Universität Trier

Fachbereich IV - Informatik

Christoph Meinel:

Schaltkreiskomplexitätstheorie

Vorlesungsskript WS 1992/93
überarbeitete Version SS 1995

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Grundbegriffe der Komplexitätstheorie

- Die Komplexitätstheorie versucht algorithmisch lösbare Probleme ("Entscheidungsprobleme" bzw. "Sprachen") gemäß ihrem Bedarf an Berechnungsressourcen (Rechenzeit, Speicherplatz, Hardwareaufwand, ...) zu klassifizieren.

- Zwei Grundaufgaben:

  1. Bereitstellung und Analyse möglichst effizienter Lösungsverfahren für konkrete Probleme
     "Theorie der effizienten Algorithmen";
     man erhält *obere Schranken* für die Komplexität der Probleme.
     (z.B.: Die Existenz eines $\mathrm{O}(n^3)$ Algorithmus für das Wortproblem kontextfreier Sprache zeigt, daß die Komplexität eines Wortproblems *höchstens* $\mathrm{O}(n^3)$ ist).
  2. Beweise von *unteren Schranken* für den Ressourcenbedarf konkreter Probleme
     "Computational Complexity";
     Nachweis unterer Schranken ist schwierig, weil Beweisargument jeden Algorithmus – auch bisher noch nicht gefundene – in Betracht ziehen müssen.

- Es ist zu spezifizieren:

  - Was sind Probleme ? Wie sind sie zu spezifizieren ?
  - Welche Berechnungsmodelle werden betrachtet ?
  - Um welche Komplexitätsmaße geht es bei der Untersuchung der Problemlösungsalgorithmen ?

1.1.1 Probleme

Probleme sind partielle Funktionen $F : \{0,1\}^* \longrightarrow \{0,1\}$.

- Wir sind besonders interessiert an *vollständig definierten* Funktionen

$$F : \{0,1\}^* \longrightarrow \{0,1\};$$

  sog. *Entscheidungsproblemen*. Entscheidungsprobleme kann man als Familie *Boolescher Funktionen* auffassen:

$$F = (f_n)_{n \in N} \quad f_n : \{0,1\}^n \longrightarrow \{0,1\} \text{ mit } f_n := F|_{\{0,1\}^n}.$$

  f_n beschreibt F auf Eingaben der Länge n.

- Anstelle von Entscheidungsproblemen spricht man auch von *Sprachen*:

$$L \subseteq \{0,1\}^* \longleftrightarrow F_L \text{ mit } F_L(w) = 1 \Leftrightarrow w \in L \ (L = F_L^{-1}(1))$$

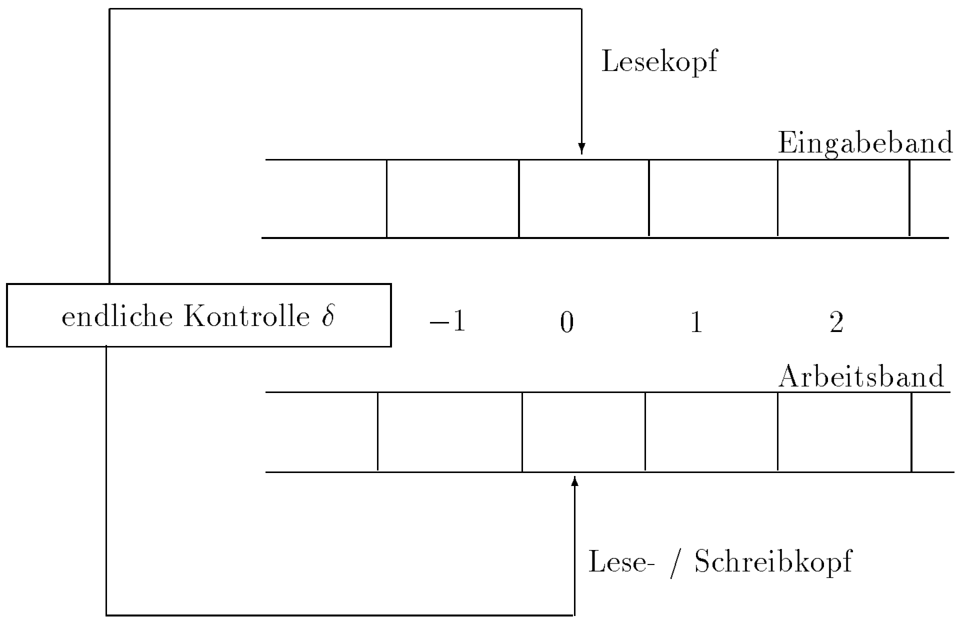
1.1.2 Berechnungsmodelle

Wir unterscheiden zwei Modelltypen:

- maschinenbasierte Modelle (z.B. Turing Maschine)
- graphenbasierte Modelle (z.B. Boolsche Schaltkreise, Formeln, ...)

Definition 1.1.
Eine *Turing Maschine* (TM) ist ein 7–Tupel $M = (Z, \Gamma, \square, q_s, \delta, q_0, q_1)$
mit

Z	: endliche Menge von *Zuständen*
Γ	: endliches *Alphabet* von Bandsymbolen ($\Gamma = \{0,1\}$)
$\square \notin \Gamma$	: *leeres Symbol*
$q_s \in Z$	: *Initialzustand*
$q_0, q_1 \in Z$	: verwerfender bzw. akzeptierender Zustand
$\delta : Z \times \{\Gamma \cup \{\square\}\}^2$	$\longrightarrow Z \times \Gamma \times \{L, R, S\}^2$: *Zustandsüberführungsfunktion*



Arbeitsweise: Aneinanderreihung von Berechnungsschritten

Berechnungsschritt: Je ein Symbol wird auf dem Eingabeband und ein Symbol auf dem Arbeitsband gelesen. In Abhängigkeit von einem Zustand q wird das Symbol auf dem Arbeitsband überschrieben und auf jedem Band wird zu einem der Nachbarsymbole übergegangen (L, R, S).

M arbeitet deterministisch, d.h. jeder Berechnungsschritt ist eindeutig bestimmt.

Startsituation einer Berechnung:

- TM M ist im Zustand q_0
- alle Köpfe sind auf Feld 0
- Felder $0, \ldots, (n-1)$ auf Eingabeband sind mit Eingabe $w \in \{0,1\}^*, |w| = n$ beschrieben
- alle übrigen Felder und gesamtes Arbeitsband sind mit leerem Symbol $\Box$ beschrieben

TM M berechnet Funktion $F_M : \{0,1\}^* \longrightarrow \{0,1\}$: ("löst Entscheidungsproblem F_M")

$$F_M(w) = \begin{cases} 0 & \text{falls M mit Eingabe } w \text{ gestartet, Zustand } q_0 \text{ erreicht} \\ 1 & \text{falls M mit Eingabe } w \text{ gestartet, Zustand } q_1 \text{ erreicht} \\ \text{nicht def.} & \text{sonst.} \end{cases}$$

$$L(M) = \{w \in \{0,1\}^* \quad | \quad \text{TM } M \text{ erreicht nach Start mit Eingabe } w \text{ den Zustand } q_1\}.$$

$L(M)$ ist die von M *akzeptierte* Sprache: $L(M) = F_M^{-1}(1)$. Aus der Berechnungstheorie ist bekannt:

Für jedes rekursive $F : \{0,1\}^* \longrightarrow \{0,1\}$ existiert eine TM M mit $F_M = F$.

Betrachte im folgenden meist rekursives Entscheidungsproblem F.

Definition 1.2. (Zeitkomplexität und Raumkomplexität)
Sei M TM mit F_M rekursiv, $w \in \{0,1\}^*$

$$\begin{aligned} time_M(n) &= \text{Zahl der Rechenschritte von } M \text{ bei Eingabe von } w; \\ space_M(n) &= \text{Zahl der auf dem Arbeitsband besuchten Felder bei Eingabe von } w; \end{aligned}$$

$time_M(w), space_M(w) : \Gamma^* \longrightarrow I\!N$ heißen Zeitkomplexität bzw. Raumkomplexität.

Definition 1.3. (Worst-case Komplexität)

$$\begin{aligned} time_M(n) &= max\{time_M(w)|; |w| = n\}, \\ space_M(n) &= max\{space_M(w)|; |w| = n\}. \end{aligned}$$

Komplexitätsklassen

Komplexitätsklassen klassifizieren Entscheidungsprobleme.

Definition 1.4.
Seien $t, s : I\!N \longrightarrow I\!N$

$$\begin{aligned} TIME(t(n)) &= \{L| \text{ es existiert eine TM } M \text{ mit } L(M) = L,\ time_M(n) = O(t(n))\}, \\ SPACE(s(n)) &= \{L| \text{ es existiert eine TM } M \text{ mit } L(M) = L,\ space_M(n) = O(s(n))\}. \end{aligned}$$

TIME(t(n)) und SPACE(s(n)) heißen *Komplexitätsklassen* (Zeit– bzw. Raumkomplexitätsklasse).

Abstrakte Komplexitätstheorie:
untersucht Eigenschaften und Beziehungen zwischen Komplexitätsklassen, Hierarchiesätze, usw.

Theorie der effizient lösbaren ("praktisch ausführbaren") Entscheidungsprobleme:
ist die Theorie der folgenden Komplexitätsklasse:

$$P := \bigcup_{k \geq 0} TIME(n^k),$$

P ist die Klasse der Entscheidungsprobleme mit polynomialen Lösungsalgorithmen.
P ist robust gegenüber der Modifikation des Berechnungsmodells!

Wichtige Komplexitätsklassen:

$$L := SPACE(log(n))$$
$$PSPACE = \bigcup_{k \geq 0} SPACE(n^k).$$

Satz 1.1.
Es gilt $L \subseteq P \subseteq PSPACE$.

Beweis.

- $P \subseteq PSPACE$ trivial
  in polynomialer Zeit können nicht mehr als polynomial viele Felder auf dem Arbeitsband berechnet werden.

- $L \subseteq P$
  Man überlegt sich sofort, daß im Laufe einer Rechnung je zwei Konfigurationen voneinander verschieden sein müssen (Stellung der Köpfe, Inschrift des Arbeitsbandes). Somit kann es auf logarithmisch beschränktem Raum nur polynomial viele verschiedene Konfigurationen geben.
  Folglich gilt: $L \subseteq P$ ■ .

1.2 Boolesche Funktionen

Wir benutzen die folgenden Bezeichnungen:

$x, y, \ldots, x_i \ldots,$	*logische* oder *Boolesche Variable.*
$X_n = \{x_1, \ldots, x_n\}$	Menge der Booleschen Variablen $x_1, \ldots, x_n$.

Boolesche Variable $x_i \in X_n$ definieren Funktionen, sogenannte *Literale:*

$$x_i \text{ (bzw. } x_i^1) \; : \; \begin{array}{l} \{0,1\}^n \to \{0,1\} \\ a = (a_1, \ldots, a_n) \longmapsto a_i \end{array}$$

und

$$\overline{x}_i \text{ (bzw. } x_i^0) \; : \; \begin{array}{l} \{0,1\}^n \to \{0,1\} \\ a = (a_1, \ldots, a_n) \longmapsto \overline{a}_i \end{array}$$

Definition 1.5.
$\mathbb{B}_{n,m}$ bezeichnet die Menge der Booleschen Funktionen

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

Schreiben kurz $\mathbb{B}_n := \mathbb{B}_{n,1}$.

Satz 1.2.
Es gilt $\#\mathbb{B}_{n,m} = 2^{m2^n}$.

Beweis.
Sind A, B endliche Mengen, dann existieren $\#B^{\#A}$ viele Abbildungen $f : A \to B$. Für $A = \{0,1\}^n$, $B = \{0,1\}^m$ folgt damit: $\#\mathbb{B}_{n,m} = 2^{m2^n}$. ∎

Beispiel 1.1.

$\#\mathbb{B}_2 = 16 \quad (= 2^4)$
$\#\mathbb{B}_3 = 256 \quad (= 2^8)$
$\#\mathbb{B}_4 = 65536 \quad (= 2^{16})$
$\#\mathbb{B}_5 > 4 * 10^9$
$\#\mathbb{B}_6 > 16 * 10^{18}$.

Bezeichnung.
Sei $f \in \mathbb{B}_n$.
$on(f) := \{(x_1, \ldots, x_n) \in \{0,1\}^n \quad | \quad f(x_1, \ldots, x_n) = 1\} = f^{-1}(1)$.
$off(f) := f^{-1}(0)$.
Wir identifizieren häufig f mit $on(f)$.

1.2.1 Spezielle Funktionen aus $\mathbb{B}_0 \cup \mathbb{B}_1 \cup \mathbb{B}_2$

- *Konjunktion* (AND, $\wedge$, $\cdot$)

  $$\wedge(x, y) = 1 \text{ falls } x = 1 \text{ und } y = 1$$

  Bez.: $x \wedge y$, xy

- *Disjunktion* (OR, $\vee$, $+$)

  $$\vee(x, y) = 1 \text{ falls } x = 1 \text{ oder } y = 1$$

  Bez.: $x \vee y$, $x + y$

- *Parität* (EX–OR, EXOR, XOR, $\oplus$, $+ \mod 2$)

  $$\oplus(x, y) = 1 \text{ falls } (x = 1 \text{ und } y = 0) \text{ oder } (x = 0 \text{ und } y = 1)$$

  Bez.: $x \oplus y$, $(x + y) \mod 2$

- *NAND*

  $$\overline{\wedge}(x, y) = 0 \text{ falls } x = 1 \text{ und } y = 1$$

- *NOR*
$$\overline{\vee}(x, y) = 0 \text{ falls } x = 1 \text{ oder } y = 1$$
- *Negation* (NOT)
$$\neg(x) = 1 \text{ falls } x = 0$$
Bez.: $\neg x$, $\overline{x}$

Satz 1.3. (Rechenregeln)

(i) (Rechnen mit Konstanten)
$x \vee 0 = x$
$x \vee 1 = 1$
$x \wedge 0 = 0$
$x \wedge 1 = x$
$x \oplus 0 = x$
$x \oplus 1 = \overline{x}$

(ii) (Assoziativ– und Kommutativgesetze)
$\vee$, $\wedge$ *und* $\oplus$ *sind assoziativ und kommutativ.*

(iii) (Distributivgesetze)
$(\wedge, \vee)$, $(\vee, \wedge)$ *und* $(\oplus, \wedge)$ *sind distributiv, d. h.*
$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
$x \wedge (y \oplus z) = (x \wedge y) \oplus (x \wedge z)$

(iv) (Vereinfachungsregeln)
$x \vee x = x$ $\quad$ $x \wedge x = x$ $\quad$ $x \oplus x = 0$
$x \vee \overline{x} = 1$ $\quad$ $x \wedge \overline{x} = 0$ $\quad$ $x \oplus \overline{x} = 1$
$x \vee (x \wedge y) = x$
$x \wedge (x \vee y) = x$

(v) (deMorgan–Regeln)
$\neg(x_1 \wedge \ldots \wedge x_n) = \overline{x_1} \vee \ldots \vee \overline{x_n}$
$\neg(x_1 \vee \ldots \vee x_n) = \overline{x_1} \wedge \ldots \wedge \overline{x_n}$

Beweis.

für $x \wedge (y \oplus z) = (x \wedge y) \oplus (x \wedge z)$:

x	y	z	$x \wedge y$	$x \wedge z$	$y \oplus z$	$x \wedge (y \oplus z)$	$(x \wedge y) \oplus (x \wedge z)$
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0
0	1	0	0	0	1	0	0
0	1	1	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	0	1	1	1	1
1	1	0	1	0	1	1	1
1	1	1	1	1	0	0	0

Z. B. deMorgan–Regel: $\neg(x_1 \wedge \ldots \wedge x_n) = \overline{x}_1 \vee \ldots \vee \overline{x}_n$:

Induktiv über n:
Induktionsanfang: n=2

x	y	$(x \wedge y)$	$\neg(x \wedge y)$	$\overline{x}$	$\overline{y}$	$\overline{x} \vee \overline{y}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Induktionsschritt: n-1 $\rightarrow$ n

$$\begin{aligned}
\neg(x_1 \wedge \ldots \wedge x_n) &= \neg((x_1 \wedge \ldots \wedge x_{n-1}) \wedge x_n) \\
&= \neg(x_1 \wedge \ldots \wedge x_{n-1}) \vee \overline{x}_n \\
&= (\overline{x}_1 \vee \ldots \vee \overline{x}_{n-1}) \vee \overline{x}_n \\
&= \overline{x}_1 \vee \ldots \vee \overline{x}_n \qquad \blacksquare
\end{aligned}$$

1.2.2 Boolesche Algebra

Funktionen $\omega \in \mathbb{B}_2$ definieren *Operationen* auf $\mathbb{B}_n$:
Seien $f,\ g \in \mathbb{B}_n$;

$$\omega(f,g)(x_1,\ldots,x_n) := \omega\left(f(x_1,\ldots,x_n), g(x_1,\ldots,x_n)\right).$$

Definition 1.6.
Eine Menge A zusammen mit zwei binären Operatoren $+, *$ und einer unären Operation $\overline{}$ heißt *Boolesche Algebra*, falls für alle $x,y,z \in A$ die folgenden Rechenregeln erfüllt sind:

(i) $x + y = y + x$; $x * y = y * x$ *(Kommutativität)*

(ii) $(x+y)+z = x+(y+z)$; $(x*y)*z + x*(y*z)$ *(Assoziativität)*

(iii) $x*(y+z) = (x*y)+(x*z)$; $x+(y*z) = (x+y)*(x+z)$ *(Distributivität)*

(iv) $x + x*y = x$; $x*(x+y) = x$

(v) $x + (y*\overline{y}) = x$; $x*(y+\overline{y}) = x$.

Satz 1.4.
$(\mathbb{B}_n, \vee, \wedge, \neg, 0, 1)$ *bildet eine Boolesche Algebra.* ■

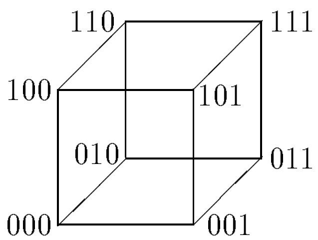
Satz 1.5. (Dualitätsprinzip)
Jede Aussage in einer Booleschen Algebra bleibt wahr, wenn man die Konstanten 0 und 1 und die Operatoren $\wedge$ *und* $\vee$ *vertauscht.*

Beweis.
Überprüfung aller Axiome. ■

1.2.3 Geometrische Veranschaulichung Boolescher Funktionen

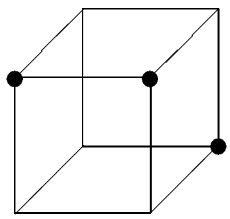
Wir fassen die Argumente $(x_1, \ldots, x_n) \in \{0,1\}^n$ als Ecken eines *n–dimensionalen Würfels* auf.

Beispiel 1.2.



Sei $f \in \mathbb{B}_n$.
Die den Argumenten aus $on(f) = f^{-1}(1)$ entsprechenden Ecken des Würfels werden markiert.

Beispiel 1.3.
$f \in \mathbb{B}_3$ $\quad on(f) = \{(100), (011), (101)\}$



Eine Funktion $f \in \mathbb{B}_{n,m}$, d. h. $f = (f_1, \dots, f_n)$, $f_i \in \mathbb{B}_n$, können wir mit m n–dimensionalen Würfeln veranschaulichen.

1.2.4 Monotone und symmetrische Boolesche Funktionen

Sei $\leq$ die komponentenweise Ordnung auf $\{0,1\}^n$.

Definition 1.7.
$f \in \mathbb{B}_n$ heißt *monoton*, falls $(x_1, \dots, x_n) \leq (y_1, \dots, y_n)$ stets $f(x_1, \dots, x_n) \leq f(y_1, \dots, y_n)$ impliziert.

Bezeichnung.
$\mathbb{M}_n$ bezeichne die Menge der monotonen Funktionen aus $\mathbb{B}_n$.

Satz 1.6.
Es gilt $\#\mathbb{M}_n \geq 2^{\binom{n}{\lfloor \frac{n}{2} \rfloor}}$.

Beweis.
Übungsaufgabe.

Hinweis:
$\binom{n}{\lfloor \frac{n}{2} \rfloor}$ ist die Zahl der monotonen Produkte der Länge $\lfloor \frac{n}{2} \rfloor$. Jede Teilmenge davon bildet eine monotone Funktion. ■

Definition 1.8.
$f \in \mathbb{B}_n$ heißt *symmetrisch*, falls für beliebige Permutationen π von $\{1, \dots, n\}$ gilt:

$$f(x_1, \dots, x_n) = f(x_{\pi(1)}, \dots, x_{\pi(n)}).$$

Bezeichnung.
$\mathbb{S}_n$ sei die Menge der symmetrischen Funktionen aus $\mathbb{B}_n$.

Satz 1.7.
Es gilt $\#\mathcal{S}_n = 2^{n+1}$.

Beweis.
Der Funktionswert symmetrischer Funktionen hängt lediglich von der Zahl der Einsen im Argument ab.
$(v_0, \ldots, v_n) \in \{0,1\}^{n+1}$ definiert eine symmetrische Funktion vermöge:

$$f(x_1, \ldots, x_n) := v_{x_1+\ldots+x_n}.$$

Umgekehrt definiert jede symmetrische Funktion auf eindeutige Weise einen Vektor $(v_0, \ldots, v_n) \in \{0,1\}^{n+1}$, den *Wertevektor* $v(f)$.
Folglich gilt: $\#\mathcal{S}_n = 2^{n+1}$. ■

Definition 1.9.
Sei $f \in \mathbb{B}_n$ symmetrisch. f heißt *Intervallfunktion*, falls gilt:

$$v(f) = (\underset{0}{0}, \ldots, 0, \underset{i}{1}, \ldots, \underset{j}{1}, 0 \ldots, \underset{n}{0}).$$

Bez.: $f = I_{i,j}$.

$f \in \mathbb{B}_n$ heißt *Schwellwertfunktion* (Thresholdfunktion), falls gilt $f = I_{k,l}$ mit $l = n$.
Bez.: $f = t_k^n \in \mathbb{B}_n$.

Es gilt: $t_k^n(x_1, \ldots, x_n) = 1$ genau dann, wenn $\sum x_i \geq k$.

1.2.5 Kanonische disjunktive und konjunktive Normalform

Wir können Boolesche Funktionen zweistufig repräsentieren.

Definition 1.10.
$\omega \in \mathbb{B}_2$ assoziativ.
Ein ω–*Monom* ist ein ω–Produkt von Literalen. Die *Länge* des Monoms ist die Zahl der enthaltenen Literale.
1 ist per Definition das leere $*$–Monom
0 ist per Definition das leere $+$–Monom

Beispiel 1.4.
$x_1 x_2 \overline{x_1}$ ist $*$–Monom

$\overline{x_1} \oplus x_3 \oplus x_3 \oplus \overline{x_1}$ ist $\oplus$–Monom

Bezeichnung.
$*$–Monome heißen kurz *Monome* (Würfel, Produkte, ...). Monome sind besonders einfach strukturierte Boolesche Funktionen.

Definition 1.11.
$(\omega, \omega'), \omega \in \mathbb{B}_2$ sei distributiv. Ein (ω, ω')– Polynom ist ein ω'–Produkt von ω–Monomen. Die *Länge* eines (ω, ω')–Polynoms ist die Summe der Länge der enthaltenen Monome.
$(*, +)$ Polynome heißen *disjunktive* NF (DNF)
$(+, *)$ Polynome heißen *konjunktive* NF (KNF)
$(*, \oplus)$ Polynome heißen *parity* NF (PNF)
Ring–Summen–Darstellungen (RSE) sind spezielle PNF.

Definition 1.12.
Sei $f \in \mathbb{B}_n$. Ein *kürzestes* (ω, ω')–Polynom für f heißt (ω, ω')–Minimalpolynom.
$S_{(\omega,\omega')}(f)$ bezeichnet die Länge eines (ω, ω')– Minimalpolynoms für f.

$$\begin{aligned} DNF(f) &:= S_{(*,+)}(f) \\ KNF(f) &:= S_{(+,*)}(f) \\ PNF(f) &:= S_{(*,\oplus)}(f) \end{aligned}$$

Definition 1.13.
Sei $a = (a_1, \ldots, a_n) \in \{0,1\}^n$; $x = (x_1, \ldots, x_n)$.
Als der *zu a gehörende Minterm* wird das $(*$–$)$Monom

$$m_a(x) := x_1^{a_1} * \ldots * x_n^{a_n}$$

bezeichnet.
Als der *zu a gehörende Maxterm* wird das $(+$–$)$Monom

$$s_a := x_1^{\overline{a_1}} + \ldots + x_n^{\overline{a_n}}$$

bezeichnet.

Lemma 1.1.
Sei $a = (a_1, \ldots, a_n) \in \{0,1\}^n$. Es gilt:
(i) $m_a(x) = 1$ genau dann, wenn für alle $1 \le i \le n$ gilt: $x_i = a_i$.
(ii) $s_a(x) = 0$ genau dann, wenn für alle $1 \le i \le n$ gilt: $x_i = a_i$.

Beweis.

$$\begin{array}{llcl} (i) & m_a(x) = 1 & \iff & \text{für alle } i \text{ gilt: } x_i^{a_i} = 1 \\ & & \iff & \text{für alle } i \text{ gilt: } x_i = a_i. \\ (ii) & s_a(x) = 0 & \iff & \text{für alle } i \text{ gilt: } x_i^{\overline{a_i}} = 0 \\ & & \iff & \text{für alle } i \text{ gilt: } x_i^{a_i} = 1 \\ & & \iff & \text{für alle } i \text{ gilt: } x_i = a_i. \quad \blacksquare \end{array}$$

Satz 1.8.
Sei $f \in \mathbb{B}_n$. Dann gilt:

$$\begin{aligned}(i)\ f(x) &= \sum_{a \in f^{-1}(1)} m_a(x),\\ (ii)\ f(x) &= \prod_{a \in f^{-1}(0)} s_a(x).\end{aligned}$$

Bis auf die Reihenfolge der Faktoren und der Summanden sind diese Darstellungen eindeutig.

Beweis.
Übungsaufgabe. ■

Definition 1.14.
Die Darstellung (i) heißt *kanonische disjunktive Normalform* (kDNF) von f.
Die Darstellung (ii) heißt *kanonische konjunktive Normalform* (kKNF) von f.

Korollar 1.1.
Die Länge einer kDNF (kKNF) ergibt sich als $n \cdot \# f^{-1}(1)$ $(n \cdot \# f^{-1}(0))$. ■

Beispiel 1.5.

$$\begin{aligned}1.)\quad f(x_1,\ldots,x_4) &= x_1\overline{x}_2x_3x_4 + \overline{x}_1x_2x_3\overline{x}_4 + \overline{x}_1x_2\overline{x}_3x_4\\ &\quad \text{Länge} : 3 \cdot 4 = 12\end{aligned}$$

$$\begin{aligned}2.)\quad f(x_1,\ldots,x_5) &= x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5\\ (&= \sum_{i=1}^{5} x_i \ \ mod\ 2) \qquad \textit{Parityfunktion}\\ kDNF: &\\ f(x_1,\ldots,x_5) &= x_1\overline{x}_2\overline{x}_3\overline{x}_4\overline{x}_5 + \overline{x}_1x_2\overline{x}_3\overline{x}_4\overline{x}_5 + \cdots + \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4x_5\\ &\quad +x_1x_2x_3\overline{x}_4\overline{x}_5 + \cdots + \overline{x}_1\overline{x}_2x_3x_4x_5\\ &\quad +x_1x_2x_3x_4x_5\end{aligned}$$

Länge: Zahl der Monome $\binom{5}{1} + \binom{5}{3} + \binom{5}{5} = 16$; Länge $16 \cdot 5 = 80$.

Man kann eine kDNF (kKNF) aus der Tabellendarstellung von f ablesen:

Beispiel 1.6.

x_1	x_2	x_3	$f(x_1,x_2,x_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$$\begin{aligned}
\text{kDNF: } f(x_1,x_2,x_3) &= \overline{x}_1\overline{x}_2x_3 + x_1\overline{x}_2\overline{x}_3 + x_1x_2\overline{x}_3\\
\text{kKNF: } f(x_1,x_2,x_3) &= (x_1+x_2+x_3)(x_1+\overline{x}_2+x_3)(x_1+\overline{x}_2+\overline{x}_3)\cdot\\
&\quad \cdot(\overline{x}_1+x_2+\overline{x}_3)(\overline{x}_1+\overline{x}_2+\overline{x}_3)
\end{aligned}$$

Andererseits kann man die kDNF mit dem Shannon'schen Entwicklungssatz gewinnen:

Satz 1.9. (Shannon'scher Entwicklungssatz)
Sei $f \in \mathbb{B}_n$. Für alle $1 \leq i \leq n$ gilt:

$$f(x_1,\ldots,x_n) = x_i f(x_1,\ldots,x_{i-1},1,x_{i+1},\ldots,x_n) + \overline{x}_i f(x_1,\ldots,x_{i-1},0,x_{i+1},\ldots,x_n).$$

Beweis.
Trivial:

$$\begin{aligned}
\text{Gilt } x_i = 1, \text{ dann ist die rechte Seite } &= f(x_1,\ldots,x_{i-1},1,x_{i+1},\ldots,x_n)\\
&= f(x_1,\ldots,x_n).\\
\text{Gilt } x_i = 0, \text{ dann ist die rechte Seite } &= f(x_1,\ldots,x_{i-1},0,x_{i+1},\ldots,x_n)\\
&= f(x_1,\ldots,x_n). \quad \blacksquare
\end{aligned}$$

Beispiel 1.7.

$$f(x_1,x_2,x_3) = x_1 \oplus x_2 \oplus x_3 = (\sum x_i) \ \ mod\ \ 2 \ :$$

$$\begin{aligned}
f(x_1,x_2,x_3) &= x_1 f(1,x_2,x_3) + \overline{x}_1 f(0,x_2,x_3)\\
&= x_1x_2 f(1,1,x_3) + x_1\overline{x}_2 f(1,0,x_3) + \overline{x}_1 x_2 f(0,1,x_3)\\
&\quad +\overline{x}_1\overline{x}_2 f(0,0,x_3)\\
&= x_1x_2x_3 f(1,1,1) + x_1x_2\overline{x}_3 f(1,1,0) + x_1\overline{x}_2x_3 f(1,0,1)\\
&\quad x_1\overline{x}_2\overline{x}_3 f(1,0,0) + \overline{x}_1x_2x_3 f(0,1,1) + \overline{x}_1x_2\overline{x}_3 f(0,1,0)\\
&\quad \overline{x}_1\overline{x}_2x_3 f(0,0,1) + \overline{x}_1\overline{x}_2\overline{x}_3 f(0,0,0)\\
&= x_1x_2x_3 + x_1\overline{x}_2\overline{x}_3 + \overline{x}_1x_2\overline{x}_3 + \overline{x}_1\overline{x}_2x_3.
\end{aligned}$$

Wegen $(f*g)^{-1}(1) = f^{-1}(1) \cap g^{-1}(1)$ und $(f+g)^{-1}(1) = f^{-1}(1) \cup g^{-1}(1)$ kann eine kDNF (kKNF) für $f*g$ bzw. $f+g$ leicht aus den kDNF (kKNF) für f und g gewonnen werden.

kDNF (kKNF) sind zwar eindeutig bestimmt, aber oft existieren kürzere Darstellungen als $(*,+)$–Polynom ($(+,*)$–Polynom).

Beispiel 1.8.

$f(x_1, x_2, x_3) = x_1 + x_2$

kDNF: $f(x_1, x_2, x_3) = x_1x_2x_3 + x_1x_2\overline{x}_3 + x_1\overline{x}_2x_3 + x_1\overline{x}_2\overline{x}_3 + \overline{x}_1x_2x_3 + \overline{x}_1x_2\overline{x}_3$.

1.2.6 Ring–Summen–Entwicklung (RSE)

Erinnerung: $(\{0,1\}; \oplus, 0, *, 1)$ ist ein Körper, der mit $\mathbb{F}_2$ bezeichnet wird.

Definition 1.15.
Eine Parity-Normalform ($(*,\oplus)$–Polynom) heißt *Ring–Summen–Entwicklung* (RSE), falls alle Monome nur aus unnegierten Literalen bestehen.

Satz 1.10.
Jede Boolesche Funktion $f \in \mathbb{B}_n$ besitzt eine eindeutig bestimmte Darstellung als RSE, d. h. die RSE ist eine kanonische Darstellung.

Beweis.
Existenz:
Wir gehen von der kDNF für f aus:

$$f(x_1, \dots, x_n) = \sum_{a \in f^{-1}(1)} m_a(x).$$

An jeder Stelle $a \in f^{-1}(1)$ ($a \in f^{-1}(0)$) wird genau ein (kein) Minterm 1, d. h., an jeder Stelle $a \in f^{-1}(1)$ ($a \in f^{-1}(0)$) wird eine ungerade (gerade) Zahl von Mintermen 1 und es gilt:

$$f(x_1, \dots, x_n) = \bigoplus_{a \in f^{-1}(1)} m_a(x).$$

Nun müssen wir noch in $m_a(x)$ eventuell vorkommende negierte Variablen beseitigen. Dies ist möglich mit der Beziehung $\overline{x}_i = x_i \oplus 1$.
Durch Ausmultiplizieren (Rechnen über $\mathbb{F}_2$) erhalten wir eine RSE für f:

$$f(x_1, \dots, x_n) = \bigoplus m_I,$$

wobei $m_I := x_{i_1}, \dots, x_{i_s}$, falls $I = \{i_1, \dots, i_s\} \subseteq \{1, \dots, n\}$.

Eindeutigkeit:
Für jedes $I \subseteq \{1, \dots, n\}$ existiert genau ein m_I. Wegen $\#\{I | I \subseteq \{1, \dots, n\}\} = 2^{2^n}$ ist

$$\#\{RSE \text{ über n Variablen } \} = 2^{2^n} = \#\mathbb{B}_n.$$

Da *jede* Boolesche Funktion $f \in \mathbb{B}_n$ eine RSE besitzt, muß diese eindeutig bestimmt sein.
■

Beispiel 1.9.

$$\begin{aligned}
f(x_1, x_2, x_3) &= x_1\overline{x}_2x_3 + x_1x_2\overline{x}_3 + \overline{x}_1x_2x_3 \\
&= x_1\overline{x}_2x_3 \oplus x_1x_2\overline{x}_3 \oplus \overline{x}_1x_2x_3 \\
&= x_1(x_2 \oplus 1)x_3 \oplus x_1x_2(x_3 \oplus 1) \oplus (x_1 \oplus 1)x_2x_3 \\
&= x_1x_2x_3 \oplus x_1x_3 \oplus x_1x_2x_3 \oplus x_1x_2 \oplus x_1x_2x_3 \oplus x_2x_3 \\
&= x_1x_2x_3 \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3.
\end{aligned}$$

Kapitel 2

Boolesche Schaltkreise

2.1 Modell

Definition 2.1.
Sei$\Omega = \{\omega_i \in \mathbb{B}_{n_i} (i \in I)\}$ *Basis.*
Ein *Boolescher Ω–Schaltkreis S* (Ω–Schaltkreis) mit n Eingaben $x_1, \ldots, x_n$ ist ein endlich azyklischer, gerichteter Graph, dessen

- Knoten vom Ingrad 0 mit $0, 1, x_1, \ldots, x_n$ markiert sind; *Eingabeknoten*
- Knoten vom Ingrad n_i, $i \in I$, mit Funktionen $\omega_i \in \Omega$ markiert sind.

In jedem Knoten v von S wird in folgender Weise eine Boolesche Funktion

$$f_{S,v} : \{0,1\}^n \longrightarrow \{0,1\}$$

berechnet:
Sei $x = (x_1, \ldots, x_n) \in \{0,1\}^n$

- Ist $v \in S$ markiert mit 0 (bzw. 1), dann gilt $f_{S,v}(x_1, \ldots, x_n) = 0$ (bzw. 1).
- Ist $v \in S$ markiert mit x_i, dann gilt $f_{S,v}(x_1, \ldots, x_n) = x_i$.
- Ist $v \in S$ markiert mit w_i und sind $v_1, \ldots, v_{n_i}$ die Vorgänger von v, dann ist $f_{S,v}(x_1, \ldots, x_n) = w_i(f_{v_i}(x), \ldots, f_{v_{n_i}}(x))$.

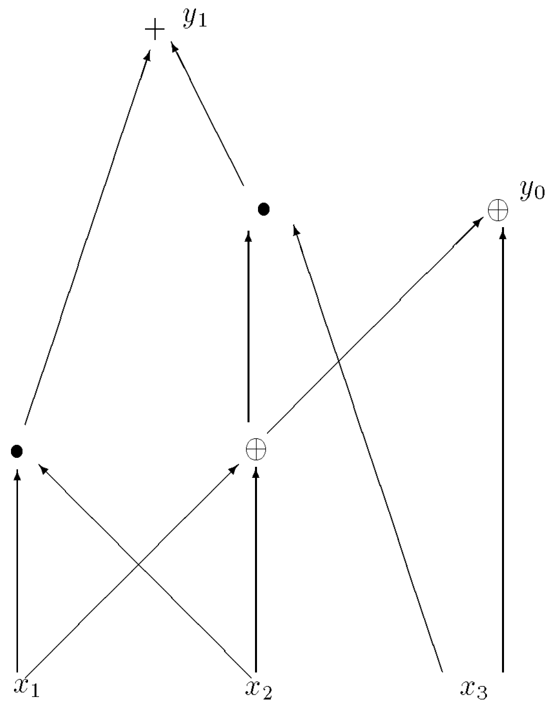
Beispiel 2.1.
DNF_S, KNF_S sind spezielle $\{\cdot, +, \neg\}$–Schaltkreise,
PNF_S sind spezielle $\{\oplus, \cdot, \neg\}$–Schaltkreise,
RSE_S sind spezielle $\{\oplus, \cdot\}$–Schaltkreise.

Definition 2.2.
$f \in \mathbb{B}_n$ wird von Ω–Schaltkreis S über $X_n = \{x_1, \ldots, x_n\}$ berechnet, wenn ein $v \in S$

existiert mit $f = f_{S,v}$.

Bemerkung.
Jede Boolesche Funktion kann mit einem $\{\cdot, +, \neg\}$–Schaltkreis berechnet werden.

Beispiel 2.2. *Volladdierer* (Addition von 3 Bits)
$f(x_1, x_2, x_3) = (y_1, y_0)$, wobei (y_1, y_0) Binärdarstellung von $x_1 + x_2 + x_3$ ist.



Definition 2.3.
Ω heißt *vollständig*, wenn jede Boolesche–Funktionen $f \in \bigcup_{n \geq 0} \mathbb{B}$ mit Hilfe eines Ω–Schaltkreises berechnet werden kann.

Beispiel 2.3.
$\{+, \cdot, \neg\}$, die *Standardbasis* (z.B. DNF, KNF) und
$\{\oplus, \cdot\}$ (RSE) sind vollständig.

Behauptung.
$\{\cdot, \neg\}$ und $\{+, \neg\}$ sind vollständig.

Beweis.
Mit Hilfe der deMorgan Regeln.

Behauptung.
$\{+, \cdot\}$ ist nicht vollständig.

Beweis.
ÜA. ■

2.2 Komplexitätsmaße für Schaltkreise

Sei S ein Ω–Schaltkreis. Wir definieren

- $C(S)$ = Anzahl der Knoten von S mit Ingrad > 0. $C(S)$ heißt *Schaltkreis–Größe von S*.
- $C_\Omega(f) = \min \{C(S) \quad | \quad S$ ist ein Ω–Schaltkreis, der f berechnet $\}$.
- $D(S)$ = Länge eines längsten Weges in S. $D(S)$ heißt *Schaltkreis–Tiefe von S*.
- $D_\Omega(f) = \min \{D(S) \quad | \quad S$ ist ein Ω–Schaltkreis, der f berechnet $\}$.

Bemerkung.
Die Schaltkreis–Größe ist Ausdruck des Hardwareaufwands zur Realisierung einer Booleschen Funktion.
Die Schaltkreis–Tiefe ist eine untere Schranke für den Zeitbedarf (auch für parallele Berechnung).

Das Schaltkreis–Modell ist robust gegenüber Änderungen der Basis.

Satz 2.1.
Seien Ω, Ω' zwei endliche vollständige Basen.

$$c := \max\{C_\Omega(g), g \in \Omega'\}, \quad d := \max\{D_\Omega(g), g \in \Omega'\}.$$

Dann gilt:

$$C_\Omega(f) \leq c \cdot C_\Omega(f) \textit{ und } D_\Omega(f) \leq d \cdot D_\Omega(f).$$

Beweis.
Betrachte Ω'–Schaltkreis S für f und ersetze in S jedes Gatter g durch einen Ω–Schaltkreis S_g für g.
Nach Voraussetzung existiert ein solcher Schaltkreis der Größe $\leq c$
Wir erhalten einen Ω–Schaltkreis S' mit

$$C(S') \leq c \cdot C(S).$$

Analog für die Tiefe. ■

2.3 Schaltkreise mit beschränktem fan–out

Betrachte Schaltkreis mit beschränktem fan–out r.

$$\begin{aligned} C_{r,\Omega}(f) &= \min\{C(S) \quad | \quad S \text{ ist } \Omega\text{–Schaltkreis mit fan–out } r \text{ für } f\}, \\ D_{r,\Omega}(f) &= \min\{D(S) \quad | \quad S \text{ ist } \Omega\text{–Schaltkreis mit fan–out } r \text{ für } f\}. \end{aligned}$$

Satz 2.2.
Seien $C_\Omega(f)$, $D_\Omega(f)$ definiert. Dann gilt

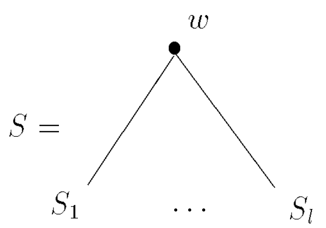
$$C_\Omega(f) \leq \ldots \leq C_{r+1,\Omega}(f) \leq C_{r,\Omega}(f) \leq \ldots \leq C_{1,\Omega}(f)$$

$$D_\Omega(f) = D_{r,\Omega}(f) = D_{1,\Omega}(f).$$

Beweis.
Die Aussage für die Schaltkreisgröße gilt nach Definition.
Für die Tiefe ist zu zeigen: für jeden Ω–Schaltkreis S mit fan–out r existiert ein Ω–Schaltkreis S' mit fan–out $\leq r-1$ und

$$D(S') \quad = \quad D(S).$$

Vollständige Induktion über $t = D(S)$:
$t = 0$: trivial
$t \to t+1$:

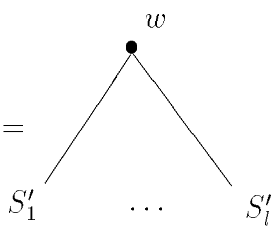


S_i berechne in Wurzel $g_i, 1 \leq i \leq l$.
Es gilt: $D(S_1), \ldots, D(S_l) \leq t$
Folglich existieren $S'_1, \ldots, S'_l$ mit fan–out $\leq r-1$ für $g_1, \ldots, g_l$ mit

$$D(S'_i) = D(S_i).$$

Betrachte

$S' =$



Offenbar hat S' fan–out $\leq r-1$ und es gilt

$$D(S') = D(S).$$

Achtung.
Die Schaltkreis–Größe kann bei der beschriebenen Prozedur explodieren.

Durch geschickte Simulation kann folgendes erreicht werden:

Satz 2.3.
Sei k fan–in der Basis Ω (= max fan–in eines $\omega \in \Omega$).
Ist f durch einen Ω–Schaltkreis berechenbar und ist $s \geq 2$, dann gilt:

$$C_{s,\Omega}(f) \leq (1 + 2\frac{k-1}{s-1}) \cdot C_\Omega(f). \quad \blacksquare$$

Achtung. Satz gilt nicht für $s = 1$.

Definition 2.4.
Ein Ω–Schaltkreis mit fan–out 1 heißt *Boolesche Ω–Formel.*
$L_\Omega(f) := C_{1,\Omega}(f)$ heißt *Ω–Formelgröße* von f.

2.4 Vergleich verschiedener Ω–Schaltkreise

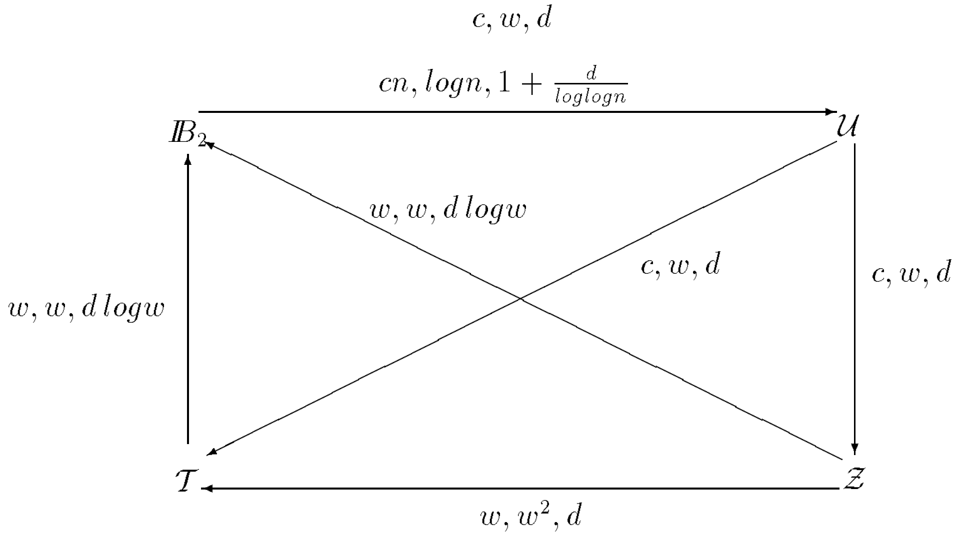
Wir betrachten die folgenden vollständigen Basen:

$$\begin{array}{lll} \mathbb{B}_2, & & \\ \mathcal{U} & := & \{\neg, *_i, +_i \,|\, i \in \mathbb{N}\}, \\ \mathcal{Z} & := & \{*_i, \oplus_i \,|\, i \in \mathbb{N}\}, \\ \mathcal{T} & := & \{t^n_k, t^n_{\leq k} \,|\, n, k \in \mathbb{N}\}. \end{array}$$

Wir wollen $\mathbb{B}_2$–Schaltkreise, $\mathcal{U}$–Schaltkreise, $\mathcal{Z}$–Schaltkreise und $\mathcal{T}$–Schaltkreise miteinander vergleichen in bezug auf:

- c – Zahl der Gatter (complexity)
- w – Zahl der Leitungen (wires)
- d – Tiefe (depth)

Satz 2.4.



(Übung: $c\,logw, w\,logw, d\,logw$)

Erklärung:

$$\Omega_1 \overset{f(c),g(w),h(d)}{\longrightarrow} \Omega_2$$

bedeutet Ω_1–Schaltkreis mit c, w, d kann simuliert werden durch Ω_2–Schaltkreis mit $O(f(c))$, $O(g(w))$, $O(h(d))$.

Beweis.

1. $I\!B_2 \overset{c,w,d}{\longrightarrow} \mathcal{U}$
   Wir können jedes $I\!B_2$–Gatter (mittels DNF) durch $\mathcal{U}$–Schaltkreis simulieren.

2. $I\!B_2 \overset{cn,cn\log n,1+\frac{d}{\log\log n}}{\longrightarrow} \mathcal{U}$
   d.h. $I\!B_2$–Schaltkreise können durch schnelle $\mathcal{U}$–Schaltkreise polynomialer Größe simuliert werden.
   O.B.d.A.: $n \geq 4$.

- Berechne parallel zu jedem Gatter g in $\mathbb{B}_2$–Schaltkreis mit $\overline{g}$ die negierte Funktion.
  Es ergibt sich eine Verdopplung der Schaltkreis-Größe
- Zerlege $\mathbb{B}_2$–Schaltkreis mit $l = \lfloor \log\log n \rfloor$ in $\lceil \frac{d}{l} \rceil$ Schichten, so daß die i–te Schicht alle Gatter g der Tiefe $d(g)$ mit $(i-1)l < d(g) \leq i \cdot l$ enthält.
- Simuliere jede Schicht des $\mathbb{B}_2$–Schaltkreises durch $\mathcal{U}$–Schaltkreis der Tiefe 2:
  Verfolge für jedes Gatter g in $\mathbb{B}_2$–Schaltkreis Rechenpfade zurück bis auf Eingaben oder Gatter früherer Schichten;
  da $\mathbb{B}_2$–Schaltkreise den fan–in 2 haben,hat g höchstens $2^l \leq \lfloor \log n \rfloor$ Vorgänger
  Berechne die an g berechnete Funktion durch DNF oder KNF in den Vorgängern (brauchen keine Negation, da alle Funktionen auch negiert berechnet werden).
  Damit wird die i–te Schicht durch $\mathcal{U}$–Schaltkreis der Tiefe 2 simuliert. Werden DNF und KNF abwechselnd verwendet, können zwei Ebenen benachbarter Schichten verschmolzen werden.
  Jedes $\mathbb{B}_2$–Gatter wird durch $\leq n+1$ $\mathcal{U}$–Gatter und $n(\lfloor \log n \rfloor + 1)$ Leitungen ersetzt. ■

3. $\mathcal{U} \overset{c,w,d}{\longrightarrow} \mathcal{Z}$

   - Negiere im $\mathcal{U}$–Schaltkreis zusätzlich alle Gatter.
   - Ersetze Negation durch $\oplus$–Addition mit 1.
   - Beseitige Disjunktion mit fan–in m mit Hilfe der de Morgan Regeln
     $y_1 + \ldots + y_m = [(y_1 \oplus 1) \cdot \ldots \cdot (y_m \oplus 1)] \oplus 1$
   - Da alle $y_i \oplus 1$ berechnet werden, wachsen Größe, Leitungszahl und Tiefe nur um konstante Faktoren.

4. $\mathcal{U} \overset{c,w,d}{\longrightarrow} \mathcal{T}$
   Trivial: $\mathcal{U}$–Schaltkreise sind $\mathcal{T}$–Schaltkreise.

5. $\mathcal{Z} \overset{w,w,d\log w}{\longrightarrow} \mathbb{B}_2$
   Wir können $*_m$ und $\oplus_m$ durch $\mathbb{B}_2$–Schaltkreis mit $m-1$ Gattern der Tiefe $\lceil \log m \rceil$ simulieren (ausgeglichener Baum).

6. $\mathcal{Z} \overset{w,w^2,d}{\longrightarrow} \mathcal{T}$
   Wir müssen lediglich $\oplus_m$–Gatter simulieren:

   Lemma 2.1.
   Sei $f \in \mathcal{S}_n$. Dann kann f durch $\mathcal{T}$–Schaltkreis der Tiefe 2 mit $I(f)$ Gattern und $(n+1)(I(f)-1)$ Leitungen simuliert werden, wobei $I(f)$ die Zahl der nicht verlängerbaren konstanten Teilvektoren aus $v(f) = (v_0, \ldots, v_n)$ (Wertevektor von f) ist.

Beweis.
O.B.d.A.: $v_0 = 0$
(denn f und $\overline{f}$ haben gleiche $\mathcal{T}$–Schaltkreis–Größe)

- Wir berechnen für jeden maximalen Teilvektor $v_{i-1} = 0, v_i = \ldots = v_j = 1$, $v_{j+1} = 0$ bzw. $j = n$ in Tiefe 1

$$t_i^n(x) \text{ und } t_{\leq j}^n(x).$$

  Konstruktion liefert stets eine 1 und genau 2 Einsen, falls $i \leq \|x\| \leq j$.
  Folglich können mit $2\lfloor\frac{I(f)}{2}\rfloor$ $\mathcal{T}$–Gattern $g_1, \ldots, g_{2r}$ in Tiefe 1 die $r := \lfloor\frac{I(f)}{2}\rfloor$ maximalen 1–Teilvektoren berechnet werden.
  Die Funktionen $g_1, \ldots, g_{2r}$ berechnen mindestens r Einsen.
  $(r+1)$–Einsen werden berechnet genau dann, wenn $f(x) = 1$.
  Also $f = t_{r+1}^{2r}(g_1, \ldots, g_{2r})$.
  D.h. f kann mit $2r+1$ $\mathcal{T}$–Gattern in Tiefe 2 berechnet werden.

- Zahl der Leitungen:
  $n(I(f)-1)$ Leitungen führen zu Gattern der Tiefe 1.
  $I(f)-1$ Leitungen führen zu Gattern der Tiefe 2. ■

Dem Lemma zufolge, können wir $\oplus_m \in \mathcal{S}_m$ durch $\mathcal{T}$–Schaltkreis mit $\leq m+1$ Gattern und $\leq m(m+1)$ Leitungen simulieren.

7. $\mathcal{T} \overset{w,w,dlogw}{\longrightarrow} \mathbb{B}_2$

Satz 2.5.
Jede symmetrische Funktion $f \in \mathcal{S}_n$ läßt sich mit $\mathbb{B}_2$–Schaltkreis der Größe $O(n)$ und der Tiefe $O(logn)$ berechnen.

Beweis.

(a) $v(f)$ oder $v(\overline{f})$ hat höchstens $\frac{n+1}{2}$ Einsen.
Folglich kann f oder $\overline{f}$ als Disjunktion von $\leq \frac{n+1}{2}$ Mintermen über $s = (s_{m-1}, \ldots, s_0)$ berechnet werden, wobei s die binäre Summe von $m+1$ Bits ist, $m = \log(n+1)$.

(b) **Lemma 2.2.**
Jede Menge von l Mintermen auf $y_1, \ldots, y_m$ läßt sich mit $\mathbb{B}_2$–Schaltkreis der Größe $l + O(2^{\frac{m}{2}})$ und der Tiefe $\log m$ berechnen.
Beweis.
<u>Aussage über die Tiefe :</u>
Wir berechnen alle Minterme auf $y_1, \ldots, y_{\lceil\frac{m}{2}\rceil}$ und alle Minterme auf $y_{\lceil\frac{m}{2}\rceil+1}, \ldots, y_m$ parallel.

Wir können dann alle Minterme auf $y_1, \ldots, y_m$ mit einem $*$–Gatter berechnen, womit sich die Aussage über Tiefe ergibt.
Aussage über die Größe:
Für $m \leq 5$ sind einzelne Fälle zu betrachten.
Für $m \geq 6$ kann induktiv geschlossen werden:

$$\begin{aligned} & 2^m + 2 \cdot 2^{\lceil \frac{m}{2} \rceil} + 2 \cdot 2^{\lfloor \frac{m}{2} \rfloor} \\ \leq \; & 2^m + 2 \cdots 2^{(m+1)/2} + 2 \cdots 2^{(m-1)/2} \\ = \; & 2^m (1 + 2^{\frac{-m}{2}} (\underbrace{2 \cdot 2^{\frac{1}{2}} + 2 \cdot 2^{\frac{-1}{2}}}_{\leq 1})) \\ \leq \; & 2 \cdot 2^m . \end{aligned}$$

Folglich können wir jede Menge von l Mintermen auf $y_1, \ldots, y_m$ mit $\mathbb{B}_2$–Schaltkreis der Größe $l + O(2^{\frac{m}{2}})$ und der Tiefe $\lceil \log m \rceil$ berechnen.

(c) **Lemma 2.3.**
Die binäre Summe

$$s = (s_{n-1}, \ldots, s_0) = \sum_{i=1}^{n} x_i$$

läßt sich mit $\mathbb{B}_2$–Schaltkreis der Größe $O(n)$ und der Tiefe $O(\log n)$ berechnen.

Beweis.
Übung ■

Es folgt die Simulation:
Jedes Thresholdgatter mit fan–in m kann durch ein $\mathbb{B}_2$–Schaltkreis der Größe $O(m)$ und der Tiefe $O(\log m)$ simuliert werden.

2.5 Schaltkreiskomplexitätsklassen

Definition 2.5.
Eine *Schaltkreisfamilie* $S = (S_n)$ ist eine Folge von Schaltkreisen, wobei S_k k Eingänge besitzt.
S hat die *Größe* $S(n)$ bzw. die *Tiefe* $d(n)$, falls S_i die Größe $s(i)$ bzw. die Tiefe $d(i)$ besitzt.

Definition 2.6.
$S = (S_n)$ *erkennt eine Sprache* $L \subseteq \{0,1\}^*$, genau dann, wenn S_n die charakteristische Funktion von $L^n = L \cap \{0,1\}^n$ berechnet,

$$S_n = \chi_{L^n} \text{ wobei } \chi_{L^n}(x) = \begin{cases} 1 & x \in L^n \\ 0 & \textit{sonst.} \end{cases}$$

Definition 2.7. *Schaltkreis–Komplexitätsklassen*

$$\begin{array}{lcl} DEPTH_\Omega(d(n)) & = & \{L : L \text{ wird von einer } \Omega\text{–Schaltkreis–Familie der} \\ & & \text{Tiefe } O(d(n)) \text{ erkannt}\}; \\ SIZE_\Omega(s(n)) & = & \{L : L \text{ wird von einer } \Omega\text{–Schaltkreis–Familie der} \\ & & \text{Größe } O(s(n)) \text{ erkannt}\}; \\ SIZE-DEPTH_\Omega(s(n),d(n)) & = & \{L : L \text{ wird von einer } \Omega\text{–Schaltkreis–Familie erkannt,} \\ & & \text{die von der Größe } O(s(n)) \text{ und der Tiefe } O(d(n)) \\ & & \text{ist}\}. \end{array}$$

Korollar 2.1.
Seien Ω, Ω' zwei endliche vollständige Basen. Dann gilt:

$$\begin{array}{rcl} SIZE_\Omega(s(n)) & = & SIZE_{\Omega'}(s(n)), \\ DEPTH_\Omega(d(n)) & = & DEPTH_{\Omega'}(d(n)), \\ SIZE, DEPTH_\Omega(s(n),d(n)) & = & SIZE, DEPTH_{\Omega'}(s(n),d(n)). \end{array}$$

Beweis.
Folgt aus Simulationsresultat aus $II.2$. ∎

Vereinbarung:
Wir betrachten im Falle endlicher Basen stets $\mathbb{B}_2$–Schaltkreise. Wir sprechen dann von Booleschen Schaltkreisen oder Schaltkreisen mit beschränktem fan–in.

Spezielle Klassen:

- $\mathcal{P}_{Schaltkreis} = \bigcup\limits_{k\geq 0} SIZE(n^k)$.
  Wir wissen bereits: $P \subseteq \mathcal{P}_{Schaltkreis}$
- $NC_i := \bigcup\limits_{k\geq 0} SIZE, DEPTH(n^k, \log^i n)$,
  $NC = \bigcup\limits_{i\geq 0} NC_i$ (*Nick's* Class, Nick Pippenger 1979)
  Es gilt:

  $$NC_1 \subseteq NC_2 \subseteq \ldots \subseteq NC_k \subseteq NC_{k+1} \subseteq \ldots \subseteq NC \subseteq \mathcal{P}_{Schaltkreis}$$

  NC gilt als Klasse der effizienten parallelen Algorithmen,
  P gilt als Klasse der effizienten sequentiellen Algorithmen.

- $AC_i := \bigcup_{k\geq 0} SIZE, DEPTH_{\mathcal{U}}(n^k, \log^i n)$
  $AC = \bigcup_{i\geq 0} AC_i$ (*Alternating* Class)
  $AC_{o,d} = \bigcup_{k\geq 0} SIZE, DEPTH_{\mathcal{U}}(n^k, \text{“}d\text{”})$. Tiefe $\leq d$, nicht $O(d)$!
  Es gilt:

  $AC_{0,1} \subseteq AC_{0,2} \subseteq \ldots \subseteq AC_{0,d} \subseteq AC_{0,d+1} \subseteq \ldots \subseteq AC_0 \subseteq \ldots \subseteq AC_k \subseteq AC_{k+1} \subseteq \ldots \subseteq AC$.

  Satz 2.6.

  1. $NC_k \subseteq AC_k$,
  2. $AC_k \subseteq NC_{k+1}$,
  3. $AC = NC$

  Beweis.

  1. Jede Funktion aus $\mathbb{B}_2$ kann von $\mathcal{U}$–Schaltkreis in konstanter Größe und Tiefe berechnet werden.
     Folglich kann jeder NC_k–Schaltkreis in einen AC_k–Schaltkreis überführt werden.

  2. Sei $S = (S_n)$ eine Folge von AC_k–Schaltkreisen.
     Wir simulieren $*$ und $+$ Gatter mit fan–in m durch $\mathbb{B}_2$–Bäume der Größe $m-1$ und Tiefe $\log m$.
     Da S_n von polynomialer Größe $p(n)$ ist, folgt $m \in p(n)$ für alle Gatter von S_n.
     Der simulierende $\mathbb{B}_2$–Schaltkreis hat also Größe $\leq p(n)^2$ und Tiefe $O(\log^k(n) \cdot \log(n)) = O(log^{k+1} n)$.
  3. folgt aus a) und b). ■

  Es gilt also:
  $AC_{0,d} \subseteq \ldots \subseteq AC_0 \subseteq NC_1 \subseteq AC_1 \subseteq \ldots \subseteq NC = AC$.

- $$\begin{aligned} ZC_i &= \bigcup_{k\geq 0} SIZE, DEPTH_{\mathcal{Z}}(n^k, \log^i n), \\ ZC_{0,d} &= \bigcup_{k\geq 0} SIZE, DEPTH_{\mathcal{Z}}(n^k, d), \\ ZC &= \bigcup_{i\geq o} ZC_i. \end{aligned}$$

-
$$\begin{aligned} TC_i &= \bigcup_{k\geq 0} SIZE, DEPTH_{\mathcal{T}}(n^k, \log^i n), \\ TC_{0,d} &= \bigcup_{k\geq 0} SIZE, DEPTH_{\mathcal{T}}(n^k, d), \\ TC &= \bigcup_{i\geq o} TC_i. \end{aligned}$$

2.6 Simulation von Turing-Maschinen durch Schaltkreise

Sei M eine Turing-Maschine, die L in Zeit $t(n)$ entscheidet.
Wir konstruieren einen Schaltkreis S_n für f_n mit

$$f_n^{-1}(1) = L^n := L \cap \{0,1\}^n$$

von möglichst geringer Größe bzgl. $t(n)$.

1. Schritt:
Vereinfachte TM–Modelle: Oblivious TM
(Problem: Simulation der Kopfbewegung)

Definition 2.8.
Eine TM M ist *oblivious* ("lesenormiert"), wenn die Folge der Kopfbewegungen nur von der Länge der Eingabe und nicht von der Eingabe selbst abhängt.

Betrachte eine TM mit einem Band, auf dem die Felder $0, \dots, n-1$ die Eingabe darstellen und auf dem Rest gearbeitet wird.

Satz 2.7.
Wenn L durch eine oblivious TM in der Zeit $t(n)$ entschieden werden kann, dann gibt es einen Schaltkreis der Größe $O(t(n))$, der f_n mit $f_n^{-1} = L^n$ berechnet.

Beweis.
Sei $pos(t, n) :=$ Position des Kopfes nach $t \leq t(n)$ Takten für eine Eingabe der Länge n.
Die t–te Konfiguration einer oblivious TM M bei Eingabe x ist eindeutig beschrieben durch

$$\begin{array}{rcll} q(t,x) & \in & Z & \text{Zustand} \\ b(t,x,j) & \in & \Gamma, & -t(n) \leq j \leq t(n) \text{ j-tes Symbol der Bandinschrift} \end{array}$$

Wir kodieren Z bzw. Γ binär durch 0–1–Folgen der Länge $\log \sharp Z$, bzw.$\log \sharp\Gamma$
Startkonfiguration:

$$\begin{aligned} q(0,x) &= q_S, \\ b(0,x,j) &= \begin{cases} x_j & 0 \le j \le n-1, \\ \square & \text{sonst.} \end{cases} \end{aligned}$$

Ausgabe:

$$q(t(n),x) \in \{q_0, q_1\}.$$

Zustandsüberführungsfunktion:

$$\begin{aligned} \delta &: Z \times \Gamma \cup \{\square\} \to Z \times \Gamma \times \{L,R,S\}, \\ \delta_i &= pr_i(\delta) \quad \text{(Projektion)}, \\ q(t+1,x) &= \delta_1(q(t,x), b(t,x,pos(t,n))), \end{aligned}$$

$$b(t+1,x,j) = \begin{cases} b(t,x,j) & \text{für } j \neq pos(t,n), \\ \delta_2(q(t,x), b(t,x,pos(t,n))) & \text{sonst.} \end{cases}$$

δ ist eine endliche Funktion.
Folglich können wir δ durch Schaltkreis S_δ konstanter Größe $O(1)$ berechnen.
Da die Position $pos(t,n)$ im voraus bekannt ist, kann M mit $t(n)$ Kopien von S_δ simuliert werden. ■

2.Schritt:

Satz 2.8.
Eine TM M mit der Zeitkomplexität $t(n)$ läßt sich durch eine oblivious TM M' mit der Zeitkomplexität $O(t^2(n))$ simulieren.

Beweis.
M' simuliert M Schritt für Schritt.
Idee: Wir verschieben die Bandinschrift, so daß jeder Schritt lediglich das 0–te Feld verändert.

Kopfbewegung nach rechts wird simuliert durch Bewegung des gesamten Bandes nach links, Zeit $O(t(n))$
Kopfbewegung nach links wird simuliert durch Bewegung des gesamten Bandes nach rechts, Zeit $O(t(n))$.
Als Zeitbedarf ergibt sich insgesamt $(O(t^2(n))$. ■

Bessere Simulation von Fischer und Pippenger (73,79):

Satz 2.9.
Eine TM M mit der Zeitkomplexität $t(n)$ läßt sich durch eine oblivious TM M' mit der Zeitkomplexität $O(t(n) \log t(n))$ simulieren. ■

Theorem 2.1.
Eine TM M mit der Zeitkomplexität $t(n)$ kann von einer Schaltkreis–Familie der Größe $O(t(n) \log t(n))$ simuliert werden.

Genauere Betrachtungen zeigen:

Korollar 2.2.
$t(n)$–zeitbeschränkte und $s(n)$–raumbeschränkte TM können von Schaltkreis–Familien der Größe $O(t(n) \log s(n))$ simuliert werden. ■

Korollar 2.3.
Jedes Problem mit einem polynomialen Lösungsalgorithmus kann von Schaltkreis–Familien polynomialer Größe berechnet werden. ■

Wir wollen nun die Tiefe der simulierten Schaltkreis–Familien verringern:

Theorem 2.2. *(Borodin 77)*
Eine TM mit Raumkomplexität $s(n), s(n) \geq \log n$ kann von Schaltkreis–Familien der Tiefe $O(s(n)^2)$ simuliert werden.

Beweis.
Sei $k(n)$ die Zahl der verschiedenen Konfigurationen bei Eingabe der Länge n. Dann gilt

$$\log k(n) \leq \underbrace{\log \sharp Z}_{\text{Zustand}} + \underbrace{\log n}_{\text{Eingabeband}} + s(n) \underbrace{\log \sharp \Gamma}_{\text{Bandinschrift}} + \underbrace{\log s(n)}_{\text{Kopfstellung}} .$$

Für die Zahl $t(n)$ der Rechenschritte gilt $t(n) \leq k(n)$ (sonst Zyklus), also $\log t(n) \leq \log k(n) = O(s(n))$.
<u>Betr.:</u> $k(n) \times k(n)$ Matrix $A = A(x) = (a_{K,K'})$, K, K' Konfigurationen mit

$$a_{k,k'} = \begin{cases} 1 & \text{falls K' Nachfolgekonfiguration von K bei Eingabe von x} \\ 0 & \text{sonst.} \end{cases}$$

Wir können A in Tiefe 1 (bzgl. Standardbasis) berechnen, da $a_{K,K'}$ nur von einem Bit von $x = (x_1, \ldots, x_n)$ abhängt.
$A^i = (a^{(i)}_{K,K'})$: i–te Potenz von A bzgl. Boolescher Matrizenmultiplikation.
Es gilt:

$$a^{(i)}_{K,K'} = \sum_{K''} a^{(i-1)}_{K,K''} \cdot a_{K'',K'}.$$

Es gilt $a^{(i)}_{K,K'} = 1$ genau dann, wenn M in Konfiguration K gestartet, bei Eingabe von x in $\leq i$ Takten die Konfiguration K' erreicht.
Folglich ist

$$f_n(x) = \sum a^{t(n)}_{K_0,K},$$

wobei K_0 die Startkonfiguration und K die akzeptierende Konfiguration ist.
Wir können $A^{t(n)}$ mit $\log t(n)$ Booleschen Matrizenmultiplikationen berechnen. Jede Multiplikation benötigt Tiefe $\log k(n) = O(s(n))$.
Die simulierenden Schaltkreise haben also die Tiefe $O(\log t(n) \cdot s(n)) \leq O(s(n)^2)$. ■

Korollar 2.4.
Jedes Problem mit einem Lösungsalgorithmus der Speicherkomplexität $O(\log n))$ kann von einer Schaltkreis–Familie der Tiefe $O((\log n)^2)$ berechnet werden.

2.7 Simulation von Schaltkreisen durch Turing–Maschinen

- Wir können jede Funktion $f_n \in \mathbb{B}_n$ in konstanter Zeit von TM M berechnen lassen. M hat 2^n Zustände und geht in akzeptierenden Zustand, falls $x \in f_n^{-1}(1)$.

- Wir wollen aber Funktionenfolgen ("Entscheidungsprobleme") $f = (f_n)_{n\in\mathbb{N}}$ mit $f_n \in \mathbb{B}_n$ von TM berechnen bzw. die zugehörigen Sprachen

  $$L_f \subseteq \{0,1\}^* \text{mit } \omega \in L \Leftrightarrow f_{|\omega|}(\omega) = 1$$

  berechnen lassen.

- Problem: L_f muß nicht rekursiv sein!

  Beispiel 2.4.
  Sei $A \subseteq \mathbb{N}$ nicht rekursiv und $f = (f_n)$ mit

  $$f_n(x) = \begin{cases} 0 & n \in A \\ 1 & n \notin A \end{cases}$$

  Es existiert keine TM, die L_f erkennen kann !!

- Hintergrund: TM ist ein endliches Berechnungsmodell, Schaltkreis–Familien aber sind unendliches Berechnungsmodell.

1. Ausweg

Betrachte eingeschränkte Schaltkreis–Familie (S_n).

(S_n) ist *uniform*, genau dann, wenn $1^n \longrightarrow \tilde{S_n}$ kann mit beschränkten Ressourcen berechnet werden ($\tilde{S_n}$: Kodierung von S_n).

Satz 2.10.
Es gilt $P = uniform\,\mathcal{P}_{Schaltkreis}$. ■

2. Ausweg

Betrachte erweiterte TM:

Definition 2.9.
Eine *nicht–uniforme TM* ist eine TM mit zusätzlichem Leseband ("Orakelband"), das bei Eingabe von Inputs der Länge n mit einem als "Advice" bezeichneten Wort α_n beschrieben ist.

<u>Komplexitätsfunktion:</u>

$t(n)$:	Zahl der Rechenschritte bei Eingabe der Länge n im worst–case
$s(n)$:	Zahl der besuchten Zellen im worst–case auf dem Arbeitsband und $\log \lvert\alpha_n\rvert$ (Stellung des Kopfes auf dem Orakelband).

Satz 2.11.
Sei $f = (f_n)$ *mit* $c_n = C(f_n) = \Omega(n)$ *und* $L = L_f$. *L kann von einer nicht–uniformen TM in Zeit* $t(n) = O(c_n^2)$ *und Speicher* $s(n) = O(c_n)$ *entschieden werden.*

Beweis.
Sei α_n die Kodierung eines optimalen Schaltkreis S_n für f_n.
Wir numerieren Eingabe und Gatter von S_n durch $1, \ldots, n, n+1, \ldots, n+c_n$.
Wir kodieren die Gatter durch:

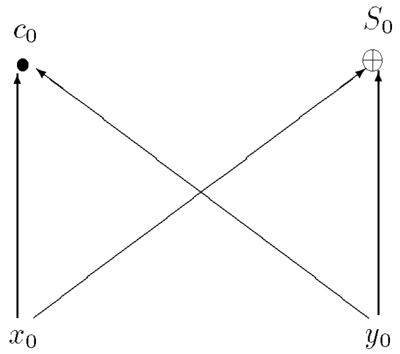
- Nummer der Gatter,
- Nummer des Gattertyps und
- Nummern der direkten Vorgänger.

Die Kodierung eines Gatters hat die Länge $O(\log c_n)$.
Die Kodierung von S_n hat die Länge $O(c_n \cdot \log c_n)$,
also $\log |\alpha_n| = O(\log c_n)$.
Wir können nun S_n Schritt für Schritt simulieren:

Nach Simulation von Gatter $i-1$ werden die Ergebnisse auf das Arbeitsband geschrieben.
Das i–te Gatter kann nun simuliert werden:
Die Eingaben stehen auf dem Arbeitsband oder dem Eingabeband, die Informationen, über die zu betrachtenden Vorgänger, stehen auf dem Orakelband.
Die Simulation des i–ten Gatters benötigt die Zeit $O(c_n)$,
die Simulation des i–ten Gatters benötigt den Speicher $O(c_n)$ für das Resultat und $O(\log c_n)$ zum Zählen. ■

Satz 2.12.
Sei $f = (f_n)$ mit $d_n = D(f_n) = \Omega(\log n)$ und $L = L_f$. L kann von einer nicht–uniformen TM in der Zeit $t(n) = O(n2^{d_n})$ und dem Speicher $s(n) = O(d_n)$ entschieden werden.

Beweis.
Sei α_n Kodierung einer tiefenoptimalen *Formel* F_n für f_n.
Wir können die Formel sehr effizient simulieren, da die Ausgabe jedes Gatters höchstens einmal genutzt wird.
Numeriere Gatter von F_n in *postorder*:



$$postorder(T) := postorder(T_l)postorder(T_r)root(T)$$

Wir codieren die Gatter durch:

- ihren Typ und
- die Zahlen $i_l, i_r \in \{0, \ldots, n\}$ mit
  $i_l = 0$, falls linker Vorgänger ein anderes Gatter ist bzw.
  $i_l = j$, falls linker Vorgänger mit Variable x_j markiert ist.
  Bei i_r verfahren wir analog.

Die Kodierung jedes Gatter hat die Länge $O(\log n)$.
Es gilt $\log |\alpha_n| = O(d_n)$, da eine Formel der Tiefe d höchstens $2^d - 1$ Gatter enthält.

TM M simuliert im Orakel gegebene Formel schrittweise:
Aufgrund der Anordnung der Gatter in postorder gilt:
Das Gatter G wird simuliert, sobald sein linker und rechter Vorgänger (Unterbaum) von G simuliert sind; alle bereits genutzten Zwischenresultate sind gelöscht.
Die Eingaben für G werden folgendermaßen gefunden:

- Die Werte der Eingabevariablen werden vom Eingabeband gelesen;
- Die Resultate der Vorgängergatter sind die letzten Bits vom rechten Rand der Inschrift des Arbeitsbandes (werden nach dem Lesen gelöscht);

Die Ausgabe von G wird am rechten Ende der Inschrift des Arbeitsbandes geschrieben.

Analyse:

Zeit: Jedes Gatter kann in Zeit $O(n)$ simuliert werden.Es ergibt sich also ein Gesamtzeitbedarf von $O(n2^{d_n})$.

Raum: Es werden höchstens d_n Zwischenresultate auf dem Arbeitsband gespeichert (Induktion !).
Die Arbeit mit Zähler von 1 bis n erfordert einen Speicherbedarf von $O(\log n)$.
Folglich ergibt sich einen Gesamtspeicherbedarf von $O(d_n)$. ∎

Korollar 2.5.
Sei $T(n) = \Omega(n)$, $S(n) = \Omega(\log n)$. Dann gilt

$$\begin{array}{lll}(1) & SIZE(T^{O(1)}) & = nonuniform - TIME(T^{O(1)}), \\ (2) & DEPTH(S^{O(1)}) & = nonuniform - SPACE(S^{O(1)}).\end{array}$$

Beweis.

zu (1): Die Simulation von Schaltkreisen durch nicht-uniforme TM liefert

$$SIZE(T^k(n)) \leq nonuniform - TIME(T^{2k}(n)).$$

Die Simulation von TM durch Schaltkreise liefert

$$nonuniform - TIME(T^k(n)) \leq SIZE(T^k(n) \log T^k(n)).$$

Insgesamt erhält man (1).

zu (2): Die Simulation von Schaltkreisen durch nicht-uniforme TM liefert

$$DEPTH(S^k(n)) \leq nonuniform - SPACE(S^k(n)).$$

Die Simulation von TM durch Schaltkreise liefert

$$nonuniform - SPACE(S^k(n)) \leq DEPTH(T^{2k}(n)).$$

Somit ergibt sich (2). ∎

Wir wollen den Begriff der Nichtuniformität schärfer fassen:

Definition 2.10.
Sei F eine Klasse von Funktionen $h : \mathbb{N} \longrightarrow \{0,1\}^*$ und sei K eine TM–Komplexitätsklassen.

$$\begin{aligned} K/F \quad = \quad & \{L \subseteq \{0,1\}^* | \ \exists \text{ nichtuniforme TM } M \text{, die mit dem Orakel } h(n),\ h \in F, \\ & L \text{ in den durch } K \text{ beschriebenen Ressourcen akzeptiert}\}. \end{aligned}$$

Beispiel 2.5.

$$F = Poly = \{h : \mathbb{N} \longrightarrow \{0,1\}^* \text{ mit } |h(n)| \text{ ist durch Polynom beschränkt}\}.$$

Die gefundenen Aussagen können damit wie folgt notiert werden:

Theorem 2.3.
$\mathcal{P}_{Schaltkreis} = P/poly.$ ■

2.8 Effiziente Schaltkreise für einige wichtige Funktionen

2.8.1 Addition

$$\begin{aligned} ADD = (ADD_n) \quad & \text{mit } ADD_n \ \in \mathbb{B}_{2n,n+1} \\ ADD_n(\underbrace{x_{n-1},\ldots,x_0}_{x},\underbrace{y_{n-1},\ldots,y_0}_{y}) = & (\underbrace{s_n,\ldots,s_0}_{s}) \end{aligned}$$

wobei:
x und y Binärdarstellungen von natürlichen Zahlen,
s Binärdarstellung der Summe von x und y.

a) Schulmethode

Algorithmus:

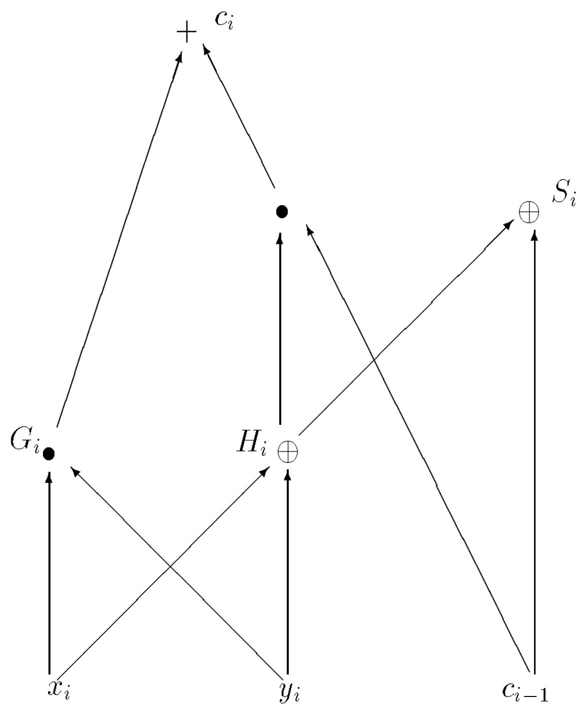
1. Addiere x_0 und y_0.
   Das Ergebnis besteht aus dem Summenbit s_0 und dem Übertragsbit (Carry) c_0.

2. Für $i = 1, \ldots, n-1$ addiere x_i, y_i und c_i.
   Das Ergebnis besteht aus dem Summenbit s_i und dem Übertragsbit c_i.

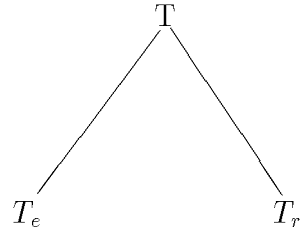
3. $s_n := c_{n-1}$

Entwerfe Schaltkreis zur Realisierung des Algorithmus:

Schritt 1 kann durch Halbaddierer mit zwei Bausteinen der Tiefe 1 realisiert werden.



Für jedes i kann Schritt 2 zwei durch Volladdierer mit fünf Bausteinen in Tiefe 3 realisiert werden.



Wir können Halbaddierer G_i, H_i parallel zu Schritt 1 rechnen lassen und erhalten, nachdem c_{i-1} zur Verfügung steht, s_i und c_i in Tiefe 2.
Gesamtgröße des Schaltkreis: $5 \cdot (n-1) + 2$,
Gesamttiefe des Schaltkreis: $2 \cdot (n-1) + 1$.

Satz 2.13.
Die Schulmethode der Addition läßt sich mit einem Schaltkreis der Größe $5n - 3$ *und der Tiefe* $2n - 1$ *realisieren.* ■

Fakt: (Redkin 81)
Es existiert kein Additions–Schaltkreis der Größe $< 5(n-1)$

Problem:
Tiefe ist bei diesem Ansatz sehr groß und deshalb ist der Schaltkreis oft nicht akzeptabel aus Zeitgründen.
Ursache: die sequentielle Berechnung der Überträge.
Ansatz: Suchen nach effizienter *paralleler Berechnung* der Überträge.

b) Carry–Look–Ahead Addierer

Wir starten mit einer Umformulierung der Schulmethode:

Algorithmus:

1. Berechnen mit n Halbaddierern
   $u_i = x_i \cdot y_i$ und $v_i = x_i \oplus y_i$ für $0 \leq i \leq n-1$.
   Größe $2n$, Tiefe 1.

2. Berechnen Carry–Bits $c_{n-1}, \ldots, c_0$ aus den Vektoren
   $u = (u_{n-1}, \ldots, u_0), v = (v_{n-1}, \ldots, u_0)$.

3. Berechne Summen–Bits:

$$\begin{array}{rcl} s_0 & = & v_0 \\ s_i & = & v_i \oplus c_{i-1}, \quad 1 \leq i \leq n-1 \\ s_n & = & c_{n-1} \end{array}$$

   Größe $n-1$, Tiefe 1.

Problem:
Berechnung von u und v in geringer Tiefe.

Betrachten (u_i, v_i):

1. Fall: $(u_i, v_i) = (0, 0)$. Dann gilt $x_i + y_i = 0$ und folglich $c_i = 0$. Die die i–te Position erreichenden Überträge werden aufgefangen. Wir sprechen hierbei von einer *E–Stelle* (eliminante).

2. Fall: $(u_i, v_i) = (0, 1)$. Dann gilt $x_i + y_i = 1$ und folglich $c_i = c_{i-1}$
Übertrag der vorherigen Stelle wird weitergeleitet. Wir sprechen hierbei von einer *P–Stelle* (propagate).

3. Fall: $(u_i, v_i) = (1, 0)$. Dann gilt $x_i + y_i = 2$ und folglich $c_i = 1$
ein neuer Übertrag wird erzeugt. Wir sprechen hierbei von einer *G–Stelle* (generate).

An Position i entsteht ein Übertrag
genau dann, wenn ein Übertrag an einer Position $j \leq i$ erzeugt und bis zur Position i weitergeleitet wird. Folglich gilt $(v_{j+1} = \ldots = v_i = 1)$,
also $c_i = \sum_{0 \leq j \leq i} u_j v_{j+1} \ldots v_i$.

- Die Produkte $u_j v_{j+1} \ldots v_i$ können mit balancierten Bäumen der Tiefe $\log n$ berechnet werden.
- Für jedes der $O(n^2)$ Summanden genügen $O(n)$ Gatter.
- Disjunktion für c_i kann in Tiefe $O(\log n)$ und Größe $O(n)$ realisiert werden.

Satz 2.14.
Carry–Look–Ahead Addierer haben die Tiefe $2 \log n + 2$ *und die polynomiale Größe* $O(n^3)$.
■

Fakt:
Die Tiefe ist asymptotisch optimal.
Die Methode führt bei unbeschränktem fan–in zu sehr geringer Tiefe:

Satz 2.15.
$ADD \in AC_{0,3}$.

Beweis.
Umsetzung der Carry–Look–Ahead Methode in $\mathcal{U}$–Schaltkreis erfordert 6 Stufen.
Durch Ineinanderschreiben der Stufen erhält man $(+, *, \neg)$ –Schaltkreis polynomialer Größe:

<u>Stufe 3:</u>
Wir berechnen
$$s_i = v_i \oplus c_{i-1} = (\overline{v_i} \cdot c_{i-1}) + (v_i \cdot \overline{c}_{i-1}).$$

(Setze $x_n = y_n = 0$ und damit $v_n = 0$ und $x_{-1} = y_{-1} = 0$ damit $c_{-1} = 0$.)
Wir brauchen lediglich noch Polynome für die $\overline{c_i}$:
Sei $w_i := \overline{x_i} \cdot \overline{y_i}$
$w_i = 1$ genau dann, wenn i ist E–Stelle. Insbesondere ist $w_{-1} = 1$.
Es gilt:

$$\overline{c_i} = \sum_{-1 \leq j \leq i} w_j \cdot v_{j+1} \dots v_i$$

Insgesamt ist

$$s_i = \sum_{0 \leq j \leq i-1} u_j \cdot v_{j+1} \dots v_{i-1} \cdot \overline{v_i} + \sum_{-1 \leq j \leq i-1} w_j \cdot v_{j+1} \dots v_{i-1} \cdot v_i$$

und

$$\begin{aligned} u_j &= x_j \cdot y_j, w_j = \overline{x_j} \cdot \overline{y_j}, \\ v_j &= x_j \oplus y_j = (\overline{x_j} + \overline{y_j})(x_j + y_j), \\ \overline{v_j} &= (x_j + \overline{y_j})(\overline{x_j} + y_j). \end{aligned}$$

Wir erhalten damit einen dreistufigen Schaltkreis:

1. Stufe: Berechnung von
$x_j + y_j$, $\overline{x_j} + y_j$, $x_j + \overline{y_j}$ und $\overline{x_j} + \overline{y_j}$.
<u>Größe:</u> $4n$ $+$–Gatter

2. Stufe: Alle $u_j v_{j+1} \dots v_{i-1} \overline{v_i}$ und $w_j v_{j+1} \dots v_{i-1} v_i$ werden berechnet.
<u>Größe:</u> $O(n^2)$ $\cdot$–Gatter

3. Stufe: Berechnung der Summenbits s_i
<u>Größe:</u> $(n+1)$ $+$–Gatter

<u>Also:</u> $ADD \in AC_{0,3}$. ■

c) Conditional Sum Addierer

Die Carry–Bit Berechnung ist schwierig.
<u>Idee:</u>
Führen Berechnung für $c_i = 0$ <u>und</u> für $c_i = 1$ aus und wählen das in der konkreten Situation jeweils richtige Ergebnis.

Definition 2.11.
Die *Auswahlfunktion* $sel \in \mathbb{B}_3$ ist definiert durch

$$sel(x, y_0, y_1) = \begin{cases} y_0 & \text{falls } x = 0, \\ y_1 & \text{falls } x = 1. \end{cases}$$

- Es gilt $sel(x, y_0, y_1) = \overline{x} y_0 + x y_1$, d.h. wir können sel in Tiefe 2 und Größe 3 realisieren.

- c_i übernimmt Auswahl, die beiden möglichen Summenbits spielen die Rolle von y_0 bzw. y_1.
  o.B.d.A.: $n = 2^k$

Algorithmus:
Addition zweier n–Bit–Zahlen x und y mit Carry–Bit $d \in \{0,1\}, n = 2^k$.

1.) Falls $k = 0$ und $d = 0$, ist $c_0 = s_1 = x_0 \cdot y_0$ und $s_0 = x_0 \oplus y_0$.
Falls $k = 0$ und $d = 1$, ist $c_0 = s_1 = x_0 + y_0$ und $s_0 = x_0 \oplus y_0 \oplus 1$.

2.) Falls $k > 1$ führe folgende Additionen parallel aus:

1. $x'' = (x_{n-1}, \ldots, x_{\frac{n}{2}})$, $\quad y'' = (y_{n-1}, \ldots, y_{\frac{n}{2}})$ und Carry–Bit 0.
   Ergebnis: $(c^0_{n-1}, s^0_{n-1}, \ldots, s^0_{\frac{n}{2}})$.
2. x'', y'' und Carry–Bit 1.
   Ergebnis: $(c^1_{n-1}, s^1_{n-1}, \ldots, s^1_{\frac{n}{2}})$.
3. $x' = (x_{\frac{n}{2}-1}, \ldots, x_0)$, $\quad y' = (y_{\frac{n}{2}-1}, \ldots, y_0)$ und Carry–Bit d
   Ergebnis: $(c^d_{\frac{n}{2}-1}, s^d_{\frac{n}{2}-1}, \ldots, s^d_0)$.

3.) Die Summe $(s_n, \ldots, s_0)$ ergibt sich wie folgt:

$$
\begin{aligned}
s_i &= s_i^d \text{ für } 0 \le i \le \tfrac{n}{2} - 1, \\
s_i &= sel(c^d_{\frac{n}{2}-1}, s_i^0, s_i^1) \text{ für } \tfrac{n}{2} \le i \le n-1, \\
c_{n-1} &= s_n = sel(c^d_{\frac{n}{2}-1}, c_i^0, c_i^1).
\end{aligned}
$$

Beispiel 2.6.

<table>
<tr><td>x</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>y</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>Stufe
0</td><td>0 1
1 0</td><td>0 1
1 0</td><td>1 0
1 1</td><td>0 1
1 0</td><td>0 0
0 1</td><td>0 1
1 0</td><td>0 1
1 0</td><td>0 0
0 1</td></tr>
<tr><td>Stufe
1</td><td colspan="2">0 1 1
1 0 0</td><td colspan="2">1 0 1
1 1 0</td><td colspan="2">0 0 1
0 1 0</td><td colspan="2">0 1 0
0 1 1</td></tr>
<tr><td>Stufe
2</td><td colspan="4">1 0 0 0 1
1 0 0 1 0</td><td colspan="4">0 0 1 1 0
0 0 1 1 1</td></tr>
<tr><td>Stufe
3</td><td colspan="8">1 0 0 0 1 0 1 1 0
1 0 0 1 0 1 1 1</td></tr>
</table>

Die Berechnungen aus 2.) können gut parallelisiert werden:

- rekursive Implementierung ist mit $O(n^{log 3})$ Bausteinen möglich,
- iterative Implementierung ist mit $O(n \cdot \log n)$ Bausteinen möglich.

Satz 2.16.
Conditional–Sum–Addierer haben für $n = 2^k$ *die Tiefe* $2logn+1$ *und die Größe* $3nlogn+10n-6$. ■

d) Ladner–Fischer–Addierer (1980)

Grundlage: *effiziente Präfixberechnung.*

Definition 2.12.
Sei $\circ$ eine assoziative binäre Operation, $(x_{n-1}, \ldots, x_0)$. Das *Präfixproblem* besteht in der Berechnung aller

$$p_i = x_i \circ \ldots \circ x_0 \text{ für } 0 \leq i \leq n-1.$$

Zwei parallele Algorithmen zur Lösung des Präfixproblems:

Algorithmus 1.

1.) Löse parallel Präfixproblem für $(x_{n-1}, \ldots, x_{\frac{2}{n}})$ und für $(x_{\frac{n}{2}-1}, \ldots, x_0)$.
Wir erhalten damit alle $p_{\frac{n}{2}-1}, \ldots, p_0$.

2.) Wir können $p_i = (x_i \circ \ldots \circ x_{\frac{n}{2}}) \circ p_{\frac{n}{2}-1}$ für jedes i mit einer Operation berechnen.

Analyse:
Algorithmus 1 löst das Präfixproblem in Tiefe $\log n$ mit $\frac{1}{2} \cdot n \cdot logn$ Operationen.

Beweis.
Der Beweis ergibt sich aus Lösung der folgenden Rekursionsgleichungen:

$$\begin{array}{lcl} D(1) = 0 & , & D(n) = D(\frac{n}{2}) + 1, \\ C(1) = 0 & , & C(n) = 2 \cdot C(\frac{n}{2}) + \frac{n}{2}. \end{array} \quad ■$$

Algorithmus 2.

1.) Berechnen parallel $\frac{n}{2}$ Paare $x_{n-1} \circ x_{n-2}, \ldots, x_1 \circ x_0$.

2.) Wenden Algorithmus rekursiv auf die $\frac{n}{2}$ Paare an. Wir erhalten alle p_i für ungerade i.

3.) Alle p_i für gerade i können parallel durch $p_i = x_i \circ p_{i-1}$ berechnet werden.

Analyse:
Algorithmus 2 löst das Präfixproblem in Tiefe $2 \cdot \log\, n - 1$ mit $2 \cdot n \cdot logn - 2$ Operationen.

Beweis.
Der Beweis ergibt sich aus Lösung der folgenden Rekursionsgleichungen:

$$\begin{array}{rclcrcl} D(2) & = & 1 & , & D(n) & = & D(\frac{n}{2}) + 2, \\ C(2) & = & 1 & , & C(n) & = & C(\frac{n}{2}) + n - 1. \quad \blacksquare \end{array}$$

Mische Algorithmus 1 und 2 um lineare Größe bei optimaler Tiefe $\log n$ zu erhalten:

Algorithmus 3.

$\boldsymbol{A_1(n)}$ Wende parallel $A_2(\frac{n}{2})$ auf $x_{\frac{n}{2}-1}, \ldots, x_0$ und $A_1(\frac{n}{2})$ auf $x_{n-1}, \ldots, x_{\frac{n}{2}}$ an.
p_i ist für $0 \leq i \leq \frac{n}{2} - 1$ berechnet.
Ist in $A_2(\frac{n}{2})$ $p_{\frac{n}{2}-1}$ berechnet und ist $A_1(\frac{n}{2})$ berechnet, dann können parallel die p_i für $\frac{n}{2} \leq i \leq n - 1$ in je einem Schritt berechnet werden.

$\boldsymbol{A_2(n)}$ Berechne parallel die $\frac{n}{2}$ Paare $x_{n-1} \circ x_{n-2}, \ldots, x_1 \circ x_0$.
Wende $A_1(\frac{n}{2})$ auf diese Paare an und berechne damit alle p_i für ungerade i, insbesondere p_{n-1}.
Die p_i für gerade i können parallel in einem Schritt berechnet werden.

Satz 2.17.
Algorithmus 3 löst das Präfixproblem in Tiefe $\log n$ *mit* $\leq 4n$ *Bausteinen.*

Beweis.
Seien $D_1(n)$ Tiefe von $A_1(n)$, $D_2(n)$ Tiefe von $A_2(n)$, $D_2^*(n)$ Tiefe, die von $A_2(n)$ zur Berechnung von p_{n-1} benötigt wird.
Dann gilt

$$\begin{aligned} D_1(1) &= D_2(1) = D_2^*(1) = 0, \\ D_1(2) &= D_2(2) = D_2^*(2) = 1. \end{aligned}$$

Für $n \geq 4$ gilt weiter

$$\begin{aligned} D_2(n) &= D_1(\tfrac{n}{2}) + 2, \\ D_2^*(n) &= D_1(\tfrac{n}{2}) + 1, \\ D_1(n) &= \max\{D_2(\tfrac{n}{2}), D_2^*(\tfrac{n}{2}) + 1, D_1(\tfrac{n}{2}) + 1\}. \end{aligned}$$

Denn zur Beendigung von $A_1(n)$ muß $A_2(\frac{n}{2})$ beendet und ein Takt nach Berechnung von $A_1(\frac{n}{2})$ und der Beendigung von $p_{\frac{n}{2}-1}$ durch $A_2(\frac{n}{2})$ vergangen sein.
Durch Einsetzen von $D_i(\frac{n}{2})$, $D_i^*(\frac{n}{2})$ erhalten wir

$$\begin{array}{rrcl} & D_1(n) & = & \max\{D_1(\frac{n}{4})+2, D_1(\frac{n}{2})+1\}, \\ \text{also} & D_1(n) & = & \log n. \end{array}$$

Für die Schaltkreisgröße gilt:

$$\begin{array}{rclcl} C_1(1) & = & C_2(1) & = & 0, \\ C_1(2) & = & C_2(2) & = & 1, \end{array}$$

und für $n \geq 4$

$$\begin{array}{rrcl} & C_1(n) & = & C_2(\frac{n}{2}) + C_1(\frac{n}{2}) + \frac{n}{2}, \\ & C_2(n) & = & C_1(\frac{n}{2}) + n - 1, \\ \text{also} & C_1(n) & = & C_1(\frac{n}{2}) + C_1(\frac{n}{4}) + n - 1. \end{array}$$

Rekursionsgleichung ist zwar nicht geschlossen lösbar, aber es gilt:

$$C_1(n) \leq 4 \cdot n.$$

(Es gilt sogar $C_1(n) \leq 4 \cdot n - 6 \cdot n^{\frac{1}{2}}$ für $n \geq 4$). ■

- Zur Erinnerung:
  Eingabetypen an Position i bei Carry–Look–Ahead Addieren:
  E–Stelle: $x_i + y_i = 0 \quad (u_i, v_i) = (0,0)$,
  P–Stelle: $x_i + y_i = 1 \quad (u_i, v_i) = (0,1)$,
  G–Stelle: $x_i + y_i = 2 \quad (u_i, v_i) = (1,0)$.
- Wir fassen E, P, G als Funktionen aus $\mathbb{B}_1$ auf, die das Carry–Bit c_{i-1} in das Carry–Bit c_i überführen:

$$\begin{array}{rcl} E(c) & = & 0, \\ P(c) & = & c, \\ G(c) & = & 1. \end{array}$$

- Für $A_i \in \{E, P, G\}$ gilt:

$$c_i = A_i(c_{i-1}) = A_i \circ A_{i-1}(c_{i-2}) = \ldots = A_i \circ A_{i-1} \circ \ldots \circ A_0(0)$$

  da $c_1 = 0$.
  D.h., die Superposition $\circ$ ist assoziativ !

- Was ist mit $A_i \circ A_{i-1}$?
  $\{E, P, G\}$ bilden eine Halbgruppe bzgl. $\circ$:

$\circ$	E	P	G
E	E	E	E
P	E	P	G
G	G	G	G

  Wir erhalten also das Präfixproblem für die Addition:
  Für $B_i = A_i \circ \ldots \circ A_0$ gilt $c_i = B_i(0)$.

- Wir wollen das Präfixproblem lösen für Folgen über (E, P, G) bzw. für deren Codierungen $\{(0,0),(0,1),(1,0)\}$.

Lemma 2.4.
Für $(u, v) = (u_2, v_2) \circ (u_1, v_1)$ *gilt:* $u = u_2 + u_1 v_2, v = v_2 \cdot v_1$.

Beweis.
Sei $u = 1$.
Dann ist $(u, v) = G$.
1. Fall: $(u_2, v_2) = G$.
Dann ist $(u_1, v_1) \in \{E, P, G\}$ beliebig und
$u_2 = 1$.
2. Fall: $(u_2, v_2) = P$ und $(u_1, v_1) = G$.
Dann ist $u_1 = v_2 = 1$.
Sei $v = 1$:
Dann steht (u, v)für P, also ist $(u_2, v_2) = (u_1, v_1) = P$ und damit $v_1 = v_2 = 1$. ■
Aus dem Lemma folgt, daß $\circ$ mit einem Schaltkreis der Größe 3 und der Tiefe 2 realisiert werden kann.

- Sei $(u_i', v_i') = B_i = A_i \circ \ldots \circ A_0$
  Es gilt $c_i = u_i'$ wegen $G(0) = 1$ und $E(0) = P(0) = 0$.

Satz 2.18.
Der Ladner–Fischer–Addierer hat Tiefe $2 \log n + 2$ *und Größe* $\leq 15n$.

Beweis.
Wir wenden den Carry–Look–Ahead Algorithmus an:
Stufe 1 und Stufe 3 benötigen $3n - 1$ Bausteine und Tiefe 2.
Für Berechnung der Carry–Bits stehen Codierungen (u_i, v_i) für $A_i \in \{E, P, G\}$ zur

Verfügung:
Die $(u_i', v_i') = B_i = A_i \circ \ldots \circ A_0$ mit $u_i' = c_i$ können mit Präfixalgorithmus in Größe $3 \cdot C_1(n) \leq 3 \cdot 4n$ und Tiefe $2 \log n$ berechnet werden, da jede Option $\circ$ mit 3 Gattern und Tiefe 2 realisiert werden kann. ■

Beispiel 2.7.

x	$=$	1	1	0	1	0	1	1	0
y	$=$	1	1	1	0	1	1	0	0
Stufe 1		G	G	P	P	P	G	P	E.

Vergleich verschiedener Addierer:

	n	4	8	16	32	64
Größe:	Schulmethode	17	37	77	157	317
	Conditional–Sum	58	122	346	794	1786
	Ladner–Fischer	23	59	140	317	695
Tiefe:	Schulmethode	7	15	31	63	127
	Conditional–Sum	5	7	9	11	13
	Ladner–Fischer	6	8	10	12	14

Favorit für $n \leq 32$: Ladner–Fischer !

2.8.2 Multiplikation

$$MUL = (MUL_n) \text{ mit } MUL_n \in I\!B_{2n,2n}$$
$$MUL_n(\underbrace{x_{n-1}, \ldots, x_0}_{x}, \underbrace{y_{n-1}, \ldots, y_0}_{y}) = (\underbrace{p_{2n-1}, \ldots, p_0}_{p})$$

wobei:
x und y Binärdarstellungen von natürlichen Zahlen,

p Binärdarstellung des Produktes von x und y.

a) Schulmethode

Algorithmus:

1.) Berechne $z_i = (x_{n-1} \cdot y_i, \ldots, x_0 \cdot y_i)$ für alle $0 \leq i \leq n-1$.

2.) Addiere $|z_i| \cdot 2^i$ für alle $0 \leq i \leq n-1$.

Schaltkreis–Realisierung:

- Schritt 1 benötigt n^2 Gatter und Tiefe 1.
- In Schritt 2 sind n Zahlen zu addieren; Summe und Zwischensummen haben Länge ≤ 2.
- Wir führen Addition mit balanciertem binären Baum der Größe $n-1$ (bzgl. der Additionen) und Tiefe $\log n$ aus, z.B. mit Ladner–Fischer–Addierer.

Satz 2.19.
Die Schulmethode für die Multiplikation ist bei Verwendung von Ladner–Fischer–Addieren mit Multiplikationsschaltkreisen der Größe $O(n^2)$ und der Tiefe $O(\log^2 n)$ realisierbar. ■

b) Parallelisierung der Schulmethode

Wir können die Tiefe wesentlich verringern durch Parallelisierung.

Trick (Ofman, Wallace):
Man berechnet aus 3 Zahlen x, y, z zwei Zahlen u, v mit gleicher Summe.
Wir reduzieren damit die Zahl der benötigten Additionen.

Realisierung:
Für l–Bit Zahlen x, y, z werden die x_i, y_i, z_i mit l Volladdieren addiert; man erhält Carry-Bits u_i und Summenbit v_i.
Die Zahlen $u := (u_{l-1}, \ldots, u_0, 0)$ und $v := (v_{l-1}, \ldots, v_0)$ haben die gewünschte Eigenschaften. Ihre Berechnung ist möglich in Tiefe 3 ("Carry–Save–Addierer").

Anwendung auf Mulitplizierer–Schaltkreis:
Nach $(n-2)$–maliger Anwendung dieser Idee ergeben sich u, v mit $|p| = |u| + |v|$.

Parallelisierung der Addition:
Wir bauen Baum zur Reduktion der Zahl der Summanden auf "Wallace Baum". Die letzte Addition wird mit Ladner–Fischer–Addierern ausgeführt.

Satz 2.20.
Die Schulmethode für die Multiplikation ist bei Verwendung eines Wallace Baums mit einem Schaltkreis der Größe $O(n^2)$ und der Tiefe $O(\log n)$ realisierbar.

Beweis.
Schritt 1 benötigt $O(n^2)$ Gatter und Tiefe 1. Schritt 2 erfordert einen Wallace Baum der Tiefe $\log n$. Jede Stufe kann in Tiefe 3 und Größe $\leq O(n)$ realisiert werden. Die letzte Stufe mit dem Ladner–Fischer–Addierer erfordert Größe $O(n)$ und Tiefe $O(\log n)$. ∎

Korollar 2.6.
$MUL \in NC_1$. ∎

c) Karatsuba–Ofman Multiplizierer

Die Schulmethode für die Multiplikation ist nicht optimal bzgl. der Größe. Karatsubu und Ofman (1963) geben Multiplikations–Schaltkreis der Größe $O(n^{\log 3})$ an.
<u>Idee:</u> Teile–und –Herrsche– Ansatz: Sei $x = (x', x'')$, $y = (y', y'')$.
Es gilt:

$$\begin{aligned} |p| &= |x| \cdot |y| \\ &= (|x'|2^{\frac{n}{2}} + |x''|)(|y'|2^{\frac{n}{2}} + |y''|) \\ &= |x'||y'|2^n + \underbrace{(|x'||y''| + |x''||y'|)}_{(|x'|+|x''|)(|y'|+|y''|)-(|x'||y'|+|x''||y''|)} \cdot 2^{\frac{n}{2}} + |x''||y''| \end{aligned}$$

Algorithmus:

Seien
$A(n)$: die Kosten für die Addition zweier n–Bit–Zahlen,
$M(n)$: Kosten für die Multiplikation zweier n–Bit–Zahlen.

1.) $s' = ADD_{\frac{n}{2}}(x', x''), s'' = ADD_{\frac{n}{2}}(y', y'')$:
<u>Kosten:</u> $2A(\frac{n}{2})$.

2.) $p' = MUL_{\frac{n}{2}}(x', y'), p'' = MUL_{\frac{n}{2}}(x'', y'')$ $p^* = MUL_{\frac{n}{2}+1}(s', s'')$:
<u>Kosten:</u> $2M(\frac{n}{2}) + M(\frac{n}{2} + 1)$.

3.) $t = ADD_n(p', p'')$:
<u>Kosten:</u> $A(n)$.

4.) $u = SUB_{n+2}(p^*, t)$:
<u>Kosten:</u> $A(n + 2)$.
Sei u^* = Verlängerung von u um $\frac{n}{2}$ Nullen.
Dann gilt $|u^*| = (|x'||y''| + |x''||y'|)2^{\frac{n}{2}}$.

5.) $v = p'p''$ (Konkatenation).
Es gilt: $|v| = |x'||y'|2^n + |x''||y''|$. :
Kosten: 0.

6.) $p = ADD_{2n}(v, u^*)$:
Kosten: $A(\frac{3n}{2})$ (da u^* am Ende $\frac{n}{2}$ Nullen hat).

Satz 2.21.
Der Karatsuba–Ofman Multiplikations–Schaltkreis hat Größe $O(n^{\log 3}) \approx O(n^{1,58})$ *und Tiefe* $O(\log^2 n)$.

Beweis.
Man überlegt sich leicht, daß folgendes gilt:
Größe:

$$\begin{array}{llcl} & M(\frac{n}{2}+1) & \leq & M(\frac{n}{2}) + A(n) + A(\frac{n}{2}) + n + 1, \\ \text{also} & M(n) & \leq & 3M(\frac{n}{2}) + cn \text{ und } M(1) = 1 \text{ (da Addition in linearer Größe).} \end{array}$$

Die Lösung der Rekursionsgleichung liefert die Behauptung $M(n) = O(n^{\log 3})$.
Tiefe:
Es gilt:

$$\begin{array}{llcl} & D(n) & \leq & D(\frac{n}{2}) + d\log n \text{ und } D(1) = 1, \\ \text{also} & D(n) & = & O(\log^2 n). \quad \blacksquare \end{array}$$

Bemerkung.
Durch Parallelisierung können wir die Tiefe auf $O(\log n)$ drücken.

d) Schönhage–Strassen–Multiplizierer (1971)

Momentaner Rekord für die Schaltkreis–Größe und Tiefe für die Multiplikation ist Multiplikationsschaltkreis nach Schönhage–Strassen:

Satz 2.22.
Der Schönhage–Strassen–Algorithmus kann mit Multiplikations–Schaltkreis der Größe $O(n \log n \log\log n)$ *und der Tiefe* $O(\log n)$ *realisiert werden.*

Beweis.
Idee: Multiplikation auf der Basis der schnellen Fouriertransformation. $\blacksquare$

Praktisch ist dieser Schaltkreis für kleine n ohne Bedeutung, da Konstanten in der O–Darstellung sehr groß.

2.8.3 Symmetrische Funktionen

Wichtige Beispiele

$$\begin{array}{lcl} E^n_k(x) & = & 1 \Longrightarrow x_1 + \ldots + x_n = k;\ \text{“exactly–}k\text{–Funktion”}; \\ T^n_k(x) & = & 1 \Longrightarrow x_1 + \ldots + x_n \geq k;\ \text{“Threshold–}k\text{–Funktion”}; \\ S^n(x) & = & (T^n_1(x), \ldots, T^n_n(x));\ \text{“Sortierfunktion”}; \\ C^n_{k,m}(x) & = & 1 \Longrightarrow x_1 + \ldots + x_n \equiv k \bmod m;\ \text{“counting–function”}; \\ & & \text{z.B.: } C^n_{1,2} = x_1 \oplus \ldots \oplus x_n = PAR_n. \end{array}$$

Satz 2.23.
$\{E^n_0, \ldots, E^n_n\}$ bildet eine Basis für $\mathcal{S}_n$.

Beweis.
Sei $f \in \mathcal{S}_n$.
Dann gilt $f(x) = \sum_{0 \leq k \leq n} v_k \cdot E^n_k(x)$ für geeignete Konstante $v_k \in \{0, 1\}$. ■

Beispiel 2.8.

$T_k = \sum_{i \geq k} E_i$. ■

Korollar 2.7.
Sei $f \in \mathcal{S}_n$. Dann gilt:

$$C(f) \leq C(E_0, \ldots, E_n) + n$$

und

$$D(f) \leq D(E_0, \ldots, E_n) + \log n. \quad ■$$

Wir wollen E_k effizient berechnen:

Satz 2.24.
Für alle $k \in \mathbb{N}$ gilt:

$$C(E_k) = O(n) \text{ und } D(E_n) = O(\log n).$$

Beweis.

Es gilt

$$E_k(x) = 1 \Leftrightarrow \sum_{i=1}^{n} x_i = k.$$

1. Berechne $BSUM_n(x_1, \ldots, x_n) = (a_{\log n}, \ldots, a_0)$, wobei $(a_{\log n}, \ldots, a_0)$ die Binärdarstellung von $x_i + \ldots + x_n$ bezeichnen soll.

Lemma 2.5.
$BSUM_n$ kann von Schaltkreisen der Größe $O(n)$ und der Tiefe $O(\log n)$ berechnet werden.

Beweis.
Berechne Summe aus n 1–Bit Zahlen mit Wallace–Tree.

Analyse:
Auf Stufe i des Wallace Trees sind i–Bit Zahlen zu addieren.
Damit haben die CSA die Größe $O(i)$.
Die Anzahl der Summanden auf Stufe j ist $\leq (\frac{2}{3})^{j-1} \cdot \frac{n}{3}$. Sie haben Länge j.
Aus $\sum_{j=1}^{n} (\frac{2}{3})^{j-1} \cdot j \leq 9$ folgt die Behauptung. ■

2. E_k sind Minterme über $(a_{logn}, \ldots, a_0)$.
   Es wurde bereits gezeigt, daß sich alle Minterme über s Literalen von Schaltkreisen der Größe $O(2^s)$ und der Tiefe $O(\log s)$ berechnen lassen.
   Folglich kann E_k von Schaltkreisen der Größe $O(2^{\log n}) = O(n)$ und der Tiefe $O(\log \log n)$ aus der Eingabe $(a_{\log n}, \ldots, a_0) = BSUM(x)$ berechnet werden. ■

Wir erhalten:

Satz 2.25.
Sei $f \in \mathcal{S}_n$. Dann gilt:

$$C(f) = O(n) \text{ und } D(f) = O(\log n). \quad ■$$

Korollar 2.8.
$\mathcal{S}_n \subseteq NC_1$.

2.8.4 Matrixmultiplikation

Seien

$$X = (x_{ij}), Y = (y_{ij}), 1 \leq i, j \leq n$$

zwei $n \times n$ Matrizen mit, $x_{ij}, y_{ij} \in R$, R kommutativer Hablbring. Die Matrizenmultiplikation besteht in der Berechnung von

$$MMUL_n(X, Y) = Z = (z_{ij}), 1 \leq i, j \leq n, \text{ mit } z_{ij} = \sum_{1 \leq k \leq n} x_{ik} \cdot y_{kj}.$$

Vorgehen:
Wir entwerfen ein $\{+,-,*\}$–Schaltkreis (*arithmetischer Schaltkreis*).
Wir erhalten einen Booleschen Schaltkreis durch Ersetzung der arithmetischen Operationen durch Boolesche Schaltkreis, die diese Operationen auf den Eingaben entsprechender Länge realisieren.

a) Schulmethode

Lemma 2.6.
Wir können $MMUL_n$ mit $2n^3 - n$ arithmetischen Operationen realisieren.

Beweis.
Die n^3 Multiplikationen $x_{kj} \cdot y_{kj}$ können parallel ausgeführt werden;
die z_{ij} können als Summe von n Summanden mit $n-1$ Additionen in Tiefe $\log n$ berechnet werden. ■

Satz 2.26.
Sei x_{ij} n–Bit Zahlen. Bei Verwendung eines Booleschen Multiplikations–Schaltkreises der Größe $M(n)$ und der Tiefe $O(\log n)$ kann $MMUL_n$ vom Schaltkreis der Größe $O(n^3 M(n))$ und der Tiefe $O(\log n)$ realisiert werden. ■

Korollar 2.9.
$MMUL_n \in NC_1$.

b) Strassen'sche Matrizenmultiplikation

Wir können die Zahl der arithmetischen Operationen (also die Größe des Schaltkreises) verringern: <u>Idee:</u> Setzen

$$X = \begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix}$$
$$Y = \begin{pmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{pmatrix}$$
$$Z = \begin{pmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{pmatrix}$$

X_{rs}, Y_{rs}, Z_{rs} sind $\frac{n}{2} \times \frac{n}{2}$ Matrizen ($n = 2^k$).
Es gilt:

$$\begin{aligned} Z_{11} &= X_{11}Y_{11} + X_{12}Y_{21} \\ Z_{12} &= X_{11}Y_{12} + X_{12}Y_{22} \\ Z_{21} &= X_{21}Y_{11} + X_{22}Y_{21} \\ Z_{22} &= X_{21}Y_{12} + X_{22}Y_{22}. \end{aligned}$$

Damit läßt sich Z mit 8 Multiplikationen berechnen. Wir können Z mit *nur* 7 Multiplikationen berechnen:
Sei

$$\begin{aligned}
M_1 &= (X_{12} - X_{22})(Y_{21} + Y_{22}) \\
M_2 &= (X_{11} + X_{22})(Y_{11} + Y_{12}) \\
M_3 &= (X_{11} - X_{21})(Y_{11} + Y_{12}) \\
M_4 &= (X_{11} - X_{12}) \cdot Y_{22}) \\
M_5 &= X_{11} \cdot (Y_{12} - Y_{22}) \\
M_6 &= X_{22} \cdot (Y_{21} - Y_{11}) \\
M_7 &= (X_{21} - X_{22}) \cdot Y_{11}).
\end{aligned}$$

Es gilt:

$$Z = \begin{pmatrix} M_1 + M_2 - M_4 + M_6 & M_4 + M_5 \\ \\ M_6 + M_7 & M_2 - M_3 + M_5 - M_7 \end{pmatrix}$$

Analyse.
Seien $C_a(n), D_a(n)$ die Größe bzw. die Tiefe des arithmetischen Schaltkreises zur Realisierung der Strassen'schen Matrixmultiplikation.
Es gilt:

$$\begin{aligned}
C_a(n) &= 7C_a(\tfrac{n}{2}) + 18(\tfrac{n}{2})^2 \\
C_a(1) &= 1 \\
D_a(n) &= D_a(\tfrac{n}{2}) + 3,
\end{aligned}$$

$\Rightarrow$

$$\begin{aligned}
C_a(n) &= 7n^{\log 7} - 6n^2 \quad (\log 7 \approx 2,81) \\
D_a(n) &= 3\log n + 1.
\end{aligned}$$

Satz 2.27.
Zwei $n \times n$ Matrizen lassen sich für $n = 2^k$ mit einem arithmetischen Schaltkreis der Größe $7n^{\log 7} - 6n^2$ und Tiefe $3\log n + 1$ multiplizieren.

c) Boolesche Matrizenmultiplikation

Sei $R = \{0, 1\}$.

Implementierung der Strassen'schen Matrizenmultiplikation: Fischer/Meyer (1971).

$$\begin{aligned} x_{ij}, y_{ij} &\in \{0,1\} \\ z_{ij} &= \bigvee_{k=1}^{n} x_{ik}y_{ij}, 1 \leq i,j \leq n \\ z_{ij}^* &= \sum_{k=1}^{n} x_{ik}y_{kj} \in \mathbb{Z} \end{aligned}$$

Es gilt:

$$\begin{aligned} &(1) \quad 0 \leq z_{ij}^* \leq n \\ &(2) \quad 0 = z_{ij}^* \Longrightarrow z_{ij} = 0. \end{aligned}$$

- Berechnen von z_{ij}^*
- Wir können über $\mathbb{Z}_m = \{0,1,\ldots,m-1; + \bmod m, \cdot \bmod m\}$ arbeiten für $m > n$.
  $\Rightarrow z_{ij}^*$ haben Länge $O(\log n)$.
- Wir können z_{ij} als Disjunktion aller Bits von z_{ij}^* berechnen in Tiefe $O(\log \log n)$.

Satz 2.28.
Das Boolsche Matrixprodukt kann durch einen Schaltkreis der Größe $O(n^{\log 7} \log n \log \log n \log \log \log n)$ und der Tiefe $O(\log n)$ berechnet werden. ■

2.9 Typische Schaltkreisgröße

Im folgenden fragen wir nach der typischen Schaltkreis–Größe eines Problems.

Definition 2.13.
Sei $\mathcal{F}_n \subseteq \mathbb{B}_n$ Menge von Boolesche–Funktionen.
Wir sagen "fast alle Boolesche–Funktionen haben Eigenschaft P", genau dann, wenn

$$\frac{\sharp\{f \in \mathcal{F}_n | f hat P\}}{\sharp \mathcal{F}_n} \longrightarrow 1 (n \to \infty).$$

Lemma 2.7.
Höchstens

$$c(b,n) = \frac{(b+n+1)^{2b} \cdot 16^b \cdot b}{b!}.$$

Beweis.
Wir schätzen die Zahl der $\mathbb{B}_2$–Schaltkreise der Größe b ab:
Für jedes Gatter können

- 16 verschiedenen Typen gewählt werden und
- $(b+n+1)$ Verbindungsmöglichkeiten zu Vorgängern ($b-1$ anderer Gatter, n Inputs, 2 Konstanten)

Jeder $\mathbb{B}_2$–Schaltkreis der Größe b berechnet höchstens b verschiedene Funktionen. Wir müssen beachten, daß $b!$ verschiedene Numerierungen der Gatter für einen Schaltkreis möglich sind und zur $b!$ fachen Zählung geführt haben. ■

Betrachte: $c_n := C(\mathbb{B}_n) := \max\{C(f) : f \in \mathbb{B}_n\}$

Wir wollen c_n abschätzen:
Es gilt: $C(c_n, n) = \sharp\mathbb{B}_n = 2^{2^n}$.
Die Anwendung der *Stirlingschen Formel* $a! \geq r \cdot a^{(a+\frac{1}{2})} \cdot e^{-a}$ ergibt

$$\log C(c_n, n) \geq \log \sharp\mathbb{B}_n,$$

und damit

$$c_n \log(c_n + n + 1) + 4c_n + \log c_n - \frac{(c_n + 1)}{2} \log c_n + c_n \log e - \log r \geq 2^n,$$

also $c_n \log c_n + (6 + \log e)c_n + \frac{1}{2} \log c_n - \log r \geq 2^n$ (für $c_n \geq n + 1$, n hinreichend groß).
<u>Annahme:</u> $c_n < \frac{2^n}{n}$.
Daraus folgt $2^n n^{-1}(n - \log n + 6 + \log e) + \frac{1}{2}(n - \log n) - \log r \geq 2^n$, Widerspruch!
Folglich gilt $C(\mathbb{B}_n) \geq \frac{2^n}{n}$.

Bemerkung.
Analog gilt:

$$C(\mathbb{B}_n^*) \geq \frac{2^n}{n} \text{ für } \log \sharp\mathbb{B}_n^* \geq 2^n - \frac{2^n}{n} \log\log n.$$

Theorem 2.4. *(folgt aus der Bemerkung)*
Für genügend große n haben wenigstens $\sharp\mathbb{B}_n(1 - 2^{-r(n)})$ der Funktionen aus $\mathbb{B}_n$ eine Schaltkreis–Größe von mindestens

$$\frac{2^n}{n}\cdot, \text{ wobei } r(n) := \frac{2^n}{n} \log\log n.$$

D.h. fast alle BooleschenFunktionen $f \in \mathbb{B}_n$ haben Schaltkreis–Größe $\geq \frac{2^n}{n}$.

Tatsächlich sind $\frac{2^n}{n} + o(\frac{2^n}{n})$ Gatter ausreichend.

Satz 2.29.
Für alle Booleschen–Funktionen $f \in \mathbb{B}_n$ gilt

$$C(f) = O(\frac{2^n}{n}).$$

(Genauer gilt: $C(f) = \frac{2^n}{n} + o(\frac{2^n}{n})$).

Beweis.
Lupanov ('58) und Shannon ('49) haben recht einfache Schemata zur Konstruktion von Schaltkreisen der Größe $O(\frac{2^n}{n})$ für bel. Boolesche–Funktionen $f \in \mathbb{B}_n$ angegeben.
Wir wollen ein recht effizientes Verfahren angeben, das Schaltkreise mit (nur) 4 logischen Stufen für beliebige Boolesche Funktionen $f \in \mathbb{B}_n$ der Größe $\frac{2^n}{n} + o(\frac{2^n}{n})$ bereitstellt:
Die Konstruktion basiert auf Lupanovs (k,s)–Repräsentation.
Sei $f \in \mathbb{B}_n$.
Betrachte für f als $2^n \times 2^{n-k}$ Matrix:

- Die Zeilen entsprechen 2^k Werten für $(x_1, \ldots, x_k)$.
- Die Spalten entsprechen 2^{n-k} Werten für $(x_{k+1}, \ldots, x_n)$.
- Im Schnittpunkt einer Zeile und einer Spalte steht der Funktionswert $f(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)$.

Wir fassen die Zeilen in $p = \lceil \frac{2^k}{s} \rceil$ Blöcke $A_1, \ldots, A_p$ zusammen ($A_1, \ldots, A_{p-1}$ enthalten genau s Zeilen, A_p enthält $s' \leq s$ Zeilen).
<u>Betrachte</u>
f_i mit

$$f_i(x) = \begin{cases} f(x) & \text{falls } (x_1, \ldots, x_k)\text{“} \in \text{”} A_i, \\ 0 & \text{sonst.} \end{cases}$$

Offenbar gilt $f(x) = f_1(x) \vee \ldots \vee f_p(x)$.
Sei $B_{i,w}$ die Menge der Spalten, die im Block A_i mit $w \in \{0,1\}^s$ (für A_p mit $w \in \{0,1\}^{s'}$) übereinstimmen.

Beispiel 2.9.
$k = 3, s = 2, n = 6$

f								
	0	0	0	0	1	1	1	1
	0	0	1	1	0	0	1	1
	0	1	0	1	0	1	0	1
000	0	1	1	0	1	1	1	0
001	1	0	0	1	0	1	0	1
010	1	0	1	1	0	0	1	0
011	0	0	0	1	1	0	0	0
100	1	1	1	0	0	1	1	0
101	0	1	1	0	1	0	1	0
110	1	0	0	1	1	0	0	1
111	1	1	0	0	0	0	1	1

$B_{2,\binom{0}{0}} = \{2, 6, 8\}$,
$B_{3,\binom{1}{0}} = \{1, 6\}$.

<u>Betrachte</u>
$f_{i,w}$ mit

$$f_{i,w}(x) = \begin{cases} f_{i,w}(x) & \text{falls } (x_{k+1}, \ldots, x_n) \in \mathbb{B}_{i,w}, \\ 0 & \text{sonst.} \end{cases}$$

Offenbar gilt $f(x) = \vee_{i,w} f_{i,w}(x)$.
$f_{i,w}$ ist sehr einfach gebaut:

$$f_{i,w} = 1 \Leftrightarrow \begin{cases} 1.) & (x_{k+1}, \ldots, x_n) \in B_{i,w}, \\ 2.) & \text{für ein} j \text{ mit } w_j = 1 \text{ ist } (x_1, \ldots, x_k)\ j\text{–te Zeile von } A_i. \end{cases}$$

$$\left.\begin{array}{ll} f_{i,w}^{(1)} & \text{sei die Boolesche–Funktion, die 1.) überprüft} \\ f_{i,w}^{(2)} & \text{sei die Boolesche–Funktion, die 2.) überprüft} \end{array}\right\} f_{i,w}(x) = f_{i,w}^{(1)} \wedge f_{i,w}^{(2)}$$

Offenbar hängt $f_{i,w}^{(1)}$ lediglich von $(x_{k+1}, \ldots, x_n)$ ab und $f_{i,w}^{(2)}$ lediglich von $(x_1, \ldots, x_k)$.
Wir erhalten nun die *Lupanovsche* (k, s)*–Darstellung*:

$$f(x_1, \ldots, x_n) = \bigvee_{1 \leq i \leq p} \bigvee_{w} f_{i,w}^{(1)}(x_1, \ldots, x_k) \wedge f_{i,w}^{(2)}(x_{k+1}, \ldots, x_n).$$

Konstruktion eines Booleschen–Schaltkreises der Tiefe 4 für f:

1.) (Konjunktion:) Wir bilden alle 2^k Minterme auf $(x_1, \ldots, x_k)$ und alle 2^{n-k} Minterme auf $(x_{k+1}, \ldots, x_n)$ ab (dazu werden $(k-1) \cdot 2^k + (n-k-1) \cdot 2^{n-k}$ ANDs benötigt).

2.) (Disjunktion:) Berechnung der $kDNF$ für alle $f_{i,w}^{(1)}$.

(Für festes i kommt jeder $(x_{k+1}, \ldots, x_n)$ Minterm nur in einem $f_{i,w}^{(1)}$ vor; da es p A_i–Blöcke und 2^{n-k} Minterme gibt, genügen $p \cdot 2^{n-k}$ ORs) Berechnung der $kDNF$ für alle $f_{i,w}^{(2)}$.

(Für festes i kommt jeder $(x_1, \ldots, x_k)$ Minterm nur in einem $f_{i,w}^{(2)}$ vor. Da es 2^s w–Vektoren mit 2^k Mintermen gibt, genügen $2^2 \cdot 2^k$ ORs.)

3.) (Konjunktion:) Berechnung von

$$f_{i,w}(x) = f_{i,w}^{(1)} \wedge f_{i,w}^{(2)},$$

(dazu werden $p, 2^s$ ANDs benötigt).

4.) (Disjunktion:) Berechnung von

$$f = \bigvee f_{i,w},$$

(dazu werden $p, 2^s - 1$ ORs benötigt).

<u>ÜA:</u> Analyse der Größe des konstruierten Schaltkreises für die Wahl von $k := \lceil 3 \cdot \log n \rceil$ und $s := n - \lceil 5 \cdot \log n \rceil$. Dies liefert die Behauptung des Satzes.

Satz 2.30.

1. *Für fast alle monotone Boolesche–Funktionen $f \in \mathbb{B}_n$ gilt* $C_n(f) = \Omega(\frac{2^n}{n\sqrt{n}})$.
2. *Für alle monotonen Boolesche–Funktionen $f \in \mathbb{B}_n$ gilt* $C_n(f) = O(\frac{2^n}{n\sqrt{n}})$.

2.10 Untere Schranken für die Schaltkreisgröße

Wir wissen, daß fast alle Boolesche–Funktionen zu ihrer Berechnung exponetiell große Schaltkreise erfordern.
Wie sieht dies für konkret gegebene Funktionen aus?

1. Ansatz:

- Numerieren aller Booleschen–Funktionen $f \in \mathbb{B}_n$ gemäß lexikographischer Ordnung ihrer Tabellen.

- $f_n^* \in \mathbb{B}_n$ sei lexikographisch erste Funktion mit

$$C(f_n^*) \geq \frac{2^n}{n}.$$

  f_n^* existiert, aber ist nicht <u>konkret</u> gegeben.
  Wir wissen nichts von f_n^*, außer, daß die Schaltkreis–Realisierung notwendig groß ist.

- Wir wollen untere Schranke für Schaltkreis–Größe von *konkret* gegebenen Funktionen oder *explizit definierten* Funktionen ableiten.

2. Ansatz:

Definition 2.14.
$\{f_n\}$ ist *explizit gegeben*, genau dann, wenn $\cup f_n^{-1}(1) \in NP$. Ist also in polynomialer Zeit von einer NTM berechenbar.

a) Nichtdegenerierte Boolesche–Funktionen

Definition 2.15.
$f \in \mathbb{B}_n$ ist *nicht degeneriert*, genau dann, wenn f wesentlich von allen Variablen abhängt. D.h.

$$f|_{x_i=0} \neq f|_{x_i=1} \quad \text{für alle } i = 1, \ldots, n.$$

Satz 2.31.
Ist $f \in \mathbb{B}_n$ nicht degeneriert, dann gilt

$$C(f) \geq n - 1.$$

Beweis.
Sei S Schaltkreis für f der Größe c, sei e die Zahl der Kanten von S.
Es gilt:

(i) $e = 2 \cdot c$, da jedes Gatter fan–in 2 hat.

(ii) $e \geq n + c - 1$, da

- kein Input–Knoten fan–out 0 hat (sonst degeneriert),
- kein Gatter, außer Ausgabegatter, fan–out 0 hat (sonst eliminierbar).

Folglich ist $2c \geq n + c - 1$,
also $c \geq n - 1$. ■

Bemerkung.
Schranke ist optimal für manche Funktionen;
z.Bsp.:

$$f_n(x) = x_1 \oplus \ldots \oplus x_n.$$

b) Eliminationsmethode

Definition 2.16.
Eine Boolesche–Funktion $f \in \mathbb{B}_n$ gehört zur Klasse $Q^n_{2,3}$, genau dann, wenn für alle verschiedenen $i, j \in \{1, \ldots, n\}$ gilt, daß wenigstens drei der Unterfunktionen

$$f|_{x_i=0,x_j=0}, f|_{x_i=1,x_j=0}, f|_{x_i=0,x_j=1}, f|_{x_i=1,x_j=1}$$

veschieden sind und wenn $f|_{x_i=0}$ oder $f|_{x_i=1}$ zu $Q^{n-1}_{2,3}$ gehört (für $n \geq 4$).

Beispiel 2.10.
Es gilt $C^n_{k,3} \in Q^n_{2,3}$. ($C^n_{k,3}(x_1, \ldots, x_n) = 1 :\Leftrightarrow x_1 + \ldots x_n \equiv k \bmod 3$). Weiter gilt

$$\begin{aligned}
C^n_{k,3}|_{x_i=0,x_j=0} &= C^{n-2}_{k,3} \\
C^n_{k,3}|_{x_i=0,x_j=1} &= C^n_{k,3}|_{x_i=1,x_j=0} = C^{n-2}_{k-1,3} \\
C^n_{k,3}|_{x_i=x_j=1} &= C^{n-2}_{k-2,3}
\end{aligned}$$

und für $n \geq 4$

$$C^n_{k,3}|_{x_i=0} = C^{n-1}_{k,3} \in Q^{n-1}_{2,3}. \quad \blacksquare$$

Satz 2.32. *(Schnorr)*
Gilt $f \in Q^n_{2,3}$, dann ist $C(f) \geq 2n - 3$.

Beweis.
Offenbar ist f nicht degeneriert, falls $f \in Q^n_{2,3}$. Ansonsten könnte f für die unwesentliche Variable x_i nur 2 Unterfunktionen bzgl. x_i,x_j besitzen.
Sei S ein optimaler Schaltkreis für f.
Sei g ein erstes Gatter von S.
Dann ist g verbunden mit zwei Eingängen x_i,x_j.
<u>Annahme:</u> x_i,x_j–Eingabeknoten haben fan–out 1.
Dann hängt f nur über g von x_i und x_j ab.
g berechnet entweder 0 oder 1.
Es folgt $\sharp\{f|_{x_i=0,x_j=0}, f|_{x_i=0,x_j=1}, f|_{x_i=1,x_j=0}, f|_{x_i=x_j=1}\} \leq 2$, Widerspruch.
Man erhält z.B. x_i hat fan–out ≥ 2.
<u>Sei $n = 3$:</u>
Wenigstens 4 Kanten verlassen die Eingabeknoten und wir können wie im letzten Satz verfahren und zeigen, daß wenigstens 3 Gatter vorhanden sein müssen.
<u>Sei $n \geq 4$:</u>
Wir ersetzen x_i durch 0 bzw. 1.
Also können wir wenigstens 2 Nachfolger von x_i eliminieren.
Wir erhalten eine Funktion f' aus $Q^{n-1}_{2,3}$.
Nach Induktionsvoraussetzung hat f' wenigstens $2n - 5$ Gatter.
Folglich hat f $2n - 3$ Gatter. $\blacksquare$

Korollar 2.10.
Es gilt, falls $n \geq 3$, $C(C^n_{k,3}) \geq 2n-3$.

Beweis.
Der Beweis folgt aus $C^n_{k,3} \in Q^n_{2,3}$ und dem Satz von Schnorr. ■

c) untere Schranke für Schaltkreise über anderen Basen

Betrachte: $\mathcal{U}_2$–Schaltkreis, $\mathcal{U}_2 = \mathbb{B}_2 - \{\oplus, \equiv\}$.

Satz 2.33.
Sei $f_c(x_1, \ldots, x_n) = x_1 \oplus \ldots \oplus x_n \oplus c$ *für* $n \geq 2, c \in \{0,1\}$.
Es gilt:

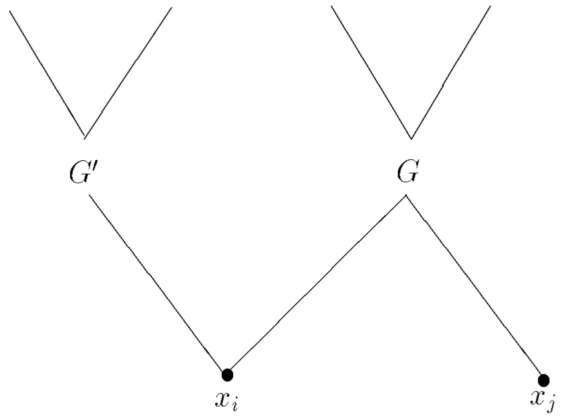
$$\begin{array}{llcl} (a) & C_{\mathbb{B}_2}(f_c) & = & n-1, \\ (b) & C_{\mathcal{U}_2}(f_c) & = & 3(n-1). \end{array}$$

Beweis.
(b) Es ist zu zeigen: $C_{\mathcal{U}_2}(f_c) = 3(n-1)$.

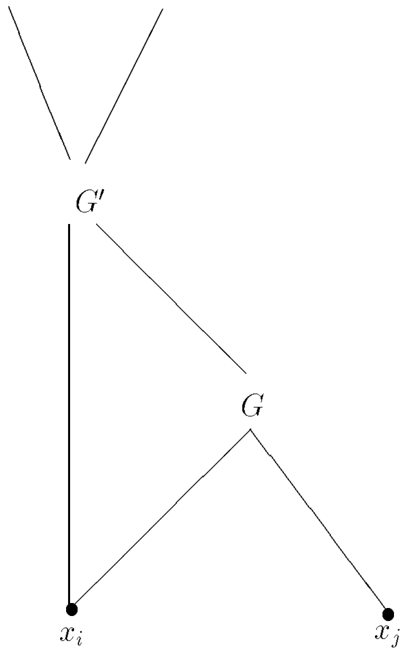
b.1: $C_{\mathcal{U}_2}(f_c) \leq 3(n-1)$.
Dies folgt aus $x \oplus y = x\overline{y} + \overline{x}y$

b.2: $C_{\mathcal{U}_2}(f_c) \geq 3(n-1)$.
Es genügt zu zeigen: es existiert ein x_i, dessen Konstantsetzung die Elimination von 3 Gattern erlaubt.
(Behauptung folgt im Falle $n = 2$ direkt, im Falle $n \geq 3$ durch Konstruktion.)
Sei G ein erstes Gatter im optimalen $\mathcal{U}_2$–Schaltkreis für f_c.
Dann sind die Inputs für G zwei verschiedene Variablen x_i,x_j.
Der fan–out von x_i ist ≥ 2, ansonsten würde (wegen der Basis $\mathcal{U}_2$) die geeignete Konstantsetzung von x_j zur Konstantsetzung von G führen und somit der Schaltkreis von x_i unabhängig werden.
Wir setzen x_i konstant, so daß G konstant gesetzt wird. G', der zweite Nachfolger von x_i, hängt nur noch von einer Variablen ab und kann damit ebenfalls eliminiert werden.

1. Fall G hat einen von G' verschiedenen Nachfolger.
Da G konstant ist, ist der Nachfolger von G nur von einer Variablen abhängig und kann eliminiert werden. Insgesamt können mindestens drei Gatter eliminiert werden.



2. Fall G'' ist der einziger Nachfolger von G.
G' ist konstant. Die Nachfolger von G' sind damit nur von einer Variablen abhängig und können ebenfalls eliminiert werden.



(a) Es ist zu zeigen: $C_{\mathbb{B}_2}(f_c) = n - 1$.

a.1: $C_{\mathbb{B}_2}(f_c) \leq n - 1$ gilt nach Definition.

a.2: $C_{\mathbb{B}_2}(f_c) \geq n - 1$ folgt mit dem Satz über nichtdegenerierte Funktionen. ∎

Definition 2.17.(Sequence Equality Test)
$SEQ_n(x_1, \ldots, x_n, y_1, \ldots, y_n) = 1$, genau dann, wenn $x_i = y_i$ für alle $1 \leq i \leq n$.
Dies ist eine wichtige Funktion für kommunikationskomplexitätstheorethische Untersuchungen.

Satz 2.34.

$$\begin{array}{lrcl} (a) & C(SEQ_n) & = & 2n-1 \\ (b) & C_{\mathcal{U}_2}(SEQ_n) & = & 4n-1. \end{array}$$

Beweis.

a.1: $C(SEQ_n) \leq 2n-1$,
$SEQ_n(x, y) = (x_1 \equiv y_1) \wedge \ldots \wedge (x_n \equiv y_n)$.

b.1: $C_{\mathcal{U}_2}(SEQ_n) \leq 4n-1$,
$x_i \equiv y_i = x_i y_i \vee \overline{x_i}\,\overline{y_i}$.

a.2: $C(SEQ_n) \geq 2n-1$,
dies folgt aus dem Satz über nichtdegenerierte Funktionen.

b.2: $C_{\mathcal{U}_2}(SEQ_n) \geq 4n-1$,
Analog wie im letzten Satz. (ÜA) ∎

d) Stand der Forschung

Die beiden folgenden Ergebnisse sind bekannt:

Satz 2.35. *(Stockmeyer 1977)*
Sei $f \in \mathcal{S}_n$ ein symmetrische Funktion.
Hat der Wertevektor $v(f)$ die Gestalt $v(f) = \omega'\omega\omega''$, wobei $\omega \in \{0,1\}^4$ ein Wort mit 3 verschiedenen Unterworten der Länge 2 ist und es sei $|\omega'|, |\omega''| \geq k$, dann gilt

$$C(f) \geq 2n + k - 3. \quad \blacksquare$$

Daraus folgt:

- $C(T_k^n) \geq 2n + \min\{k-2, n-k-1\} - 3$,
  und somit $C(T^n_{\lceil \frac{n}{2} \rceil}) \geq 2,5n - 5$
- $C(C^n_{k,m} \geq 2,5n - 0,5m - 4$.

Satz 2.36. *(Blum 1984)*

$$C(f) \geq 3n - 3$$

für die folgende Funktion $f \in \mathbb{B}_{n+3k+3}$

$$f(a, b, c, p, q, r, x) = q[x_{|a|} x_{|b|} + (p, x_{|b|} x^{r}_{|c|})] + \overline{q}(x_{|a|} \oplus x_{|b|})$$

mit: $n = 2^k$
$a = (a_{k-1}, \ldots, a_0)$
$b = (b_{k-1}, \ldots, b_0)$
$c = (c_{k-1}, \ldots, c_0)$
$x = (x_0, \ldots, x_{n-1})$
$p, q, r \in \mathbb{B}$. ■

Kapitel 3

Boolesche Formeln

3.1 Modell

Definition 3.1.
Boolesche Schaltkreise mit fan–out 1 heißen *Boolesche Formeln.*
$L_\Omega(f) := C_{1,\Omega}(f)$ *Formelgröße* von Booleschen Ω–Formeln.

Wir wissen bereits:

- Formeln sind baumartige Schaltkreise.
- Wir können zu jedem Ω–Schaltkreis S berechnungsäquivalente Ω–Formeln F konstruieren mit
$$depth(S) = depth(F).$$
- Die Arbeit zur Untersuchung unterer Schranken für Schaltkreis–Tiefen erfolgt (stets) mit Formeln.
(Achtung: i.a. ist die Größe der Formel größer als die Schaltkreis–Größe)

Satz 3.1.
Sei Ω eine Basis mit fan–in k. Dann gilt

$$C_\Omega(f) \overset{(1)}{\leq} L_\Omega(f) \overset{(2)}{\leq} k^{D_\Omega(f)} \quad \text{für alle } f \in \mathbb{B}_n.$$

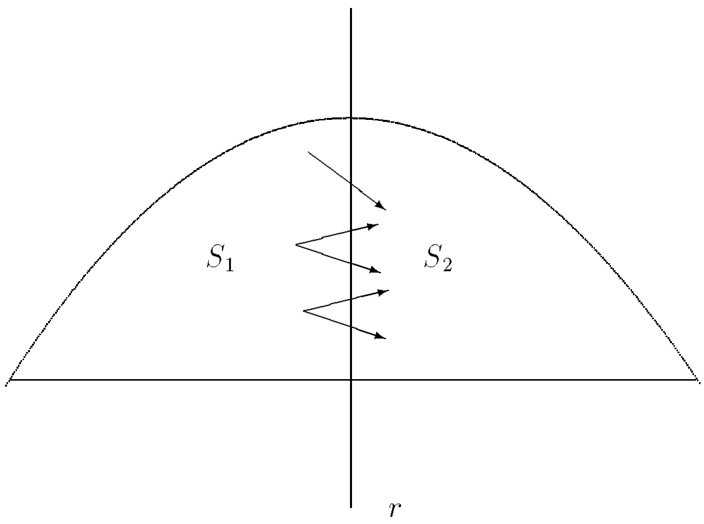
Beweis.
(1) trivial.
(2) Sei $L_\Omega(f)$ realisiert durch die Formel F.
Dann ist $size(F) \leq k^{D_\Omega(f)}$. ■

Satz 3.2. *(Paterson, Valiant 1976)*
Es gilt

$$D(f) = O(\frac{C(f)}{\log C(f)}).$$

(wünschenswert ist: $D(f) = \omega(\log C(f))$).

Beweis.
<u>Idee:</u>
Sei S ein Schaltkreis mit $size(S) = C(f)$.



Wir partitionieren S in S_1, S_2, so daß kein Weg von S_2 nach S_1 führt.

1. Fall: r schneidet viele Kanten.
Dann hat S große Weite.
Also genügt es die Tiefe beider Teile S_1, S_2 zu beschränken.

2. Fall: r schneidet wenige Kanten.
Also hängt f von S_1 und von den entlang dieser Kante berechneten Funktionen $g_1, \ldots, g_m$ ab.
Betrachte $f_c(x)$ für $c = (g_1(x), \ldots, g_m(x)) \in \{0,1\}^m$.
Dann wird f von S_2 berechnet.
Es folgt

$$f(x) = \bigvee_{c \in \{0,1\}^m} g_1(x)^{c(1)} \wedge \ldots \wedge g_m(x)^{c(m)} \wedge f_c(x). \qquad (*)$$

Wir berechnen $g_1, \ldots, g_m$ und alle f_c parallel in geringer Tiefe und berechnen f nach $(*)$.

3.2 Satz von Spira

Definition 3.2.
Sei $sel \in \mathbb{B}_3$, wobei

$$sel(x, y, z) = \begin{cases} y \text{ falls } x = 0, \\ z \text{ falls } x = 1 \end{cases}$$

Dann ist $k(\Omega)$ definiert durch $k(\Omega) := \frac{D_\Omega(sel)+1}{\log 3 - 1}$.

Beispiel 3.1.
$k(\mathbb{B}_2) = \frac{3}{\log 3 - 1} \approx 5,1$.

Theorem 3.1. *(Spira 1971)*
Sei $f \in \mathbb{B}_n$, $\Omega \subseteq \mathbb{B}_2$ vollständig. Dann gilt:

$$\log(L_\Omega(f) + 1) \overset{(1)}{\leq} D_\Omega(f) \overset{(2)}{\leq} k(\Omega) \log(L_\Omega(f) + 1).$$

D.h.: $D_\Omega(f) = \Theta(\log L_\Omega(f))$.

Beweis.

zu (1): Sei S ein Ω–Schaltkreis für f mit $depth(S) = D_\Omega(f) = d$.
Wir können eine Formel F für f konstruieren mit $depth(F) = d$. F ist binärer Baum der Tiefe d.
Somit ist die Zahl der inneren Knoten von $F \leq 2^d - 1$, also gilt $L_\Omega(f) \leq 2^d - 1$.

zu (2): Durch vollständige Induktion über $l = L_\Omega(f)$.

I.A.: Für $l \leq 2$ trivial, da $k(\Omega) > 2$.

I.S.: Sei F optimale Ω–Formel für f.
Falls $depth(F)$ zu groß, dann wird F umgebaut:
Seien F_l, F_r linker bzw. rechter Unterbaum von F.
F_l berechne f_l, F_r berechne f_r und
$size(F_l) = l_l$, $size(F_r) = l_r$, $size F = l$
<u>O.B.d.A.:</u> $l_l \leq l_r$
Es gilt $l_l + l_r = l - 1$, $0 \leq l_l \leq \frac{(l-1)}{2}$, $1 \leq \frac{(l-1)}{2} \leq l_r \leq l - 1$.

Wir können die I.V. auf F_l anwenden.
Was ist mit F_r ?
Sei F_0 kleinster Unterbaum von F_r mit $size(F_0) \geq \frac{l_r}{3}$.
Dann gilt $1 \leq \lceil\frac{l_r}{3}\rceil \leq l_o \leq 2 \cdot (\lceil\frac{l_r}{3}\rceil - 1) + 1 \leq \frac{2l_r+1}{3}$.
F_0 berechne f_0.
$f_{r,i}$ sei die Funktion, die von F_r nach Ersetzung von F_0 durch die Konstante $i \in \{0,1\}$ berechnet wird.
Dann ist $f_r = sel(f_0, f_{r,0}, f_{r,1})$.
Berechne zunächst f_0, $f_{r,0}$, $f_{r,1}$ parallel und dann f_r.

Es folgt
$$\begin{aligned} D_\Omega(f_r) &\leq D_\Omega(sel) + \max\{D_\Omega(f_0), D_\Omega(f_{r,0}), D_\Omega(f_{r,1})\}, \\ D_\Omega(f) &\leq 1 + \max\{D_\Omega(f_l), D_\Omega(f_r)\}. \end{aligned}$$

Nach Voraussetzung gilt wegen
$$\begin{array}{rcccccc} L_\Omega(f_0) & \leq & \frac{2l_r+1}{3} & \leq & \frac{2l-1}{3} & & \\ L_\Omega(f_{r,i}) & \leq & (l_r - l_0) & \leq & \frac{2l_r}{3} & \leq & \frac{2l-1}{3}, \end{array}$$

daß $D_\Omega(f_r) \leq D_\Omega(sel) + k(\Omega)\log(\frac{2l-1}{3} + 1)$.
Daraus folgt $D_\Omega(f) = D_\Omega(sel) + k(\Omega)\log(\frac{2}{3}) + k(\Omega)\log(l+1)$ da $D_\Omega(f_l) \leq D_\Omega(f_2)$ wegen $L_\Omega(f_l) = l_l = \frac{l-1}{2}$. ■

Die untere Schranke ist optimal zum Beispiel für:
$$f(x_1, \ldots, x_n) = x_1 \cdot \ldots \cdot x_n.$$

Definition 3.3.
$\mathcal{P}_{\Omega-F} = \{L : L$ wird von Familien von Ω–Formeln polynomialer Größe erkannt$\}$.

Korollar 3.1.
$\mathcal{P}_F = \mathcal{NC}_1(= \bigcup_{k\geq 0} SIZE, DEPTH(n^k, \log n) = DEPTH(\log n))$.

Beweis.
Folgt mit dem Satz von Spira. ■

<u>Allgemeinere Vermutung:</u> $\mathcal{P}_F \overset{\subset}{\neq} \mathcal{P}_{Schaltkreis}(\mathcal{NC}_1 \overset{\subset}{\neq} \mathcal{P}_{S}chaltkreis)$.

Wir wissen, für vollständige Basen Ω, Ω' und Konstanten $c = c(\Omega, \Omega')$ gilt
$$D_\Omega(f) \leq c \cdot D_{\Omega'}(f).$$

Korollar 3.2.
Seien Ω, $\Omega' \subseteq \mathbb{B}_2$ zwei vollständige Basen. Dann gilt
$$L_\Omega(f) \leq (L_{\Omega'}(f) + 1)^{ck(\Omega')} - 1$$
für ein geeignetes c.
<u>D.h.:</u> die Formelgrößen bzgl. verschiedener Basen sind polynomial vebunden.

3.3 Typische Formelgröße

Satz 3.3.
1. Für fast alle Boolesche Funktionen $f \in \mathbb{B}_n$ gilt:

$$L(f) = \Omega(\frac{2^n}{\log n}).$$

2. Für fast alle Boolesche Funktionen $f \in \mathbb{B}_n$ gilt:

$$L(f) = O(\frac{2^n}{\log n}).$$

Beweis.

1. **Lemma 3.1.**
   Es können höchstens $\alpha(n, b) := (n+2)^{b+1} \cdot 16^b \cdot 4^b$ Funktionen $f \in \mathbb{B}_n$ von ($\mathbb{B}_2-$)Formeln der Größe b berechnet werden.

   Beweis.
   ÜA.

   Sei $\mathbb{B}_n^*$, $\sharp\mathbb{B}_n^* = 2^{2^n(1-\log^{-1} n)}$, die Menge der Booleschen Funktionen $f \in \mathbb{B}_n$ mit den kleinsten Formeln.
   Sei $l_n^* := \max\{L(f) | f \in \mathbb{B}_n^*\}$.
   Offenbar gilt dann $\alpha(n, l_n^*) \geq 2^{2^n(1-\log^{-1} n)}$. $\quad (*)$
   $(*)$ ist für große n noch nicht erfüllt für $l := 2^n \log^{-1} n(1 - \log\log^{-1} n)$. Folglich existiert ein $f \in \mathbb{B}_n^*$ mit $L(f) \geq l$.
   Also gilt für alle Funktionen $f \in \mathbb{B}_n - \mathbb{B}_n^*$: $L(f) \geq l$.
   Wegen
   $$\begin{aligned} \sharp(\mathbb{B}_n - \mathbb{B}_n^*) &= 2^{2^n} - 2^{2^n(1-\log^{-1} n)} \\ &= 2^{2^n}(1 - 2^{-2^n} \log^{-1} n) \\ &\to 2^{2^n} \quad (n \to \infty). \end{aligned}$$
   und damit
   $$\frac{\sharp\{f \in \mathbb{B}_n | L(f) < l\}}{\sharp\mathbb{B}_n} \to 0 \quad (n \to \infty).$$

2. Folgt aus dem Nachweis über Lupanov's (k, s)–Repräsentation. ■

3.4 Untere Schranken für die Formelgröße

a) Methode von Hodes/Specker Idee:

Funktionen mit einfachen Formeln geringer Größe haben einfach symmetrische Funktionen auf nicht zu wenigen Variablen als Unterfunktion.

Theorem 3.2.
Für alle binären Basen Ω gibt es ein $\varepsilon_\Omega > 0$, so daß für alle Funktionen $f \in \mathbb{B}_n$ und für alle $r \geq 0$ mit

$$L_\Omega(f) \leq \varepsilon_\Omega n(\log\log n - r)$$

gilt:
es gibt $n - r$ Variablen, nach deren 0–Setzung eine symmetrische Unterfunktion $f' \in \mathbb{B}_r$ entsteht, für deren Wertevektor $v(f') = (v_0, v_1, \ldots, v_r)$ gilt

$$v_1 = v_3 = v_5 = \ldots \text{ und } \quad v_2 = v_4 = \ldots . \quad \blacksquare$$

Anwendung:

Korollar 3.3.
$L(T_2^n) = \Omega(n \log\log n)$.

Beweis.
Sei $r = 3$.
Wir ersetzen beliebige $n - 3$ Variablen durch 0 und erhalten T_2^3 als Unterfunktion.
Es gilt dann $v(T_2^3) = (0, 0, 1, 1)$ und $v_1 \neq v_3$. $\blacksquare$

b) Methode von Fischer, Meyer, Peterson

Idee:
Einfache Funktionen besitzen sehr einfache Unterfunktionen und erhalten bessere untere Schranken für weniger Funktionen.

Definition 3.4.
$f(x_1, \ldots, x_n) = x_{i_1} \oplus \ldots \oplus x_{i_r} \oplus c$, wobei $i_1, \ldots, i_r \in \{1, \ldots, n\}$, heißt *affine Funktion.*
r heißt *Größe von f.*
Die Unterfunktion f' von f heißt *zentrale Einschränkung*, falls für die Zahl n_1, der durch 1 ersetzten Variablen, und n_0, der durch 0 ersetzten Variablen, gilt $n_1 - n_0 \in \{0, 1\}$.

Theorem 3.3.
Sei $f \in \mathbb{B}_n$. Sei $a(f)$ die maximale Größe einer durch die zentrale Einschränkung gewonnenen affinen Unterfunktion von f. Dann gilt:

$$L(f) \geq \varepsilon n(\log n - \log a(f)).$$

für ein geeignetes ε. $\blacksquare$

Anwendung:

$C^n_{0,4}(x_1, \ldots, x_n) = 1 :\Leftrightarrow \sum\limits_{i=1}^{n} x_i \equiv 0 \bmod 4.$

Satz 3.4.
$L(C^n_{0,4}) = \Theta(n \log n)$.

Beweis.
Es ist zu zeigen: $L(C^n_{0,4}) = \Omega(n \log n)$.

Lemma 3.2.
Sei $k \in I\!N$. Gilt $f(x) = c$ für alle x mit $\sum\limits_{i=1}^{n} x_i = k$ und gilt $f(x) = \overline{c}$ für alle x mit $\sum\limits_{i=1}^{n} x_i = k+2$, dann gilt $L(f) \geq \varepsilon' n \log(\min\{k, n-k\})$ für ein geeignetes $\varepsilon' > 0$.

Beweis.
O.B.d.A.: Sei $k \leq \lfloor \frac{n}{2} \rfloor$.

1. Fall: $k = \lfloor \frac{n}{2} \rfloor$.

Es genügt zu zeigen: $a(f) \leq 2$.
Wir betrachten die zentrale Einschränkung $f' \in I\!B_3$.
Behauptung.
f' ist nicht affin mit $a(f') \geq 3$.
Beweis.
Da f' die zentrale Einschränkung ist , folgt
$\lceil \frac{n-3}{2} \rceil$ Variablen werden durch 1 ersetzt (n_1) und
$\lfloor \frac{n-3}{2} \rfloor$ Variablen werden durch 0 ersetzt (n_0). Wegen $k - \lceil \frac{n-3}{2} \rceil = 1$ gilt
$f'(1,0,0) = c$ und daraus folgt $f'(1,1,1) = \overline{c}$.
Ann.: f' ist affin mit $a(f') = 3$.
Dann ist $f'(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3 \oplus d$.
Wegen $f'(1,0,0) = c$, gilt $d = \overline{c}$.
Also $f'(1,1,1) = c$, was ein Widerspruch ist.

2. Fall: Sei $k < \lfloor \frac{n}{2} \rfloor$.

Sei F die optimale Formel für f.
Sei b_i die Zahl der mit x_i markierten Blätter von F.
Ersetze die $n - 2k$ Variablen durch 0, die die größten b–Werte besitzen.
Wir erhalten $f' \in I\!B_{2k}$.
Mit dem ersten Fall folgt $L(f') \geq \varepsilon' 2k \log k$.
Also existiert ein x_i, so daß $\geq \varepsilon' \log k$ Blätter der Unterformel F' von F mit x_i markiert sind. Nach Konstruktion folgt $b_j \geq \varepsilon' \log k$ für alle x_j, die durch 0 ersetzt wurden. Somit gilt

$$\begin{aligned} L(f) &= size(F) \\ &\geq (n-2k)\varepsilon' \log k + \varepsilon'(2k) \log k \\ &= \varepsilon' n \log k, \text{ wegen dem Lemma.} \quad \blacksquare \end{aligned}$$

Anwendung des Lemmas auf $C_{0,4}^n$:
Jedes Teilwort von $v(C_{0,4}^n)$ der Länge 4 enthält höchstens eine 1.
Wähle ein k mit $\frac{n}{2} - 2 \leq k \leq \frac{n}{2} + 2$.
Mit dem Lemma folgt dann $L(C_{0,4}^n) = \Omega(n \log n)$.

Weiter ist zu zeigen: $L(C_{0,4}^n) = O(n \log n)$.
Sei o.B.d.A.: $n = 2^m$.
Sei $D_1^n(x)$, $D_0^n(x)$ die letzten beiden Bits von $\sum\limits_{i=1}^{n} x_i$.
Dann ist $C_{0,4}^n(x) = NOR(D_1^n(x), D_0^n(x))$.
Offenbar gilt $L(D_0^n(x)) = n - 1$ da $D_0^n(x) = x_1 \oplus \ldots \oplus x_n$ und $L(D_1^n(x)) \leq n \log n$.

Beweis dieser Aussage:
Sei $x = (x', x'')$ mit $x', x'' \in \{0,1\}^{\frac{n}{2}}$. Dann folgt

$$D_1^n(x) = D_1^{\frac{n}{2}}(x') \oplus D_1^{\frac{n}{2}}(x'') \oplus (D_0^{\frac{n}{2}}(x') \wedge D_0^{\frac{n}{2}}(x'')).$$

Und somit $L(D_1^n) \leq 2L(D_1^{\frac{n}{2}}) + n$ und $L(D_1^2) = 2$.
Also $L(D_1^n(x)) \leq n \log n$.
Insgesamt gilt also $L(C_{0,4}^n) \leq n(1 + \log n)$. ■

c) Methode von Nechiporuk

Beobachtung:
Funktionen mit vielen verschiedenen Unterfunktionen können keine kleinen Formeln besitzen.

Definition 3.5.
Sei $S \subseteq X$ eine Menge von Variablen. Alle auf S definierten Unterfunktionen heißen S–*Unterfunktion.*

Beispiel 3.2.
Sei $X = \{x_1, x_2, x_3\}$, $S = \{x_1, x_2\}$ und $f(x) = x_1 \oplus x_2 \oplus x_3$.
Als S–Unterfunktion von f erhält man:

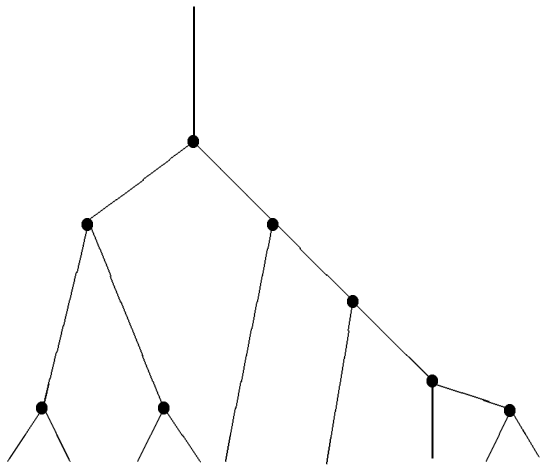
$$\begin{aligned} f|_{x_3=0}(x_1, x_2) &= x_1 \oplus x_2 \oplus 0 &= x_1 \oplus x_2, \\ f|_{x_3=1}(x_1, x_2) &= x_1 \oplus x_2 \oplus 1 &= \overline{x_1 \oplus x_2} \end{aligned}$$

Satz 3.5.
Sei $f \in \mathbb{B}_n$ und hängt wesentlich von allen Variablen ab. Sei $S_1, \ldots, S_k \subseteq X$ disjunkte Menge von Variablen und sei s_i die Zahl der S_i–Unterfunktionen, $i = 1, \ldots, k$.
Dann gilt:

$$L(f) \geq \frac{1}{4} \sum_{i=1}^{k} \log s_i.$$

Beweis.
Sei F die optimale Formel für f, a_i Zahl der mit $x_j \in S_i$ markierten Blätter von F.
Zu zeigen ist :$a_i \geq \frac{1}{4} \log s_i$.
Sei T_i ein Unterbaum von F, der alle mit einem

- $x_j \in S_i$ markierten Blätter enthält und
- alle Wege von diesen Blättern zur Wurzel enthält.



Sei ω_i Menge der Eingabeknoten von T_i mit Ingrad 2.
Dann ist $\sharp\omega_i = a_i - 1$.
Es genügt zu zeigen: $\sharp\omega_i \geq \frac{1}{4} \log s_i - 1$.
Sei P_i die Menge der Wege in T_i, die

- in einem Blatt oder einem Knoten von ω_i starten,
- in der Wurzel oder in einem Knoten von ω_i enden,
- keinen Knoten von ω_i als inneren Knoten enthalten.

Es gilt $\sharp P_i \leq 2(\sharp\omega_i + 1)$ (da die Knoten von ω_i fan–in 2 haben, also in jedem Knoten von ω_i höchstens zwei Wege enden können).
Fixiere die Variablen $x_k \notin S_i$.
Man erhält s_i verschiedene Unterformeln.
Sei $p \in P_i$ und alle $x_k \notin S_i$ fixiert und h werde im ersten Knoten von p berechnet, dann wird entlang der letzten Kante von p $0, 1, h$ oder $\overline{h}$ berechnet, da keine Variable $x_k \in S_i$ auf die Berechnung Einfluß hat. T_i besteht genau aus den Wegen von P_i.
Folglich ist die Zahl der S_i–Unterfunktionen $\leq 4^{\sharp P_i}$.
Also gilt $s_i \leq 4^{\sharp P_i} \leq 4^{2\cdot(\sharp\omega_i+1)} = 4^{2a_i} = 2^{4a_i}$.

Und somit $\log s_i \leq 4a_i$. $\blacksquare$

Welche Schranken können mit Nechiporuk erzielt werden ?

Satz 3.6.
Sei $N(S_1,\ldots,S_n) := \frac{1}{4}\sum\limits_{1\leq i\leq k} \log s_i$. *Dann gilt* $N(S_1,\ldots,S_n) \leq \frac{2n^2}{\log n}$.

Beweis.
Sei $t_i := \sharp S_i$.
Dann gibt es 2^{n-t_i} Möglichkeiten, die Variable $x_j \notin S_i$ durch Konstanten zu ersetzen.
Somit gibt es nicht mehr als $2^{2^{t_i}}$ Funktionen auf S_i.
Also ist $\log s_i \leq \min\{n-t_i, 2^{t_i}\}$.
Sei $1 \leq q \leq k$ mit o.B.d.A. $t_i > \log n$ für alle $i \geq q$.
Da S_i disjunkt, gilt $q < \frac{n}{\log n}$ und $\sum\limits_{1\leq i\leq q} \log s_i \leq \sum\limits_{1\leq i\leq q}(n-t_i) \leq \frac{n^2}{\log n}$.
Weiter gilt $\sum\limits_{q<i\leq k} \log s_i \leq \sum\limits_{q<i\leq k} 2^{t_i}$. Da $x \rightarrow 2^x$ konvex, ist $\sum\limits_{q<i\leq k} \log s_i$ maximal, falls für möglichst viele i gilt $t_i = \log n$.
Folglich ist $\sum\limits_{q<i\leq k} \log s_i \leq \frac{n^2}{\log n}$. $\blacksquare$

Anwendung:

Definition 3.6.
Die *Storage Access Funktion* $ISA_n \in \mathbb{B}_n, n = 2p+k, p = 2^{2^l}$ und $k = \log p - \log\log p$, ist definiert auf den Variablen $x = (x_1,\ldots,x_p), y = (y_0,\ldots,y_{p-1})$ und $a = (a_0,\ldots,a_{k-1})$.
Sei $d = (x_{\|a\|\log p+1},\ldots,x_{(\|a\|+1)\log p})$, wobei $\|a\|$ die durch a dargestellte Binärzahl bezeichnet, dann ist

$$ISA_n(x,y,a) = y_{\|d\|}.$$

Beispiel 3.3.
Sei $p = 4$.
Es folgt $k = 2 - 1 = 1$ und $n = 9$.
Bestimme $ISA_9(x_1,x_2,x_3,x_4,y_0,y_1,y_2,y_3,a_0)$.
Z.B. für $a_0 = 0$. Dann ist $d = (x_1,x_2)$ und $ISA_9(x_1,x_2,x_3,x_4,y_0,y_1,y_2,y_3,a_0) = y_{\|(x_1,x_2)\|}$.
Man erhält

$$\begin{aligned} \text{z.B. für:} \quad d = (0,1) \quad &\Rightarrow \quad ISA_9() = y_1 \\ d = (1,1) \quad &\Rightarrow \quad ISA_9() = y_3 \end{aligned}$$

Z.B. für $a_0 = 1$. Dann ist $d = (x_3,x_4)$ und $ISA_9(x_1,x_2,x_3,x_4,y_0,y_1,y_2,y_3,a_0) = y_{\|(x_3,x_4)\|}$.
Man erhält

$$\text{z.B. für:} \quad d = (1,0) \quad \Rightarrow \quad ISA_9() = y_2.$$

Indirekte Adressierung. ∎

Satz 3.7.
$L(ISA_n) = \Omega(\frac{n^2}{\log n})$.
(Aber: $C(ISA_n) \leq 2n + o(n)$.)

Beweis.
Sei $S_i = \{x_{i \log p+1}, \ldots, x_{(i+1)\log p}\}$ für $0 \leq i \leq \frac{p}{\log p} - 1$.
Fixiere ein i.
Ersetze a, so daß $\|a\| = i$.
Dann liefert y eine Wertetabelle für S_i–Unterfunktionen.
Jeder der 2^p verschiedenen Werte liefert eine andere S_i–Unterfunktion.
Somit ist $s_i \geq 2^p$. Mit Nechiporuk folgt $L(ISA_n) \geq \frac{1}{4} \cdot \frac{p^2}{\log p} = \Omega(\frac{n^2}{\log n})$. ∎

Weitere Anwendungen:

Es gilt

- $L(det_n) = \Omega(n^3)$ für die Determinantenfunktion $det_n \in \mathbb{B}_{n^2}$ mit

$$det_n(x_{11}, \ldots, x_{nn}) = \bigoplus_{\pi \in \gamma_n} x_{1,\pi(1)} \cdot \ldots \cdot x_{n,\pi(n)}.$$

- $L(d_n, m) = \Omega((n-m)^3)$, falls $m \geq 3$, für die Clique–Funktion $Cl_{n,m} \in \mathbb{M}_{\binom{n}{2}}$, wobei

$$Cl_{n,m}(G(x)) = \begin{cases} 1 & G(x) \text{ enthält } m\text{–Clique,} \\ 0 & \text{sonst.} \end{cases}$$

Kapitel 4

Schaltkreise beschränkter Tiefe

4.1 Modell

Wir schränken die Schaltkreis–Struktur weiter ein, um bessere untere Schranken beweisen zu können:
Betrachte Schaltkreis mit <u>konstanter</u> Tiefe .
Um alle Eingaben lesen zu können, brauchen wir Basen mit <u>unbeschränktem</u> fan–in.

Erinnerung:

$$\left.\begin{array}{rcl} \mathcal{U} & = & \{\neg, \cdot_i, +_i | i \in I\!N\} \\ \mathcal{Z} & = & \{\cdot_i, \oplus_i | i \in I\!N\} \\ \mathcal{T} & = & \{t_k^n, t_{\leq k}^n \,|\, n, k \in I\!N\} \end{array}\right\} \text{sind vollständige Basen mit \underline{unbeschränktem} fan–in.}$$

(vgl. $II.4$).

Wir haben $\mathcal{U}$–Schaltkreise stets in der Standardform betrachtet:

- Negation lediglich bei Eingabeknoten,
- $\cdot$ und $+$ ($\mathcal{U}$) bzw. $\cdot$ und $\oplus$ ($\mathcal{Z}$) Gatter alternieren,
- synchron (gleiche Entfernung zu Eingabeknoten auf allen existierenden Verbindungswegen)

$$\begin{array}{rcl} AC_0 & = & \bigcup\limits_{k\geq 0, d\geq 0} SIZE, DEPTH_{\mathcal{U}}(n^k, d) \\ & = & \bigcup\limits_{d\geq 0} AC_{0,d} \quad , \\ AC_{0,d} & = & \bigcup\limits_{k\geq 0} SIZE, DEPTH_{\mathcal{U}}(n^k, d). \end{array}$$

AC_0 enthält interessante Funktionen:

Satz 4.1.

(1) $ADD \in AC_0$, *(vgl. II.7)*
(wir haben sogar $ADD \in AC_{0,3}$ *gezeigt).*

(2) $MAX \in AC_0$,
$MAX = (MAX_{m^2})$,
$MAX_{m^2} \in I\!B_{m^2,m}$,
$MAX_{m^2}(a_{11}, \ldots, a_{1m}, \ldots, a_{n1}, \ldots, a_{nm}) = (a_{i1}, \ldots, a_{im})$,
falls $\|a_i\| = \max\{\|a_i\|, \ldots, \|a_m\|\}$, $a_i \in \{0,1\}^m$ *m–bit Zahlen.*

(3) $BMAT$ *= Boolesche Matrixmultiplikation,*
$BMAT \in AC_0$.

Beweis.

(1) bereits gezeigt.

(3) $BMAT \in AC_{0,2}$ gilt nach Definition.

(2) Seien $a_1, \ldots, a_m(a_1 = (a_{11}, \ldots, a_{1,m}), \ldots, a_m = (a_{m1}, \ldots, a_{mm}))$ m–bit Zahlen.
Berechne in Tiefe 4 mit $O(m^4)$ Gattern
$c_{ij} = 1 \Leftrightarrow a_i \geq a_j (1 \leq i, j \leq m)$
$d_i := \prod_{j=1}^{m} c_{ij}$
$d_i = 1 \Leftrightarrow \|a_i\| = \max(\|a_i\|, \ldots, \|a_m\|)$
Berechne j–tes Ausgabebit als $\sum_{i=1}^{n} a_{ij} d_i$
Dann folgt $MAX \in AC_{0,6}$.
(Geht übrigens noch besser: $MAX \in AC_{0,4}$). ■

4.2 Exponentielle untere Schranke für $\mathcal{U}$–Schaltkreise beschränkter Tiefe

Zumindest für $\mathcal{U}$–Schaltkreis beschränkter Tiefe können exponentielle untere Schranken für explizit gegebene Funktionen bewiesen werden.

Idee: (First, Saxe, Sipser (1984); Yao (1985); Hastad (1986))

(1) Beweisen, daß $\mathcal{U}$–Schaltkreis der Tiefe 2 für betrachtete Funktion groß sind.

(2) Zeigen, daß kleine Tiefe k $\mathcal{U}$–Schaltkreise für die betrachtete Funktion in kleine Tiefe $k-1$ $\mathcal{U}$–Schaltkreise überführt werden können.

Betrachte $PAR_n \in I\!B_n$ Parity–Funktion; $PAR_n(x_1, \ldots, x_n) = \bigoplus_{i=1}^{n} x_i$.

Satz 4.2.
$\mathcal{U}$–Schaltkreise der Tiefe 2, die PAR_n berechnen, haben die Größe $\geq 2^{n-1}$.

Beweis.

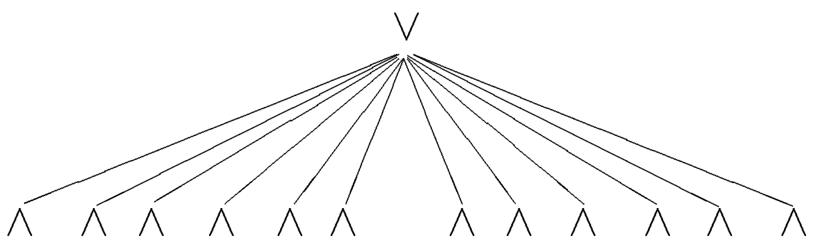
1. Fall: Sei S $OR - AND$–Schaltkreis der Tiefe 2 für PAR_n, d.h. S sei eine DNF.
Offenbar gilt $size(S) = k + 1$.
$PAR_n(x) = \sum_{i=1}^{k} m_i(x)$.
Sei i beliebig.
Behauptung.
$|m_i| = n$.
Ann.: Sei $|m_i| < n$.
Dann existieren zwei Eingaben a, b mit

$$m_i(a) = m_i(b) = 1 \text{ und } \sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i + 1.$$

Somit gilt $1 = \sum_{i=1}^{k} m_i(a) = \sum_{i=1}^{k} m_i(b)$ und folglich $PAR_n(a) \neq PAR_n(b)$, was ein Widerspruch ist.
Also gilt $\sharp m_i^{-1}(1) = 1$ für alle i, somit $k = 2^{n-1}$.

2. Fall: Sei S $AND - OR$–Schaltkreis der Tiefe 2 für PAR_n.
Analoge Betrachtung für kNF. ■

Wir wollen nun Tiefe k Schaltkreis in Tiefe $k-1$ Schaltkreis umformen.
Wir können $AND - OR$–Schaltkreis (kNF) der untersten beiden Stufen in $OR - AND$–Schaltkreis (DNF) umwandeln (die Groesse des Schaltkreises kann dabei explodieren):



Wir können adjazente v–Ebenen vereinigen: Wir haben damit einen sehr großen Tiefe $k-1$ Schaltkreis erhalten.

Problem: Tiefe $k-1$ Schaltkreis ist sehr groß.

Idee: Einige Variablen konstant setzen, um Schaltkreis zu vereinfachen und damit zu verkleinern.

Problem: Die Auswahl der Variablen.

Idee: Auswahl durch Auswürfeln.

Definition 4.1.
Eine *Einschränkung* ist eine Abbildung $\rho : X \longrightarrow \{0, 1, *\}$ mit

$$\rho(x_i) = \begin{cases} 0 & \text{bedeutet "}x_i\text{ konstant 0 setzen",} \\ 1 & \text{bedeutet "}x_i\text{ konstant 1 setzen",} \\ * & \text{bedeutet "Variable bleibt unersetzt".} \end{cases}$$

Bezeichnung.
$f|_\rho$: f nach Anwendung der Einschränkung ρ.

Beispiel 4.1.
Sei $f(x_1, x_2, x_3, x_4, x_5) = 1 \Leftrightarrow \sum_{i=1}^{n} x_i \geq 3$.

$\rho : x_1 \longrightarrow 1$
$\rho : x_2 \longrightarrow *$
$\rho : x_3 \longrightarrow *$
$\rho : x_4 \longrightarrow 1$
$\rho : x_5 \longrightarrow *$

$f|_\rho(x_2, x_3, x_5) = 1 \Leftrightarrow x_2 + x_3 + x_5 \geq 1$. **Beobachtung.**

$PAR|_\rho = PAR$ oder $\overline{PAR}$.

Wir müssen zur Verkleinerung der Schaltkreise die Variablen so konstant setzen, daß möglichst viele Unterschaltkreise verschwinden.
Dies ist schwierig zu übersehen, deshalb folgende Definition:

Definition 4.2.
Für eine *zufällige Einschränkung* $\rho \in R_p$ gilt

$$\begin{array}{ll} \rho(x_i) = 0 & \text{mit Wahrscheinlichkeit } \frac{1}{2} - \frac{p}{2}, \\ \rho(x_i) = 1 & \text{mit Wahrscheinlichkeit } \frac{1}{2} - \frac{p}{2}, \\ \rho(x_i) = * & \text{mit Wahrscheinlichkeit } p, \end{array}$$

unabhängig für alle x_i.

Lemma 4.1. *(Switching Lemma, Hastad 1986)*
Sei S ein $AND - OR$–Schaltkreis der Größe $\leq t$ für eine Funktion F und $\rho \in R_p$ eine zufällige Einschränkung.
Dann ist die Wahrscheinlichkeit, daß $F|\rho$ <u>nicht</u> durch einen $OR - AND$–Schaltkreis der

Größe $\leq s$ berechnet werden kann, kleiner als α^s, wobei α die eindeutig bestimmte positive Wurzel der Gleichung

$$(1+\frac{4p}{1+p}\cdot\frac{1}{\alpha})^t = (1+\frac{2p}{1+p}\cdot\frac{1}{\alpha})^t + 1$$

ist. ■

Bemerkung.
Eine analoge Aussage gilt für $OR - AND$–Schaltkreis. (Anstelle von S wird $\neg S$ betrachtet.)

Daraus folgt:

Theorem 4.1.
Es gibt keinen Tiefe k Schaltkreis für PAR_n der Größe $\leq 2^{(\frac{1}{10})^{\frac{k}{k-1}} n^{\frac{1}{k-1}}}$ falls $n \geq n_0$ für ein geeignetes n_0.

Korollar 4.1.
$PAR_n \notin SIZE, DEPTH(poly, const)$, d.h. $PAR_n \notin AC_0$. ■

4.3 Exponentielle untere Schranke für $\mathcal{Z}$–Schaltkreise beschränkter Tiefe

Wir wissen $AC_0 \subseteq ZC_0 \subseteq NC^1$,

$$\left.\begin{array}{lll} PAR & \notin & AC_0 \\ PAR & \in & ZC_0 \end{array}\right\} \Rightarrow AC_0 \subsetneqq ZC_0$$

Wie sieht das Verhalten von ZC_0 und NC^1 aus?

Satz 4.3. *(Razborov 1986)*
Jeder Schaltkreis zur Berechnung von MAJ_n der Tiefe k hat Größe $\geq 2^{cn^{\frac{1}{k+1}}}$.

Beweisidee: Algebraische Approximation.

Wir wissen: Jede Boolesche Funktion kann als $\{\oplus, \cdot\}$–Polynom dargestellt werden.
Zu zeigen: Jede Funktion aus ZC_0 kann gut approximiert werden durch $\{\oplus, \cdot\}$–Polynome niedrigen Grades.
<u>Aber:</u> MAJ kann nicht gut durch $\{\oplus, \cdot\}$–Polynom niedrigen Grades approximiert werden.
■

Korollar 4.2.
$ZC_0 \underset{\neq}{\subset} NC^1$.

Beweis.
$MAJ \notin ZC_0$ folgt aus dem Satz.
$MAJ \in NC^1$ (ÜA). ■

4.4 Reduktionen

Die Reduktionen spielen eine Schlüsselrolle in der Komplexitätstheorie.
Das Reduktionskonzept ist eine $\leq$–Relation, so daß $f \leq g$ impliziert, daß f nicht *schwerer* als g ist.

Bedeutung.
Sei $f \leq g$.

- Effiziente Schaltkreise für g liefern effiziente Schaltkreise für f.
- Untere Schranken für f liefern untere Schranken für g.

Definition 4.3.
Sei $f = (f_n)$, $g = (g_n)$.
Ein *g–Orakel Schaltkreis S* enthält neben $\wedge, \vee, \neg$ Gattern Gatter zur Berechnung von g_i, $\overline{g_i}$, wobei i polynomial beschränkt ist und auf jedem Weg in S höchstens eines dieser g_i, $\overline{g_i}$ Gatter vorkommt.

Definition 4.4.
Sei $f = (f_n)$, $g = (g_n)$.
f ist AC_0–reduzierbar auf g genau dann, wenn f_n mit einem g–Orakel Schaltkreis polynomialer Größe und konstanter Tiefe berechnet werden kann.
Bez.: $f \leq_{cd} g$ (*cd*: constant depth).

Satz 4.4.
(i) $\leq_{cd}$ ist transitiv.
(ii) $g \in AC_k$ und $f \leq_{cd} g$. Dann ist $f \in AC_k$.
(iii) $g \in NC_k$ und $f \leq_{cd} g$. Dann ist $f \in NC_k$.

Beweis.
zu (i):
Sei $f \leq_{cd} g$, $g \leq_{cd} h$.

Dann kann f_n mit g–Orakel Schaltkreis polynomialer Größe und konstante Tiefe berechnet werden.
Ersetze g–Orakel Schaltkreis durch h–Orakel Schaltkreis polynomialer Größe und konstanter Tiefe .
Man erhält h–Orakel Schaltkreis für f_n, der in der Größe polynomial und in der Tiefe konstant ist.

zu (ii):
Wir können f_n mit AC_0 g–Orakel Schaltkreis berechnen.
Wir ersetzen g–Orakelknoten durch AC_k Schaltkreis für g.
Größe polynomial und konstante Tiefe .

zu (iii): analoge Vorgehensweise. ■

Satz 4.5.
$PAR \leq_{cd} MAJ =_{cd} MUL =_{cd} MADD =_{cd} SORT$, wobei
MUL : Multiplikation 2er n–Bit Zahlen,
$MADD$: Multiplikation und Addition; Addition von m n–Bit Zahlen,
$SORT$: Sortierung von m n–Bit Zahlen,

Beweis.
$PAR \leq_{cd} MUL$:
Sei $X = (x_1, \ldots, x_n)$, $l = \log n$.
Wir konstruieren Zahlen u, v der Länge $n \cdot l$, nämlich

$$u = \sum_{i=1}^{n} x_i 2^{l(i-1)}, \quad v = \sum_{i=1}^{n} 2^{l(i-1)}$$

Dann gilt $u \cdot v = \sum_{i=1}^{2n} w_i \cdot 2^{l(i-1)}$. mit $w_i = \sum_{k=1}^{i} x_k$ für $i \leq n$.
Somit ist $w_n = \sum_{k=1}^{n} x_k$ und $PAR_n(x) =$ letztes Bit von w_n.

Beispiel 4.2.

$x = (1001), l = 2$
$u = 01 \quad 00 \quad 00 \quad 01$

$v = 01 \quad 01 \quad 01 \quad 01$

$u \cdot v$	$=$	01	00	00	01	$\cdot$	01	01	01	01
		01	01	01	01					
			01	00	00	01				
				01	00	00	01			
					01	00	00	01		
		01	01	01	10	01	01	01		
					$(x_1$	$(x_2$	$(x_3$			
					$+x_2$	$+x_3$	$+x_4)$			
					$+x_3$	$+x_4)$				
					$+x_4)$					

<u>$MAJ \leq_{cd} MUL$:</u>
Idee: Analog wie $PAR \leq_{cd} MUL$.
Vergleiche nun w_n mit der Binärdarstellung von $\frac{n}{2}$, d.h. untersuchen, ob wenigstens 1 Stelle der ersten $\frac{l}{2}$ Stellen von w_n 1 ist.

<u>$MUL \leq_{cd} MADD$:</u>
Um $x = (x_{n-1}, \ldots, x_0)$, $y = (y_{n-1}, \ldots, y_0)$ zu multiplizieren, werden die $2n$–Bit Zahlen $z_i = x_i,\ y,\ 2^i,\ 0 \leq i < n$, generiert und die Summen $\sum z_i$ bestimmt.

<u>$MADD \leq_{cd} MAJ$:</u>
Recht kompliziert.
Man zeigt:

$$MADD \leq_{cd} BSUM \leq_{cd} SORT \leq_{cd} Threshold \leq_{cd} MAJ,$$

wobei
$BSUM$: binäre Summe der n Eingabebits,
$SORT$: Sortieren von n–n–Bit Zahlen. ∎

Kapitel 5

Binäre Entscheidungsgraphen

Wir können Boolesche Funktionen durch Entscheidungsbäume oder Branching–Programme repräsentieren.

5.1 Modell

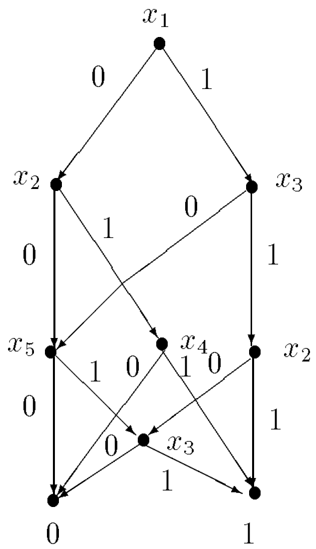
Definition 5.1.
Ein *Branching–Programm* (BP) über $X_n = \{x_1, \ldots, x_n\}$ ist ein gerichteter Graph mit genau einer Wurzel, dessen

- Senken mit Booleschen Konstanten oder 1 bewertet sind,
- innere Knoten fan–out 2 besitzen und mit Booleschen Variablen aus X_n markiert sind,
- Kanten mit 0 oder 1 bewertet sind, wobei die beiden Kanten, die einen Knoten verlassen, verschiedene Werte tragen.

Die Größe $SIZE(P)$ eines Branching–Programm P ist die Zahl der inneren Knoten.

Beispiel 5.1.

P:



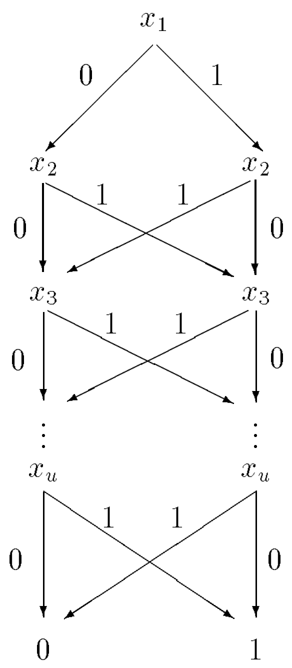
Size (P) = 7

Das Branching–Programm P berechnet die Boolesche Funktion $f_p \in \mathbb{B}_n$:
Sei $a = \{a_1, \ldots, a_n\} \in \{0,1\}^n$.
Start in der Quelle.
Ist v erreicht und ist v mit x_i markiert, so wird entlang der Kante 0 (1) weitergegangen, falls $a_i = 0$ (bzw. $a_i = 1$).
Die schließlich erreichte Senke gibt den Funktionswert $f_p(a)$ an.

Beispiel 5.2.

$f_p(0,1,0,1,1) = 1$,
$f_p(1,0,0,1,1) = 0$.

Beispiel 5.3.
Ein Branching–Programm für $PAR_n \in \mathbb{B}_n$



Definition 5.2.
Sei $f \in \mathbb{B}_n$. Die *Branching–Programm–Größe von* f ist definiert durch

$$BP(f) = min\{SIZE(P) | P \text{ ist Branching–Programm über } X_n \text{ für } f\}.$$

Frage: Läßt sich jede Funktion durch ein Branching–Programm berechnen?

Satz 5.1.
Jede Funktion $f \in \mathbb{B}_n$ läßt sich mit einem Branching–Programm berechnen.

Beweis.
<u>Induktion über n:</u>
<u>$n = 0$:</u> Trivial.
<u>$n - 1 \to n$:</u> Sei $f = \overline{x_n} f|_{x_n=0} + x_n f|_{x_n=1}$. Nach Voraussetzung existieren Branching–Programme P_0, P_1 für $f|_{x_n=0}$, $f|_{x_n=1}$.
Wir erhalten ein Branching–Programm P für f gemäß

$$
\begin{tikzcd}
 & x_n \arrow[dl, "0"'] \arrow[dr, "1"] & \\
P_0 & & P_1
\end{tikzcd}
$$

Achtung:
Diese Konstruktion von Branching–Programmen ist baumartig und hat die Größe $2^n - 1$. Oft existieren kompaktere Branching–Programme.

5.2 Simulation von Branching–Programmen durch Turing Maschinen und umgekehrt

Wir haben die Turing Maschinen–Zeit zur Schaltkreis–Größe ins Verhältnis gesetzt. Wir können auch den Turing Maschinen–Speicher zur Branching–Programm–Größe ins Verhältnis setzen.

Theorem 5.1.
Sei $f_n \in \mathbb{B}_n$; $s(f)$ sei der Speicherplatzbedarf von einer nichtuniformen Turing Maschine zur Berechnung von f.
Falls $BP(f) \geq n$ und $s(f) \geq \log n$, dann gilt:

$$
\begin{array}{lrcl}
(1) & s(f_n) & = & O(\log(BP(f_n))), \\
(2) & Branching - Programm(f_n) & = & 2^{O(s(f_n))}.
\end{array}
$$

Beweis.
<u>zu (1):</u>
Sei P_n ein optimales Branching–Programm für f_n.
Wir codieren P_n als $0-1$ String und nehmen diesen als Orakel für nichtuniforme Turing Maschine M.
Die Länge der Kodierung jedes Knotens benötigt $O(\log(BP(f_n)))$ Bits.
Die Kodierung der beiden Nachfolger jedes inneren Knotens benötigt $O(\log(BP(f_n)))$ Bits, die Kodierung des Indexes der Knotenmarkierung (Variable) benötigt $O(\log n)$ Bits.
Somit hat die Kodierung von P_n die Länge $O(BP(f_n) \cdot \log(BP(f_n))$.
Die nichtuniforme Turing Maschine M kann ein Branching–Programm Schritt für Schritt simulieren:
Auf dem Arbeitsband wird die aktuelle Knotennummer gespeichert, vom Eingabeband wird das aktuelle Eingabebit gelesen und durch Konsultation mit dem Orakel wird die Nachfolgeknotennummer ermittelt.

Der Speicherbedarf ist $s(f_n) \leq O(\log(BP(f_n)))$.

zu (2):
Sei M eine nichtuniforme Turing Maschine für f_n mit dem Speicherbedarf $s(f_n)$.
Die Zahl der Konfigurationen ist $\leq 2^{O(s(f_n))}$.
Wir simulieren M durch ein Branching–Programm P_n:
Jede Konfiguration von M wird als Knoten aufgefaßt. Die initiale Konfiguration ist die Quelle.
Die akzeptierten Konfigurationen sind die 1–Senken und
die verwerfbaren Konfigurationen sind die 0–Senken.
Die Nachfolgeknoten v_0, v_1 zum Knoten v zur Konfiguration c sind die Knoten, die zu den Konfigurationen c_0, c_1 gehören, in die M übergeht bei Eingabe von 0 bzw. 1.
Somit erhalten wir ein Branching–Programm zur Berechnung von f_n mit $SIZE(P_n) = 2^{O(s(n))}$. ■
Sei $\mathcal{P}_{Branching-Programm} = \{f | Branching - Programm(f_n)$ ist polynomial in $n\}$.

Korollar 5.1.
$\mathcal{P}_{Branching-Programm} = L/poly$.

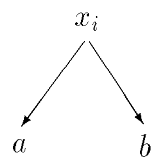
D.h. polynomiale Branching–Programme entsprechen logarithmischem Raum.
(Wir wissen: $\mathcal{P}_{Schaltkreis} = P/poly$.)

5.3 Verhältnis Branching–Programm und Schaltkreis

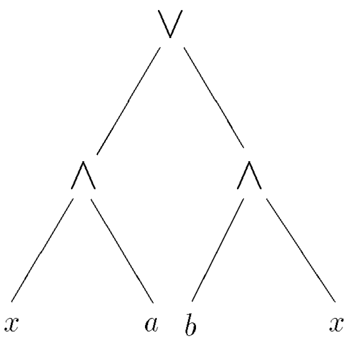
Satz 5.2.

(1)	*C(f)*	$\leq$	$3 \cdot BP(f)$	*für alle* $f \in \mathbb{B}_n$,
(2)	*BP(f)*	$\leq$	$L_\Omega(f) + 1$	*für alle* $f \in \mathbb{B}_n$.

Beweis.
zu (1):
Sei P ein Branching–Programm für f.
Ersetze Testknoten x_i in P durch



Schaltkreise für Selektfunktion.



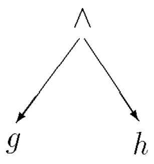
Wir erhalten einen Schaltkreis für f der Größe $3 \cdot SIZE(P)$.
zu (2):
Induktion über $l = L(f)$.
$l = 0$: trivial
$l - 1 \rightarrow l$:
Sei F optimale Formel für f mit Wurzel r.

1. Fall:

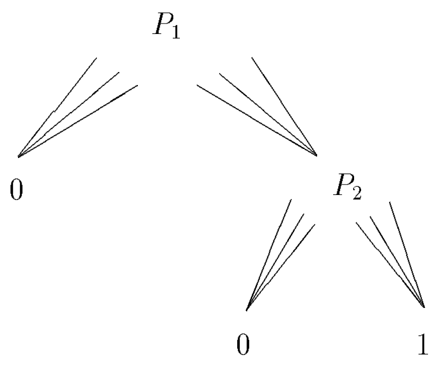
Das Gatter in r ist $\neg$. Wegen $BP(f) = BP(\overline{f})$ folgt die Behauptung aus der Induktionsvoraussetzung.

2. Fall:

Das Gatter in r ist $\wedge$.



Da $L(g) + L(h) = l - 1$ folgt mit der Induktionsvoraussetzung $BP(g) \leq L(g) + 1$ und $BP(h) \leq L(h) + 1$.
Es genügt zu zeigen, daß $BP(f) \leq BP(g) + BP(h)$.
Sei P_1 ein optimales Branching–Programm für g und P_2 ein optimales für h.
Dann ist das Programm, bei dem die 1-Senke von P_1 mit der Wurzel von P_2 identifiziert wurde, ein Branching–Programm für $g \wedge h = f$.



und $SIZE(P) = SIZE(P_1) + SIZE(P_2)$.

3. Fall:
Das Gatter in r ist $\vee$. Analog zum 2. Fall. ■

5.4 Typische Branching–Programm–Größen

Wir wissen: $Branching - Programm(f) \in [\frac{C(f)}{3}, L(f)]$.

Was ist eine typische Branching–Programm–Größe ?

Satz 5.3.
(1) Für alle $f \in \mathbb{B}_n$ gilt

$$BP(f) \geq \frac{1}{3} \cdot \frac{2^n}{n}.$$

(1) Für alle $f \in \mathbb{B}_n$ gilt

$$BP(f) = O(\frac{2^n}{n}).$$

Beweis.
zu (1):
Dies folgt aus dem letzten Satz und aus der entsprechenden Aussage über Schaltkreise.
zu (2):
Benutzung von Lupanovs (k, s)–Repräsentation wie bei Schaltkreisen. ■

5.5 Untere Schranke für Branching–Programme

- Wir können keine untere Schranke der Größe $\omega(n^2)$ erwarten, da $BP(f) \leq L(f)+1$.
- Wir könne Nechiporuk–Methode auf Branching–Programm anwenden:

Satz 5.4.
Sei $f \in \mathbb{B}_n$ und seien alle Variablen von f wesentlich, $S_1, \ldots, S_k \subseteq X$ Partition und sei s_i die Zahl der S_i–Unterfunktionen f.
Dann gilt

$$BP(f) = \Omega(\sum_{i=1}^{k} \frac{\log s_i}{\log \log s_i}) \quad .$$

Beweis.
Sei P optimales Branching–Programm für f.
Sei $N(r,t)$ die Zahl der Branching–Programme der Größe r t über r Variablen.
Dann ist die Zahl der Kanten $\leq 2t$ und $N(r,t) \leq r^t \cdot t^{2t}$.
(r^t : alle möglichen Bewertungen der inneren Knoten
t^{2t} : alle möglichen Nachfolger).
$r_i := \sharp s_i$
t_i = Zahl der Knoten aus P, die mit Variablen aus S_i markiert sind
Dann kann jede S_i–Unterfunktion kann mit einem Branching–Programm der Größe $\leq t_i$ berechnet werden.
Somit ist $s_i \leq N(r_i, t_i) \leq r_i^{t_i} \cdot t_i^{2t_i}$, falls $t_i \geq 3$. Da f wesentlich für alle Variablen ist, gilt: Man hat $r_i \leq t_i$, also $s_i \leq t_i^{3t_i}$. Folglich gilt $t_i = \Omega(\frac{\log s_i}{\log \log s_i})$.
Wegen $\sum t_i = BP(f)$ folgt die Behauptung. ■

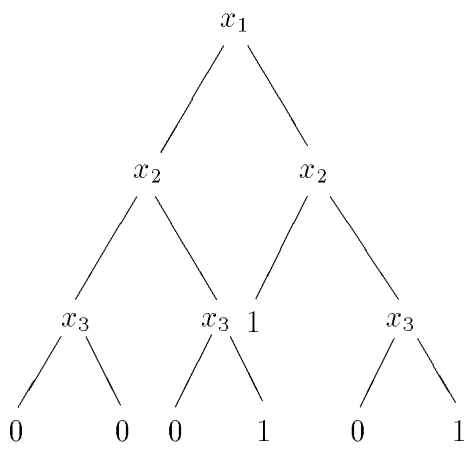
Wir erhalten eine untere Schranke zum Beispiel für

$$\begin{aligned} Branching - Programm(ISA_n) &= \Omega(\tfrac{n^2}{\log^2 n}) \quad (C(ISA_n) = O(n)), \\ Branching - Programm(det_n) &= \Omega(\tfrac{n^3}{\log n}). \end{aligned}$$

5.6 Entscheidungsbäume

Definition 5.3.
Ein baumartiges Branching–Programm heißt *Entscheidungsbaum* (DT).
$DT(f) = \min\{SIZE(T), T$ Entscheidungsbaum für $f\}$.

Beispiel 5.4.
$f(x_1, x_2, x_3) = x_1\overline{x_2} + x_1 x_3 + \overline{x_1} x_2 x_3$



Beobachtungen:

1. Alle Booleschen Funktionen $f \in \mathbb{B}_n$ können von einem Entscheidungsbaum der Tiefe n berechnet werden.
(jede Variable braucht nur einmal gelesen zu werden)

2. Die Shannon–Entwicklung für $f \in \mathbb{B}_n$ liefert einen Entscheidungsbaum für f.

Satz 5.5.
Für alle $f \in \mathbb{B}_n$ gilt:

$$D(f) \leq c \cdot \log(DT(f) + 1).$$

Beweis.
Entscheidungsbäume liefern Boolesche Formeln über der vollständigen Basis $\{sel\}$. Mit dem Satz von Spira folgt $D_{\{sel\}}(f) \leq c' \cdot \log(DT(f) + 1)$.
Wegen $D(f) \leq D_{\{sel\}}(f)$ folgt die Beh. ■

Entscheidungsbäume sind im allgemeinen recht groß.
Wir können leicht eine exponentielle untere Schranke beweisen.

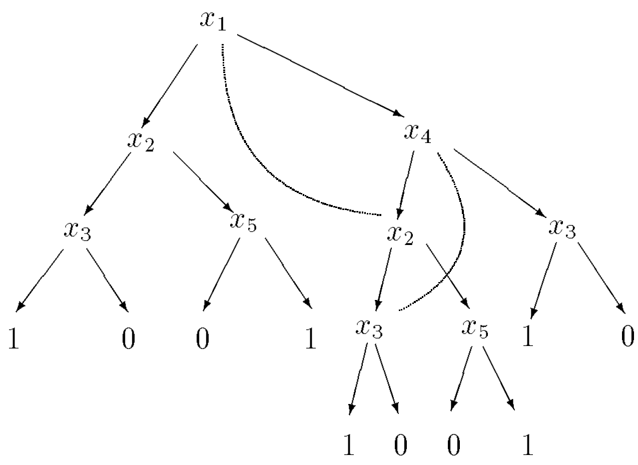
Satz 5.6.
$DT(PAR_n) \geq 2^{n-1} - 1.$

Beweis.
Betrachte die 1–Senke v von einem Entscheidungsbaum.
Die Menge der Eingaben, die v erreichen, bildet einen "Würfel".
Somit ist die Zahl der 1–Senken eines Entscheidungsbaumes für $PAR_n \geq 2^{n-1}$ und folglich ist die Zahl der inneren Knoten $\geq 2^{n-1} - 1$. ■

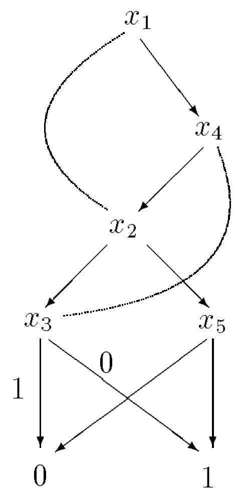
5.7 Read–Once–Only Branching Programme

Wir können isomorphe Unterbäume in Entscheidungsbäumen identifizieren.

Beispiel 5.5.



Wir erhalten hieraus ein "read–once–only Branching–Programm".



Definition 5.4.
Ein *read–once–only Branching–Programm* (BP1) ist ein Branching–Programm, bei dem

auf jedem Weg jede Variable höchstens einmal getestet wird.
$Branching - Programm1(f) = \min\{SIZE(P) | P$ BP1 für $f\}$.

Korrespondierendes Turing Maschinenmodell ist ein *eraser* Modell.
Somit geben die unteren Schranken für Branching–Programme 1 die unteren Schranken für den Raumbedarf von eraser Turing Maschinen.

Beobachtung:
Bei einem Branching–Programm 1 über X_n gibt es zu jedem Weg p von der Quelle zur Senke eine Eingabe $a = a(p) = (a_1, \ldots, a_n)$, so daß a entlang p bearbeitet wird.
Beweis.
Sei $p = (x_{i_1}, \varepsilon_1, \ldots, x_{i_m}, \varepsilon_m)$, $m \le n$, dann gilt für Belegung $a = (a_1, \ldots, a_n)$ mit

$$a_j = \begin{cases} \varepsilon_k & \text{falls } j = i_k \text{ für ein } 1 \le k \le m, \\ 1 & \text{sonst.} \end{cases} \quad \blacksquare$$

Bemerkung.

"Jeder Weg in einem BP1 P ist ein Berechnungsweg".
Diese Eigenschaft gilt nicht in uneingeschränkten Branching–Programms !

Satz 5.7.
Für alle $f \in S_n$ existiert ein optimales Branching $--$Programm$_1$, das synchron ist und in allen Variablen des l–ten Niveaus die Variable x_{l+1} testet.

Beweis.

Lemma 5.1.
Ein Branching–Programm 1 für eine symmetrische Funktion $f \in S_n$, f sei nicht konstant, hat die volle Länge n.

Beweis.(Lemma)
Ann.: Es existiert für $f \in S_n$ ein Branching–Programm 1 P mit Länge(P)$< n$.
Also wird auf jedem Weg von der Wurzel zur Senke wenigstens eine Variable nicht getestet.
Sei $v(f)$ mit $v_i \neq v_{i+1}$.
Folge dem Weg, auf dem die ersten i getesteten Variablen den Wert 1 haben und betrachte die entsprechende Eingabe.
Die übrigen $\le n - 1 - i$ Variablen auf dem Weg werden auf 0 gesetzt.
Wir erhalten eine Eingabe a.
Werde x_k nicht getestet, dann betrachte a' mit

$$a'_j = \begin{cases} a_j & j \neq k, \\ 1 & j = k. \end{cases}$$

Offenbar gilt $P(a) = P(a')$ aber $f(a) \neq f(a')$. $\blacksquare$

Sei P optimales Branching–Programm 1 für $f \in \mathcal{S}_n$.
Sie $v \in P$ und seien p, p' Vebindungswege von der Quelle zu v.

Behauptung

1. Länge(p) = Länge(p').
2. $f_p = f_{p'}$, wobei f_p die Unterfunktion von f ist, die durch Festsetzung der entlang p getesteten Variablen i der durch p vorgeschriebenen Weise entsteht.
3. Auf p, p' werden die gleichen Variablen getestet.

Beweis.(Beh.)
O.B.d.A. seien f_p, f_p' nicht konstant.
f_p ist eine symmetrische Funktion auf n–Länge(p) Variablen.
Mit dem Lemma folgt, daß der längste in v startende Weg p'' zur Senke die Länge n–Länge(p) hat.
Analoges gilt für $f_{p'}$.
Somit gilt Länge(p) = Länge(p') (1.).
Auf p, p' werden alle Variablen getestet, die nicht auf p'' getestet werden (2.) und es gilt $f_p = f_{p_v} = f_{p'}$ (3.).

Somit ist P synchron.
Wir ordnen allen Knoten des Niveaus l die Variable x_{l+1} zu und erhalten ein Branching–Programm 1 P'.
Betr.: $f_{p'} = f_p$.
Klar, da f_p symmetrisch, kommt es auf die Anzahl der 1en in einer Eingabe an und nicht auf die Position ihres Auftretens. ■

Bemerkung.
Optimale read-once-only Branching–Programme für symmetrische Funktionen können effizient konstruiert werden.
Es kann gezeigt werden:

$$Branching - Programm1(f) \leq \frac{n^2}{2} - n\log n + O(n)$$

für alle symmetrischen Funktionen $f \in \mathcal{S}_n$.

Wir können für bestimmte Funktionen exponentielle untere Schranken beweisen:
Betrachte: $f \in \mathbb{B}_{n^2}$.

$$f(A) = 1 \quad \Leftrightarrow \quad \begin{array}{l} A \text{ ist Permutationsmatrix, d.h.} \\ A \text{ ist } n \times n \text{ Matrix mit} \\ \text{Koeffizienten aus } \{0,1\}; \\ \text{in jeder Spalte und in jeder} \\ \text{Zeile ist genau eine 1.} \end{array}$$

Wir haben eine eineindeutige Beziehung:

$$\sigma \longrightarrow A^\sigma = (a_{ij}) \text{ mit } a_{ij} = \begin{cases} 1 & j = \sigma(i), \\ 0 & \text{sonst.} \end{cases}$$

Satz 5.8.
$BP1(f) = 2^{\Omega(n)}$.

Beweis. "Cut and Paste"–Technik.
Sei P ein Branching–Programm 1 für f über $\{x_{11}, \ldots, x_{nn}\}$.
O.B.d.A. sei n gerade.
Sei p_1 ein Weg von der Quelle zum Knoten v,
p_2, p_3 seien Wege, die in v starten.
Dann sind p_1, p_2 und p_1, p_3 Berechnungswege (BP1–Eigenschaft).
Sei p ein akzeptierender Weg in P, dann existiert genau eine Permutationsmatrix $A = A^\sigma$, die entlang p akzeptiert wird.
(<u>Ann.:</u> Es existieren zwei Permutationsmatrizen, die entlang p akzeptiert werden.
Dann läßt sich sofort eine Matrix A' angeben, die entlang p akzeptiert wird, aber keine Permutationsmatrix ist).
Fixiere für alle $\sigma \in \gamma_n$ einen akzeptierenden Weg p_σ für die Eingabe A^σ.
Wir ordnen jedem $\sigma \in \gamma_n$ die Knoten v_σ auf p_σ zu.
Ist (v, v') $\frac{n}{2}$–te Kante auf p_σ mit Label 1, dann setze $v_\sigma := v'$.
Bezeichne das Anfangsstück von p_σ, das in v_σ endet als $p_\sigma^{(1)}$ und das Reststück als $p_\sigma^{(2)}$.
Sei $v := \{v_\sigma | \sigma \in \mathcal{S}_n\}$.

Behauptung.
$\sharp v \geq \frac{2^n}{2\sqrt{n}}$.
Beweis.
Sei $v \in V$ und $\mathcal{S}_n = \{\sigma \in \gamma_n | v_\sigma = v\}$.
Wir erreichen durch Umbenennung, daß $id \in \mathcal{S}_v$ und $x_i i$ auf $p_{id}^{(1)}$ getestet wird, genau dann, wenn $i \leq \frac{n}{2}$.
Annahme: Sei $\sigma \in \mathcal{S}_v$, $\sigma \neq id$.
$p_{id}^{(1)}$, $p_\sigma^{(2)}$ und $p_\sigma^{(1)}$, $p_{id}^{(2)}$ seien neben p_{id} und p_σ akzeptierende Wege.
Wird x_{ij} auf $p_\sigma^{(1)}$ (oder auf $p_\sigma^{(2)}$) positiv getestet und ist $i \leq \frac{n}{2}, j > \frac{n}{2}$, dann ist $p_\sigma^{(1)}$, $p_{id}^{(2)}$ (oder $p_{id}^{(1)}$, $p_\sigma^{(2)}$) akzeptierend und x_{ij} und x_{jj} werden positiv getestet.
<u>Weil</u> P nur Permutationsmatrizen akzeptiert, ist $\sharp\mathcal{S}_v \leq ((\frac{n}{2})!)^2$.
Daraus folgt

$$\sharp V \geq \frac{\sharp\mathcal{S}_n}{\max\limits_{v \in V} \sharp\mathcal{S}_v} \geq \frac{n!}{(\frac{n}{2})!)^2} - \binom{n}{\frac{n}{2}} \geq \frac{2^n}{2\sqrt{n}}$$

wegen der Stirling Formel. Wegen $SIZE(P) \geq \sharp V$ folgt die Behauptun. ■

5.8 Branching–Programm beschränkter Weite

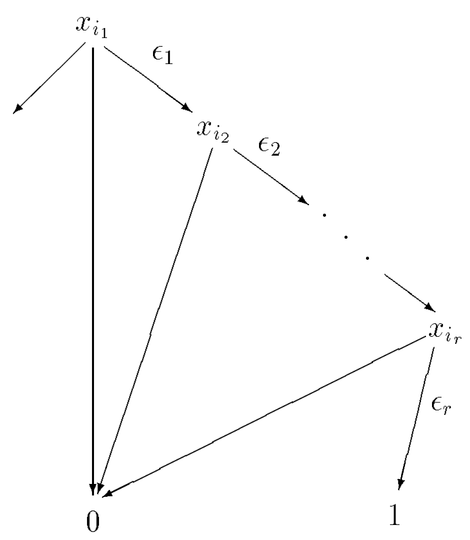
Definition 5.5.
Sei P ein *synchrones* Branching–Programm, d.h. alle Wege von der Quelle zu einem Knoten $v \in P$ sind von gleicher Länge.
Sei $l(v)$ = Länge eines solchen Weges, Länge $P = \max\{l(v)|v \in P\}$. Dann hat man folgende Definition

$$width(P) := \max_{1 \leq k \leq \text{Länge P}} \sharp\{v|l(v) = k\}.$$

Satz 5.9.
Jede Boolesche Funktion $f \in \mathbb{B}_n$ läßt sich von einem Branching–Programm der Weite 2 berechnen.

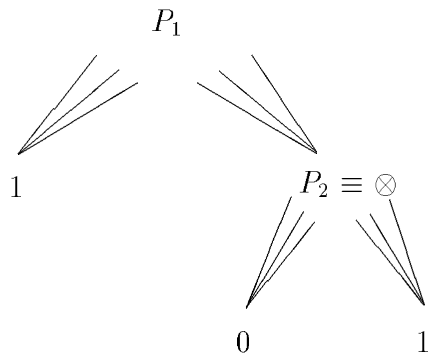
Beweis.
Sei $f \in \mathbb{B}_n$. Dann ist $f(x) = \sum_j m_j(x)$ die DNF.

1. Wir können jedes Monom $x_{i_1}^{\varepsilon_1}, \ldots, x_{i_r}^{\varepsilon_r}$ durch Branching–Programm der Weite 2 berechnen (siehe nachfolgendes Bild plus Einfügen von Dummy-Knoten):

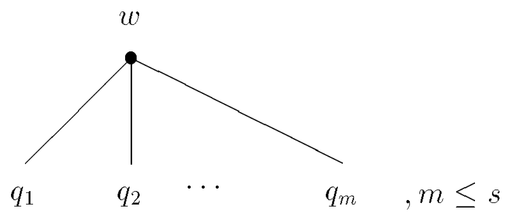


2. Wir können die Disjunktion von Funktionen, die durch ein Branching–Programm der Weite k berechnet werden können, durch ein Branching–Programm der Weite k berechnen:

$f_1 \vee f_2$:



Analyse der Simulation:
Sei $f \in \mathbb{B}_n$.
f werde von einem $\mathcal{U}$–Schaltkreis S mit s Gattern in k Stufen berechnet.
Dann hat ein Branching–Programm P für f der Weite 2 die Größe $\leq 2 \cdot s^{k-1}n$.
Beweis.
Induktion über k:
$\underline{k = 1:}$ klar
$\underline{k-1 \to k:}$ S:



g_i haben $\leq k-1$ Stufen.

Somit erhalten wir ein Branching–Programm P_i der Weite 2 für g_i mit $\leq 2 \cdot s^{k-2}n$.
Als erhalten wir ein Branching–Programm P der Weite 2 für $w(g_1, \ldots, g_n)$ der Größe $\leq 2 \cdot s \cdot s^{k-2}n = 2 \cdot s^{k-1} \cdot n$. ■

$$\mathcal{P}_{width\ k-Branching-Programm} = \{A | A \text{ wird von Branching-Programm polynomialer Größe und Weite } k \text{ akzeptiert}\}$$
$$\mathcal{P}_{bw-Branching-Programm} = \bigcup_{k \geq 1} \mathcal{P}_{widthk-Branching-Programm}$$

Korollar 5.2.
$AC_0 \subseteq \mathcal{P}_{width\ 2-Branching-Programm}$.

Satz 5.10. *(Barington 86)*
$\mathcal{NC}^1 = \mathcal{P}_{bwBranching-Programm}$.