

Vorlesungsskript  
Schaltkreiskomplexität  
Sommersemester 2009

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

*8. Mai 2009*

# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>1</b>
1.1	Addition von Binärzahlen . . . . .	1
1.1.1	Schulmethode . . . . .	1
1.1.2	Carry-lookahead Methode . . . . .	2
<b>2</b>	<b>Grundlegende Definitionen</b>	<b>4</b>
2.1	Boolesche Schaltkreise . . . . .	4
2.2	Schaltkreis-Familien . . . . .	6
2.3	Die NC-Hierarchie . . . . .	7
<b>3</b>	<b>Berechnung weiterer elementarer Funktionen</b>	<b>8</b>
3.1	Multiplikation von Binärzahlen . . . . .	8
3.2	Addition von logarithmisch vielen Binärzahlen . . . . .	9
<b>4</b>	<b><math>AC^0</math>-Reduktionen</b>	<b>11</b>
4.1	Addieren von Bits . . . . .	12
4.2	Multiplikation von Binärzahlen . . . . .	13
4.3	Sortieren von Binärzahlen . . . . .	15

# 1 Einführung

## 1.1 Addition von Binärzahlen

$$\text{ADD}^{2n} : \{0, 1\}^{2n} \mapsto \{0, 1\}^{n+1}$$

$$\text{mit: } \text{ADD}^{2n}(a_{n-1} \dots a_0 b_{n-1} \dots b_0) = s_n \dots s_0 \quad ,$$

$$\text{wobei: } \sum_{i=0}^n s_i 2^i = \sum_{i=0}^{n-1} a_i 2^i + \sum_{i=0}^{n-1} b_i 2^i$$

Für  $k \in \mathbb{N}$  sei

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

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

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

$k$	$\text{bin}(k)$	$ \text{bin}(k) $
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.

$$s_0 = a_0 \oplus b_0$$

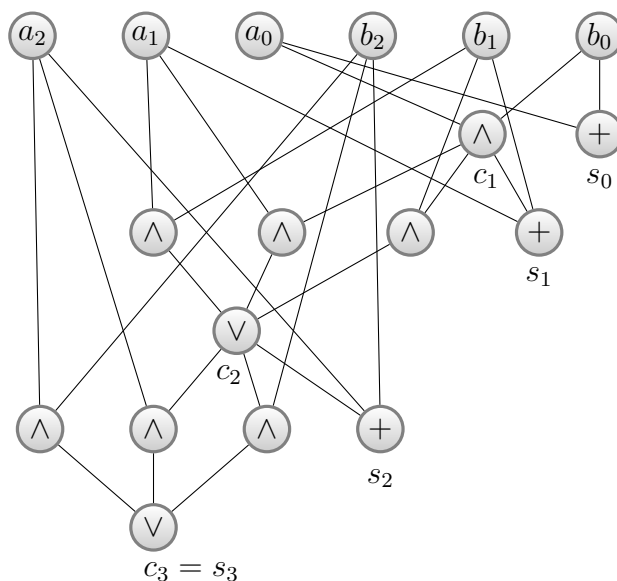
$$c_1 = a_0 \wedge b_0$$

$$c_i = (a_{i-1} \wedge b_{i-1}) \vee (a_{i-1} \wedge c_{i-1}) \vee (b_{i-1} \wedge c_{i-1}) \quad i = 2, \dots, n$$

$$s_i = a_i \oplus b_i \oplus c_i \quad i = 1, \dots, n-1$$

$$s_n = c_n$$

**Beispiel 1.** Für  $n = 3$  erhalten wir folgenden Schaltkreis.



Größe (Anzahl der Gatter):

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

Tiefe (maximale Pfadlänge):  $2n - 1$ .

### 1.1.2 Carry-lookahead Methode

$$g_i = a_i \wedge b_i \quad (\text{Position } i \text{ generiert Carry}) \quad i = 0, \dots, n-1$$

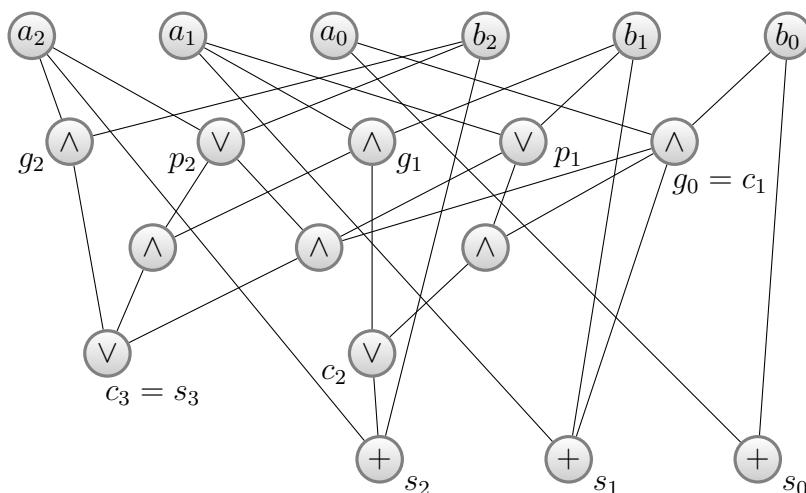
$$p_i = a_i \vee b_i \quad (\text{Position } i \text{ propagiert Carry}) \quad i = 1, \dots, n-1$$

$$\rightsquigarrow c_i = \bigvee_{j=0}^{i-1} \left( g_j \wedge \bigwedge_{k=j+1}^{i-1} p_k \right) \quad i = 1, \dots, n$$

$$s_i = a_i \oplus b_i \oplus c_i \quad i = 1, \dots, n$$

$$s_0 = a_0 \oplus b_0$$

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



**Größe:**

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

**Tiefe:** Da die Tiefe für die Carry-Berechnung durch 3 begrenzt ist, ist die Gesamttiefe  $\leq 4$ .

## 2 Grundlegende Definitionen

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

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

**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"
- $\wedge^m : \{0, 1\}^m \mapsto \{0, 1\}$  mit  $\wedge^m(a_1 \dots a_m) = \prod_{i=1}^m a_i$ .

**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  $f^n(a_1 \dots a_n)$  schreiben wir auch einfach  $f(a_1 \dots a_n)$ .

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

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

### 2.1 Boolesche Schaltkreise

**Beispiel 8.**

- $B = \{\wedge^2, \vee^2, \oplus^2, \oplus^3\}$ .
- $B_0 = \{\neg, \wedge^2, \vee^2\}$  heißt Standardbasis mit beschränktem Fanin.
- $B_1 = \{\neg, (\wedge^m)_{m \in \mathbb{N}}, (\vee^m)_{m \in \mathbb{N}}\}$  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 ist und  $\alpha, \beta$  und  $\omega$  Funktionen der Form

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

mit folgenden Eigenschaften sind.

1.  $\alpha$  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  $val_v : \{0, 1\}^n \rightarrow \{0, 1\}$  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  $v_1, \dots, v_k$  die Vorgänger von  $v$ , geordnet gemäß  $\alpha$  (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  $\sum a_i$  ab), kann auf die Angabe von  $\alpha$  natürlich verzichtet werden.

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

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

mit

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

Die **Größe** eines Schaltkreises  $C$  ist

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

und seine **Tiefe** ist

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

## 2.2 Schaltkreis-Familien

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

$$f_{\mathbf{C}}(a_1 \dots a_n) = f_{C_n}(a_1 \dots a_n).$$

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

$$L(\mathbf{C}) = \{x \in \{0, 1\}^* \mid f_{\mathbf{C}}(x) = 1\}.$$

Wir werden im Folgenden die charakteristische Funktion einer Sprache  $L$  mit dieser identifizieren.

Seien  $s, d : \mathbb{N} \rightarrow \mathbb{N}$ .  $\mathbf{C}$  hat die **Größe**  $s$  (**Tiefe**  $d$ ), falls gilt:

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

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

$$\text{Size}_B(s) = \left\{ f_{\mathbf{C}} \mid \begin{array}{l} \mathbf{C} \text{ ist eine Schaltkreisfamilie über } B \text{ mit} \\ \text{size}(\mathbf{C}) = \mathcal{O}(s) \end{array} \right\},$$

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

$$\text{Depth}_B(d) = \left\{ f_{\mathbf{C}} \mid \begin{array}{l} \mathbf{C} \text{ ist eine Schaltkreisfamilie über } B \text{ mit} \\ \text{depth}(\mathbf{C}) = \mathcal{O}(d) \end{array} \right\}$$

und

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

**Bemerkung 10.**

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

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

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

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



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

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

*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  $\log k$ .  $\square$

## 2.3 Die NC-Hierarchie

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

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

Zusammen mit dem Negationsgatter bilden diese die Basis  $B_2 = \{\neg, (\text{MAJ}^n)\}$  mit unbeschränktem Fanin.

Für eine Klasse  $\mathbb{F}$  von Funktionen  $f : \mathbb{N} \rightarrow \mathbb{N}$  ist

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

und analog für die anderen Klassen. Die wichtigsten Schaltkreisklassen sind

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

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

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

und

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

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

$$\text{NC}^0 \subseteq \text{AC}^0 \subseteq \text{TC}^0 \subseteq \text{NC}^1 \subseteq \text{AC}^1 \subseteq \text{TC}^1 \subseteq \text{NC}^2 \subseteq \dots \subseteq \text{NC},$$

wobei  $\text{NC} = \bigcup_{i \geq 0} \text{NC}^i = \bigcup_{i \geq 0} \text{AC}^i = \bigcup_{i \geq 0} \text{TC}^i$  ist.

Wie wir gesehen haben, ist  $\text{ADD} \in \text{Size-Depth}_B(n^2, 1)$  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  $\text{ADD} \in \text{Size-Depth}_{B_1}(n^2, 1) \subseteq \text{AC}^0$ .

## 3 Berechnung weiterer elementarer Funktionen

### 3.1 Multiplikation von Binärzahlen

Als nächstes betrachten wir die Multiplikation von Binärzahlen.

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

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

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.**  $\text{ITADD}^{n^2} : \{0, 1\}^{n^2} \rightarrow \{0, 1\}^{2n}$  mit

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

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  $\text{AC}^1$  Schaltkreis für  $\text{ITADD}$ . Tatsächlich können wir  $\text{ITADD}$  sogar in  $\text{NC}^1$  berechnen (wir werden später sehen, dass es sogar in  $\text{TC}^0$  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:

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

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

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

### 3 Berechnung weiterer elementarer Funktionen

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  $\text{ITADD} \in \text{NC}^1$ .

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

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

wobei  $\sum_{i=1}^n a_i = \sum_{j=0}^{\text{len}(n)-1} b_j 2^j$  ist. Wir werden sehen, dass BCOUNT und ITADD vergleichbare Schaltkreiskomplexität haben (formal: ITADD ist in konstanter Tiefe auf BCOUNT reduzierbar; siehe Abschnitt 4).

Auch die Majoritätsfunktion MAJ hat ähnliche Komplexität. Um  $\text{MAJ}(a_1 \dots a_n)$  in  $\text{NC}^1$  zu berechnen, berechnet man zuerst  $b = \text{BCOUNT}^n(a_1, \dots, a_n)$  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  $\text{LEQ}^{2n} : \{0, 1\}^{2n} \rightarrow \{0, 1\}$  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  $\text{AC}^0$  berechenbar ist (siehe Übungen), folgt  $\text{MAJ} \in \text{NC}^1$  und somit  $\text{TC}^i \subseteq \text{NC}^{i+1}$ .

**Korollar 16.**

1.  $\text{MULT} \in \text{NC}^1$ ,
2.  $\text{BCOUNT} \in \text{NC}^1$ ,
3.  $\text{MAJ} \in \text{NC}^1$ .

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

## 3.2 Addition von logarithmisch vielen Binärzahlen

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

### 3 Berechnung weiterer elementarer Funktionen Addition von logarithmisch vielen Binärzahlen

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

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

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

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

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

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

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

Die Funktion  $\text{LOGITADD}$  wird sich noch bei der Berechnung von  $\text{ITADD}$  (und damit auch von  $\text{MULT}$  und  $\text{BCOUNT}$ ) in  $\text{TC}^0$  als nützlich erweisen.

## 4 $AC^0$ -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  $g : \{0, 1\}^* \rightarrow \{0, 1\}^+$  heißt **längenverträglich**, falls  $|g(x)| = r(|x|)$  für eine Funktion  $r : \mathbb{N} \rightarrow \mathbb{N}^+$  ist.
2. Für eine längenverträgliche Funktion  $g$  sei  $\text{bits}(g) = \{g_i^n \mid n \geq 0, i = 1, \dots, r(n)\}$  die Menge der booleschen Funktionen  $g_i^n$ , die das  $i$ -te Bit von  $g^n(x)$  berechnen, d.h. im Falle  $g^n(x) = a_1 \dots a_{r(n)}$  ist  $g_i^n(x) = a_i$  für  $i = 1, \dots, r(n)$ . Im Fall  $r(n) = 1$  (d.h.  $g$  ist eine Familie von booleschen Funktionen) gilt also  $\text{bits}(g) = \{g^n \mid n \geq 0\}$ .
3. Entsprechend definieren wir für eine Menge  $\mathcal{F}$  von längenverträglichen Funktionen die Menge  $\text{bits}(\mathcal{F}) = \bigcup_{f \in \mathcal{F}} \text{bits}(f)$ .
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  $B_1 \cup \text{bits}(\mathcal{G})$  der Größe  $n^c + c$  und Tiefe  $d$  berechenbar.

Hierbei trägt jedes Gatter aus  $\text{bits}(\mathcal{G})$  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  $f \leq_{AC^0} g$  oder  $f \in AC^0(g)$  geschrieben.

**Proposition 20.**

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

**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:  $f \leq_{cd} g, g \in AC^i \Rightarrow f \in AC^i$ .
- iv) Für  $i \geq 0$  gilt:  $f \leq_{cd} g, g \in TC^i \Rightarrow f \in TC^i$ .
- v) Für  $i \geq 1$  gilt:  $f \leq_{cd} g, g \in NC^i \Rightarrow f \in NC^i$ .
- vi)  $f \in AC^0 \Rightarrow f \leq_{cd} g$ .

## 4.1 Addieren von Bits

Um  $\text{BCOUNT} \leq_{cd} \text{MAJ}$  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.**  $\{T_m \mid m \in \mathbb{N}\} \leq_{cd} \text{MAJ}$ .

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

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

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

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

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

**Satz 23.**  $\text{BCOUNT} \leq_{cd} \text{MAJ}$ .

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

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

Allgemeiner ist  $s_j = 1$ , wenn ein  $k \in S_j^n$  ex. mit  $\sum a_i = k$ , wobei

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

ist.  $\square$

**Korollar 24.**  $\text{BCOUNT} \in \text{TC}^0$ .

## 4.2 Multiplikation von Binärzahlen

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

**Satz 25.**  $\text{ITADD} \leq_{cd} \text{BCOUNT}$ .

*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  $\text{BCOUNT}$ -Gattern. Die Bits dieser Zahlen lassen sich wie im Beweis von Satz 18 auf  $\text{len}(n)$  Zahlen mit je  $n + \text{len}(n)$  Bits verteilen, deren Summe  $s$  ergibt. Nach Satz 18 können diese in  $\text{AC}^0$  aufaddiert werden.  $\square$

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

Umgekehrt lässt sich  $\text{BCOUNT}$  auch auf  $\text{MULT}$  reduzieren.

**Satz 27.**  $\text{BCOUNT} \leq_{cd} \text{MULT}$

*Beweis.* Um die Binärdarstellung der Summe  $s$  von  $n$  Bits  $a_{n-1}, \dots, a_0$  zu erhalten, setzen wir  $k = \text{len}(n)$  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 Binärdarstellung

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

haben. Als Produkt  $w$  von  $u$  und  $v$  erhalten wir die Zahl  $w = \sum_{i=0}^{2n-2} c_i \cdot 2^{ki}$ , wobei  $c_i$  für  $i = 0, \dots, n-1$  den Wert  $c_i = a_0 + \dots + a_i$  und für  $i = n, \dots, 2n-2$  den Wert  $c_i = a_{i-n+1} + \dots + a_{n-1}$  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.  $\square$

**Korollar 28.**  $\text{MAJ} \equiv_m \text{MULT} \equiv_m \text{ITADD} \equiv_m \text{BCOUNT}$ .

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

Wir werden  $\text{ITMULT}$  auf  $\text{MAJ}$  reduzieren. Hierzu benutzen wir den chinesischen Restsatz (CRS), welcher besagt, dass ein lineares Kongruenzgleichungssystem

$$\begin{aligned} x &\equiv_{m_1} b_1 \\ &\vdots \\ x &\equiv_{m_k} b_k \end{aligned} \tag{4.1}$$

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

$$f : \mathbb{Z}_m \rightarrow \mathbb{Z}_{m_1} \times \dots \times \mathbb{Z}_{m_k}$$

mit  $f(x) = (x \bmod m_1, \dots, x \bmod m_k)$  ein Isomorphismus zwischen den Ringen  $\mathbb{Z}_m$  und  $\mathbb{Z}_{m_1} \times \dots \times \mathbb{Z}_{m_k}$  ist.

Tatsächlich lässt sich für das Gleichungssystem (4.2) 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

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

welche sich mit dem erweiterten Euklidischen 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  $i = 1, \dots, k$ . Dies zeigt, dass (4.2) für alle  $(b_1, \dots, b_k)$  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. (4.2) ist sogar eindeutig lösbar.

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

*Beweis.* Gegeben sind  $n$  Binärzahlen  $a_1, \dots, a_n$ , wobei  $a_i = a_{i,n-1}, \dots, a_{i,0}$  für  $i = 1, \dots, n$  ist. Das Produkt dieser Zahlen sei  $c$  mit der Binärdarstellung  $c_{n^2-1} \dots c_0$ . Seien  $p_1 = 2, p_2 = 3, \dots, p_k$  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  $i = 1, \dots, n$  und  $j = 1, \dots, k$ , im 2. Schritt die modularen Produkte

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

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

$$\begin{aligned} x &\equiv_{p_1} b_1 \\ &\vdots \\ x &\equiv_{p_k} b_k \end{aligned} \tag{4.2}$$

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 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  $b_j = \prod_{i=1}^n b_{i,j} \bmod p_j$ . Da die multiplikative Gruppe  $\mathbb{Z}_{p_j}^*$  zyklisch ist, existiert ein Erzeuger  $g_j \in \mathbb{Z}_{p_j}^*$  mit  $\mathbb{Z}_{p_j}^* = \{g^e \mid e = 0, \dots, p_j - 2\}$ . Daher gilt

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

wobei die Zahlen  $e_{i,j} \in \{0, \dots, p_j - 2\}$  so gewählt sind, dass  $g_j^{e_{i,j}} = b_{i,j}$  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  $c = f^{-1}(b_1, \dots, b_k)$ . 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  $c' = \sum_{j=1}^k b_j y_j \bmod m$  durch ein ITADD-Gatter bestimmen. Um schließlich aus  $c'$  den Wert von  $c$  zu erhalten, bestimmen wir zuerst das größte  $d \in \{0, \dots, k\}$  mit  $dm \leq c'$  und berechnen  $c = c' - dm$ . □

### 4.3 Sortieren von Binärzahlen

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

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

für  $i = 1, \dots, n - 1$  und die beiden Multimengen

$$\{\{a'_{1,n-1} \dots a'_{1,0}, \dots, a'_{n,n-1} \dots a'_{n,0}\}\} \text{ und } \{\{a_{1,n-1} \dots a_{1,0}, \dots, a_{n,n-1} \dots a_{n,0}\}\}$$

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

$$\text{un}_n(k) = \underbrace{1 \dots 1}_k \underbrace{0 \dots 0}_{n-k} = 1^k 0^{n-k}$$

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

**Lemma 30.**  $\text{UCOUNT} \leq_{cd} \text{BCOUNT}$ .

*Beweis.* Sei  $\text{UCOUNT}(a_1, \dots, a_n) = b_1 b_2 \dots b_n$ . 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  $\text{BCOUNT}$ -Gatter berechnen lässt.  $\square$

Auch die umgekehrte Reduktion lässt sich leicht zeigen.

**Lemma 31.**  $\text{BCOUNT} \leq_{cd} \text{UCOUNT}$ .

*Beweis.* Sei  $\text{UCOUNT}(a_1, \dots, a_n) = b_1 b_2 \dots b_n$ . Zudem sei  $b_0 = 1$  und  $b_{n+1} = 0$ , also  $b_0 b_1 \dots b_{n+1} \in 1^+ 0^+$ . 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 weiter oben definierten Mengen  $R_j = \{r \in \{0, \dots, n\} \mid j\text{-es Bit in } r \text{ ist } 1\}$  folgt  $\text{bin}(\sum a_i) = c_{\text{len}(n)-1} \dots c_0$ , wobei  $c_j = \bigvee_{i \in R_j} d_i$  ist.  $\square$

**Satz 32.**  $\text{SORT} \leq_{cd} \text{UCOUNT}$ .

*Beweis.* Um  $n$  Binärzahlen  $a_i = a_{i,n-1} \dots a_{i,0}$ ,  $i = 1, \dots, n$  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  $c'_j = \text{UCOUNT}(c_{1,j}, \dots, c_{n,j})$ . Dann gilt  $c'_k = c'_{k,1} \dots c'_{k,n} = \text{un}_n(i)$ , wobei  $i$  die Position von  $a_k$  in der *sortierten* Folge ist. Sei  $a'_1, \dots, a'_n$  die sortierte Folge mit  $a'_i = a'_{i,n-1} \dots a'_{i,0}$ . Dann gilt

$$a'_{i,j} = \bigvee_{k=1}^n \underbrace{c'_{k,i} \wedge \neg c'_{k,i+1}}_{c'_k = 1^i 0^{n-i} = \text{un}_n(i)} \wedge a_{k,j}.$$

$\square$

**Korollar 33.**  $\text{SORT} \equiv_{cd} \text{UCOUNT} \equiv_{cd} \text{MAJ}$ .