizieren die Zahlen $u = \sum_{i=0}^{n-1} a_i 2^{ik}$ und $v = \sum_{i=0}^{n-1} 2^{ik}$, die die Binärdarstellungen

$$a_{n-1} \underbrace{0 \cdots 0}_{(k-1)\text{-mal}} \cdots a_0 \underbrace{0 \cdots 0}_{(k-1)\text{-mal}} \quad \text{und} \quad 1 \underbrace{0 \cdots 0}_{(k-1)\text{-mal}} \cdots 1 \underbrace{0 \cdots 0}_{(k-1)\text{-mal}}$$

haben. Als Produkt w von u und v erhalten wir die Zahl $w = \sum_{i=0}^{2n-2} c_i \cdot 2^{k \cdot i}$, wobei c_i für $i = 0, \dots, n-1$ den Wert $c_i = a_0 + \cdots + a_i$ und für $i = n, \dots, 2n-2$ den Wert $c_i = a_{i-n+1} + \cdots + a_{n-1}$ hat. Folglich ist $c_{n-1} = s$ und da die Werte $c_i \leq s < 2^k$ sind, können wir die Binärdarstellung von c_{n-1} als Substring der Binärdarstellung von w an den Stellen $k(n-1), \dots, kn-1$ ablesen. ■

Korollar 28. MAJ $\equiv_m$ MULT $\equiv_m$ ITADD $\equiv_m$ BCount.

In den Übungen werden wir sehen, dass sich die Ganzzahl-Division auf die iterierte Multiplikation ITMULT$^{n^2} : \{0,1\}^{n^2} \to \{0,1\}^{n^2}$ zurückführen lässt, die wie folgt definiert ist. ITMULT$(a_{1,n-1} \dots a_{1,0} \dots a_{n,n-1} \dots a_{n,0}) = c_{n^2-1} \dots c_0$, wobei

$$\sum_{i=0}^{n^2-1} c_i 2^i = \prod_{i=1}^{n} \left(\sum_{j=0}^{n-1} a_{i,j} \cdot 2^j \right)$$

ist. Um ITMULT auf MAJ zu reduzieren, benutzen wir den chinesischen Restsatz (CRS), welcher besagt, dass ein lineares Kongruenzgleichungssystem

$$\begin{aligned} x &\equiv_{m_1} b_1 \\ &\vdots \\ x &\equiv_{m_k} b_k \end{aligned} \tag{2.1}$$

für beliebige ganze Zahlen $b_1, \ldots, b_k$ genau eine Lösung x modulo $m = \prod_{i=1}^{k} m_i$ hat, falls die Module $m_1, \ldots, m_k$ paarweise teilerfremd sind. Eine alternative Formulierung des CRS ist, dass die Funktion

$$f : \mathbb{Z}_m \to \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_k}$$

mit $f(x) = (x \bmod m_1, \ldots, x \bmod m_k)$ ein Isomorphismus zwischen den Ringen $\mathbb{Z}_m$ und $\mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_k}$ ist.
Tatsächlich lässt sich für das Gleichungssystem (2.2) eine Lösung x wie folgt berechnen. Da die Zahlen $n_i = {}^{m}/_{m_i}$ teilerfremd zu m_i sind, existieren Zahlen u_i und v_i mit

$$u_i n_i + v_i m_i = \mathrm{ggT}(n_i, m_i) = 1,$$

welche sich mit dem erweiterten Euklidschen Algorithmus berechnen lassen. Dann gilt

$$u_i n_i \equiv_{m_i} 1$$

und

$$u_i n_i \equiv_{m_j} 0$$

für $j \neq i$. Folglich erfüllt $x = \sum_{j=1}^{k} y_j b_j \bmod m$ mit $y_j = u_j n_j \bmod m$ die Kongruenzen

$$x \equiv_{m_i} u_i n_i b_i \equiv_{m_i} b_i$$

für $i = 1, \ldots, k$. Dies zeigt, dass (2.2) für alle $(b_1, \ldots, b_k)$ lösbar, also die Funktion f eine Surjektion ist. Da der Definitions- und der Wertebereich von f die gleiche Mächtigkeit (nämlich m) haben, muss f dann aber auch injektiv sein, d.h. (2.2) ist sogar eindeutig lösbar.

Satz 29. ITMULT $\leq_{cd}$ MAJ.

Beweis. Gegeben sind n Binärzahlen $a_1, \ldots, a_n$, wobei $a_i = a_{i,n-1}, \ldots, a_{i,0}$ für $i = 1, \ldots, n$ ist. Das Produkt dieser Zahlen sei c mit der Binärdarstellung $c_{n^2-1} \ldots c_0$. Seien $p_1 = 2, p_2 = 3, \ldots p_k$ die ersten k Primzahlen, wobei k so groß gewählt wird, dass $2^{n^2} \leq \prod_{j=1}^{k} p_j$ ist. Aus der Zahlentheorie ist bekannt, dass $p_k = \mathcal{O}(n^2)$ ist. Setzen wir $m = \prod_{j=1}^{k} p_j$, so folgt $c < m$ und $m = \mathcal{O}(n^2 2^{n^2})$. Wir berechnen c mit Hilfe des Chinesischen Restsatzes, indem wir im 1. Schritt die Zahlen

$$b_{i,j} = a_i \bmod p_j$$

für $i = 1, \ldots, n$ und $j = 1, \ldots, k$, im 2. Schritt die modularen Produkte

$$b_j = \Big(\prod_{i=1}^{n} b_{i,j} \Big) \bmod p_j$$

für $j = 1, \ldots, k$ und im 3. Schritt die eindeutige Lösung $c \in \{0, \ldots, m-1\}$ des linearen Kongruenzgleichungssystems

$$\begin{aligned} x &\equiv_{p_1} b_1 \\ &\vdots \\ x &\equiv_{p_k} b_k \end{aligned} \tag{2.2}$$

berechnen.

Zu Schritt 1: Berechnung von $b_{i,j} = a_i \bmod p_j$. Es gilt $b_{i,j} = s_{i,j} \bmod p_j$, wobei

$$s_{i,j} = \sum_{l=1}^{n-1} a_{i,l}(2^l \bmod p_j) < np_j \leq np_k = \mathcal{O}(n^3)$$

ist. Die Binärdarstellung der $s_{i,j}$ lässt sich mit je einem ITADD-Orakelgatter in konstanter Tiefe berechnen (man beachte, dass $a_{i,l} \in \{0,1\}$ sind und dass die Zahlen $2^l \bmod p_j$ nur von n und nicht von der Eingabe abhängen und daher als Konstanten fest verdrahtet werden können). Da die $s_{i,j}$ nur aus $\mathcal{O}(\log n)$ Bits bestehen, lassen sich hieraus die $b_{i,j}$ durch Schaltkreise konstanter Tiefe und exponentieller Größe in $\log n$ (also linearer Größe in n) berechnen.

Zu Schritt 2: Berechnung von $b_j = \left(\prod_{i=1}^n b_{i,j}\right) \bmod p_j$. Da die multiplikative Gruppe $\mathbb{Z}_{p_j}^*$ zyklisch ist (hier benutzen wir, dass p_j prim ist), existiert ein Erzeuger $g_j \in \mathbb{Z}_{p_j}^*$ mit $\mathbb{Z}_{p_j}^* = \{g_j^e \bmod p_j \mid e = 0, \ldots, p_j - 2\}$. Daher gilt

$$b_i = g_j^{(\sum_{i=1}^n e_{i,j}) \bmod p_j - 1} \bmod p_j,$$

wobei die Zahlen $e_{i,j} \in \{0, \ldots, p_j - 2\}$ so gewählt sind, dass $g_j^{e_{i,j}} \bmod p_j = b_{i,j}$ ist. Da die Bits der $e_{i,j}$ nur von $b_{i,j}$, also nur von $\mathcal{O}(\log n)$ bereits berechneten Bits abhängen, lassen sie sich in konstanter Tiefe und linearer Größe (in n) berechnen. Aus den Zahlen $e_{i,j}$ lassen sich die Summen $\sum_{i=1}^n e_{i,j}$ durch je ein ITADD-Gatter berechnen, und da diese nur aus $\mathcal{O}(\log n)$ Bits bestehen, lassen sich hieraus die Zahlen $b_i = g_j^{(\sum_{i=1}^n e_{i,j}) \bmod p_j - 1} \bmod p_j$ in Tiefe $\mathcal{O}(1)$ und Größe $\mathcal{O}(n)$ berechnen.

Zu Schritt 3: Berechnung von $c = f^{-1}(b_1, \ldots, b_k)$. Aus dem Beweis des CRS folgt $c = (\sum_{j=1}^k b_j y_j \bmod m) \bmod m$. Da die Bits der Summanden $z_j = b_j y_j \bmod m$ nur von b_j, also nur $\mathcal{O}(\log n)$ Bits abhängen (die y_j sind Konstanten), lassen sich diese in Tiefe $\mathcal{O}(1)$ und Größe $\mathcal{O}(n)$ berechnen. Wegen $z_j < m$ ist deren Länge durch $\mathcal{O}(n^2)$ beschränkt und daher lässt sich $c' = \sum_{j=1}^k z_j$ durch ein ITADD-Gatter bestimmen. Um schließlich aus c' den Wert von c zu erhalten, bestimmen wir zuerst das größte $d \in \{0, \ldots, k\}$ mit $dm \leq c'$ mithilfe von MULT-Orakelgattern und berechnen $c = c' - dm$. ■

2.3 Sortieren von Binärzahlen

Die Funktion SORT sortiert n Zahlen $a_1, \ldots, a_n$ mit je n Bits in nicht-absteigender Folge, d.h., $\text{SORT} : \{0,1\}^{n^2} \to \{0,1\}^{n^2}$ mit $\text{SORT}(a_{1,n-1} \ldots a_{1,0} \ldots a_{n,n-1} \ldots a_{n,0}) = (a'_{1,n-1} \ldots a'_{1,0} \ldots a'_{n,n-1} \ldots a'_{n,0})$, wobei für $i = 1, \ldots, n-1$

$$\sum_{j=0}^{n-1} a'_{i,j} \cdot 2^j \leq \sum_{j=0}^{n-1} a'_{i+1,j} \cdot 2^j$$

ist und die beiden Multimengen

$$\{\{a'_{1,n-1} \ldots a'_{1,0}, \ldots, a'_{n,n-1} \ldots a'_{n,0}\}\} = \{\{a_{1,n-1} \ldots a_{1,0}, \ldots, a_{n,n-1} \ldots a_{n,0}\}\}$$

gleich sind. Wir betrachten zunächst das Problem, die Funktion UCount : $\{0,1\}^n \to \{0,1\}^n$ mit $\text{UCount}(a_1,\ldots,a_n) = \text{un}_n(\sum_{i=1}^n a_i)$ zu berechnen, wobei

$$\text{un}_n(k) = \underbrace{1\ldots1}_{k}\underbrace{0\ldots0}_{n-k} = 1^k0^{n-k}$$

die n-stellige **unäre Repräsentation** von $k \leq n$ ist. UCount sortiert also einzelne Bits in absteigender Folge.

Lemma 30. UCount $\leq_{cd}$ BCount.

Beweis. Sei $\text{UCount}(a_1,\ldots,a_n) = b_1b_2\ldots b_n$. Dann gilt $b_i = 1 \Leftrightarrow \sum_{j=1}^n a_j \geq i$, wobei sich die Summe $\sum_{j=1}^n a_j$ mit einem BCount-Gatter berechnen lässt. ■

Auch die umgekehrte Reduktion lässt sich leicht zeigen.

Lemma 31. BCount $\leq_{cd}$ UCount.

Beweis. Sei $\text{UCount}(a_1,\ldots,a_n) = b_1b_2\ldots b_n$. Zudem sei $b_0 = 1$ und $b_{n+1} = 0$, also $b_0b_1\cdots_n b_{n+1} \in 1^+0^+$. Definiere $d_j = b_j \wedge \neg b_{j+1}$ für $0 \leq j \leq n$. Dann gilt $d_j = 1 \Leftrightarrow \sum a_i = j$. Unter Verwendung der Mengen $S_j^n = \{k \leq n \mid bin(k) \text{ hat an der } j\text{-ten Stelle eine Eins}\}$ folgt $\sum_{i=1}^n a_i = \sum_{j=0}^{len(n)-1} c_j 2^j$, wobei $c_j = \bigvee_{k\in S_j} d_k$ ist. ■

Satz 32. Sort $\leq_{cd}$ UCount.

Beweis. Um n Binärzahlen $a_i = a_{i,n-1}\ldots a_{i,0}$, $i = 1,\ldots,n$, zu sortieren, berechnen wir für $1 \leq i,j \leq n$ die Bits $c_{i,j} = (a_i < a_j) \vee (a_i = a_j \wedge i \leq j)$ und hieraus die Unärzahlen $c'_j = \text{UCount}(c_{1,j},\ldots,c_{n,j})$. Dann gilt $c'_k = c'_{k,1}\ldots c'_{k,n} = \text{un}_n(i)$, wobei i die Position von a_k in der *sortierten* Folge ist. Sei $a'_1,\ldots,a'_n$ die sortierte Folge mit $a'_i = a'_{i,n-1}\ldots a'_{i,0}$. Dann gilt

$$a'_{i,j} = \bigvee_{k=1}^{n} \underbrace{c'_{k,i} \wedge \neg c'_{k,i+1}}_{c'_k = 1^i0^{n-i} = \text{un}_n(i)} \wedge a_{k,j}.$$

■

Lemma 33. UCount $\leq_{cd}$ Sort.

Beweis. Um für eine Bitfolge $a_1,\ldots,a_n$ die Unärdarstellung von $\sum_{i=1}^n a_i$ mithilfe eines Sort-Orakelgatters zu berechnen, stellen wir die Orakelfrage $a_1\underbrace{0\ldots0}_{n-1}\cdots a_n\underbrace{0\ldots0}_{n-1}$. Als Ergebnis erhalten wir die sortierte Zahlenfolge $a'_1\underbrace{0\ldots0}_{n-1}\cdots a'_n\underbrace{0\ldots0}_{n-1}$. Dann ist $\text{un}_n(\sum a_i) = a'_na'_{n-1}\ldots a'_1$. ■

Korollar 34. Sort $\equiv_{cd}$ UCount $\equiv_{cd}$ MAJ.

2.4 Obere Schranken für beliebige boolesche Funktionen

In diesem Abschnitt beweisen wir ein Resultat von Lupanov, dass jede boolesche Funktion $f \in \mathbb{B}^n$ von einem Schaltkreis der Größe $(1 + o(1))2^n/n$ über der Basis $\{0, 1, \oplus, \wedge\}$ berechnet werden kann. Für den Beweis benötigen wir folgende Darstellung von booleschen Funktionen.

Satz 35 (Ringsummenexpansion)**.** *Jede boolesche Funktion $f \in \mathbb{B}^n$ lässt sich schreiben als*

$$f(x_1, \ldots, x_n) = \bigoplus_{a \in \{0,1\}^n} \alpha_a \wedge x_1^{a_1} \wedge \cdots \wedge x_n^{a_n} \quad \textit{mit } a = a_1 \ldots a_n \textit{ und } \alpha_a \in \{0, 1\},$$

wobei

$$x_i^{a_i} = \begin{cases} x_i, & a_i = 1, \\ 1, & a_i = 0 \end{cases}$$

ist. Der Vektor $(\alpha_a)_{a \in \{0,1\}^n}$ ist dabei durch f eindeutig bestimmt.

Beweis. Wir stellen f zunächst in DNF, genauer als Disjunktion $C_1 \vee \cdots \vee C_j$ von vollständigen Konjunktionen der Form $C_i = (\neg)x_1 \wedge \cdots \wedge (\neg)x_n$ über den Literalen $x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n$ dar. Da für jede Belegung höchstens eine Konjunktion C_i wahr wird, können wir f auch in der Form $f(x_1, \ldots, x_n) = C_1 \oplus \cdots \oplus C_j$ darstellen. Ersetzen wir nun die negativen Literale $\neg x_i$ in allen Konjunktionen durch $x_i \oplus 1$ und multiplizieren die erhaltene Formel under Anwendung der Distributivität $(u \oplus v) \wedge w \equiv (u \wedge w) \oplus (v \wedge w)$ aus, so erhalten wir eine Summe von monotonen Konjunktionen, die sich unter Anwendung von $t \oplus t \equiv 0$ auf die angegebene Form vereinfachen lässt.

Dies zeigt, dass sich jede Funktion in dieser Form darstellen lässt. Da andererseits nur 2^{2^n} verschiedene Vektoren (α_a) existieren, kann es für jede der 2^{2^n} n-stelligen booleschen Funktionen $f \in \mathbb{B}^n$ auch nur eine solche Darstellung geben. ■

Die Ringsummenexpansion lässt sich leicht wie folgt verallgemeinern: Für jedes $k \in \{0, \ldots, n\}$ lässt sich jede boolesche Funktion $f \in \mathbb{B}^n$ schreiben als

$$f(x_1, \ldots, x_n) = \bigoplus_{a \in \{0,1\}^{n-k}} h_a(x_1, \ldots, x_k) \wedge x_{k+1}^{a_{k+1}} \wedge \cdots \wedge x_n^{a_n} \quad \text{mit } a = a_{k+1} \ldots a_n,$$

wobei die Funktionen $h_a \in \mathbb{B}^k$ durch f eindeutig bestimmt sind.

Für eine boolesche Funktion $f \in \mathbb{B}^n$ und eine Basis B bezeichne $S_B(f)$ die minimale Größe eines Schaltkreises über B, der f berechnet. Entsprechend bezeichne $S_B(\mathcal{F})$ für eine Menge $\mathcal{F} \subseteq \mathbb{B}^n$ die minimale Größe eines Schaltkreises C über B, der alle Funktionen $f \in \mathcal{F}$ berechnet, d.h. $\mathcal{F} \subseteq \{val_v \mid v \in V(C)\}$.

Satz 36 (Lupanov)**.** *Jede Funktion $f \in \mathbb{B}^n$ kann von einem Schaltkreis der Größe $(1 + o(1))2^n/n$ über der Basis $B = \{0, 1, \oplus, \wedge\}$ berechnet werden.*

Beweis. Sei $S(m, t) = \max\{S(\mathcal{F}) \mid \mathcal{F} \subseteq \mathbb{B}^m \text{ und } \|\mathcal{F}\| \leq t\}$.

Behauptung 1. *Für jedes $p \geq 1$ gilt $S(m,t) \leq 2^m + \lceil 2^m/p \rceil \cdot 2^p + t2^m/p$.*

Beweis. Wir geben einen Schaltkreis an, der alle t Funktionen in $\mathcal{F}$ simultan berechnet.
Schritt 1: Für alle $a_1, \ldots, a_m \in \{0,1\}$, berechne $x_1^{a_1} \wedge x_2^{a_2} \wedge \cdots \wedge x_m^{a_m}$. Dies sind genau 2^m Konjunktionen. Berechnen wir diese in der Reihenfolge ihrer Größe $a_1 + \cdots + a_m$, so wird für jede weitere Konjunktion höchstens ein weiteres Gatter benötigt.
Schritt 2: Teile die 2^m Konjunktionen in $q = \lceil 2^m/p \rceil$ Gruppen $C_1, \ldots, C_q$ mit je $\leq p$ Elementen auf und berechne innerhalb jeder Gruppe alle möglichen $\oplus$-Summen von Konjunktionen. Da in jeder Gruppe maximal 2^p Summen zu berechnen sind und für jede weitere Summe höchstens ein weiteres Gatter benötigt wird, reichen hierfür $q \cdot 2^p = \lceil 2^m/p \rceil \cdot 2^p$ weitere Gatter aus.
Schritt 3: Wir benutzen nun die Ringsummenexpansion

$$\bigoplus_{i=1}^{q} \alpha_i \wedge g_i,$$

um jedes $f \in \mathcal{F}$ mit $\leq q-1$ weiteren $\oplus$-Gattern zu berechnen. Insgesamt werden also nicht mehr als $t \cdot (q-1) \leq t \cdot 2^m/p$ weitere $\oplus$-Gatter benötigt. □

Behauptung 2. *Für $p \geq 1$ und $0 \leq m \leq n$ gilt*

$$S(n,1) \leq 3 \cdot 2^{n-m} + 2^m + \lceil 2^m/p \rceil \cdot 2^p + 2^n/p.$$

Beweis. Sei $f \in \mathbb{B}^n$ beliebig. Wir benutzen die erweiterte Ringsummenexpansion

$$f(x_1, \ldots, x_n) = \bigoplus_{a_{m+1}, \ldots, a_n \in \{0,1\}} h_{a_{m+1} \ldots a_n}(x_1, \ldots, x_m) \wedge x_{m+1}^{a_{m+1}} \wedge \cdots \wedge x_n^{a_n},$$

um f zu berechnen. Nach Behauptung 1 benötigen wir für die Berechnung der Funktionen $h_{a_{m+1} \ldots a_n}$ maximal $2^m + \lceil 2^m/p \rceil \cdot 2^p + 2^{n-m} \cdot 2^m/p = 2^m + \lceil \frac{2^m}{p} \rceil \cdot 2^p + 2^n/p$ Gatter. Da sich zudem die Konjunktionen $x_{m+1}^{a_{m+1}} \wedge \cdots \wedge x_n^{a_n}$ mit 2^{n-m} Gattern berechnen lassen, werden für f nur noch 2^{n-m} weitere $\wedge$-Gatter zur Berechnung der Summanden sowie $2^{n-m} - 1$ $\oplus$-Gatter zur Berechnung der Summe benötigt. □

Setzen wir nun die Werte $m = \lceil \sqrt{n} \rceil$ und $p = n - m - \lceil \log n \rceil$ in Behauptung 2 ein, so erhalten wir für die einzelnen Terme folgende Abschätzungen:

$$3 \cdot 2^{n-m} + 2^m \leq 3 \cdot 2^{n-\sqrt{n}} + 2^{\sqrt{n}+1} = o\left(2^n/n\right).$$

$$\lceil 2^m/p \rceil \cdot 2^p = O(2^{m+p}/p) = O(2^{n-\log n}/p) = o\left(2^n/n\right).$$

$$\frac{2^n}{p} = \frac{2^n}{n}\left(1 + \frac{n-p}{p}\right) = \frac{2^n}{n}\left(1 + \frac{\lceil \sqrt{n} \rceil + \lceil \log n \rceil}{n - \lceil \sqrt{n} \rceil - \lceil \log n \rceil}\right) = \frac{2^n}{n}\left(1 + o(1)\right).$$

■

2.5 Untere Schranken

Satz 37 (Shannon)**.** *Fast alle n-stelligen booleschen Funktionen benötigen zu ihrer Berechnung durch Schaltkreise über $\mathbb{B}^2$ mindestens $2^n/n$ Berechnungsgatter, d.h.*

$$\lim_{n\to\infty} \frac{\|\{f \in \mathbb{B}^n \mid S_{\mathbb{B}^2}(f) \leq 2^n/n\}\|}{\|\mathbb{B}^n\|} = 0.$$

Beweis. Für $q \geq 1$ definiere $N_{q,n} = \|\{f \in \mathbb{B}^n \mid S_{\mathbb{B}_2}(f) \leq q\}\|$. Jeder Schaltkreis C über $\mathbb{B}^2$ mit n Eingabegattern und q Berechnungsgattern kann durch eine Abbildung

$$\tau\colon \{1,\ldots,q\} \to G^2 \times \{1,\ldots,16\}$$

beschrieben werden, wobei $G = \{x_1,\ldots,x_n\} \cup \{1,\ldots,q\}$ die Menge der Eingangs- und Berechnungsgatter ist. Dabei bedeutet $\tau(i) = (j,k,l)$, dass das i-te Berechnungsgatter die Eingänge j und k hat und die l-te Funktion aus $\mathbb{B}^2$ berechnet ($\|\mathbb{B}^2\| = 16$). Das Ausgabegatter sei das q-te Gatter.

Für die Menge $T_{q,n}$ aller Abbildungen, die korrekten Schaltkreisen entsprechen, gilt dann

$$\|T_{q,n}\| \leq (16 \cdot (n+q)^2)^q.$$

Viele unterschiedliche Elemente $\tau \in T_{q,n}$ entsprechen Schaltkreisen $C(\tau)$, die die gleiche Funktion berechnen. Daher definieren wir auf $T_{q,n}$ eine Äquivalenzrelation wie folgt: $\tau \sim \sigma$, falls $C(\tau)$ und $C(\sigma)$ die gleiche Funktion $f \in \mathbb{B}^n$ berechnen. Da jede Funktion $f \in \mathbb{B}^n$, die mit $\leq q$ Gattern berechenbar ist, auch von einem Schaltkreis mit genau q Gattern berechnet wird, entsprechen die Äquivalenzklassen von $\sim$ genau den Funktionen $f \in \mathbb{B}^n$ mit $S_{\mathbb{B}_2}(f) \leq q$.

Sei $\pi \in S_G$ eine Permutation auf G mit $\pi(x_i) = x_i$ für $i = 1,\ldots,n$ und $\pi(q) = q$. Für eine Abbildung $\tau \in T_{q,n}$ definieren wir die Abbildung τ_π wie folgt:

$$\tau_\pi(\pi(i)) = (\pi(j), \pi(k), l), \text{ falls } \tau(i) = (j,k,l) \text{ ist.}$$

Dann berechnet Berechnungsgatter v in $C(\tau)$ die gleiche Funktion wie Berechnungsgatter $\pi(v)$ in $C(\tau_\pi)$.

Ein Schaltkreis C heißt *reduziert*, falls alle Gatter von C unterschiedliche Funktionen berechnen, d.h. für alle Gatter $v \neq v'$ in C ist $val_v \neq val_{v'}$.

Entspricht τ einem reduzierten Schaltkreis und ist $\tau_\pi = \tau$, so muss π die Identität sein. Demnach sind alle Abbildungen τ_π verschieden, aber die Schaltkreise $C(\tau_\pi)$ berechnen gleiche Funktion. Da zudem jede Funktion, die mit q Gattern berechenbar ist, auch von einen reduzierten Schaltkreis mit genau q Berechnungsgattern berechnet wird, enthält jede Äquivalenzklassen von $\sim$ mindestens $(q-1)!$ Elemente. Daher folgt

$$N_{q,n} \leq \frac{|T_{q,n}|}{(q-1)!} \leq q \cdot \frac{[16(n+q)^2]^q}{q!}.$$

Mit der Stirlingschen Formel $q! \sim \sqrt{2\pi q}(\frac{q}{e})^q \geq \frac{q^q}{e^q}$ folgt

$$\begin{aligned} N_{q,n} &\leq q \cdot \frac{\left(16(n+q)^2\right)^q}{q!} \\ &\leq q \cdot d^q \cdot \frac{(n+q)^{2q}}{q^q} && \text{für } d = 16e \\ &\leq q \cdot d^q \cdot \frac{4^q \cdot q^{2q}}{q^q} && \text{für } q \geq n, \text{ d. h. } n+q \leq 2q \\ &= q(cq)^q \leq (2cq)^q && \text{für } c = 4d. \end{aligned}$$

Demnach ist für $q = 2^n/n$

$$\begin{aligned} N_{q,n} &\leq (2c2^n/n)^{2^n/n} \\ &\leq 2^{2^n} 2^{-2^n/n} && \text{für } n \geq 4c. \end{aligned}$$

Also konvergiert $N_{q,n}/\|\mathbb{B}^n\| \leq 2^{-2^n/n}$ gegen 0 für $n \to \infty$. ■