Universität Trier

Fachbereich IV - Informatik

Christoph Meinel:

Logikminimierung

Vorlesungsskript WS 1994/95

Inhaltsverzeichnis

Zusammenfassung

Zu den verschiedenen Unterbereichen der praktischen Informatik gehört die *Logikoptimierung*, deren wichtigster Bestandteil wiederum die *Logikminimierung* ist. „Logik“ ist hierbei nicht im üblichen Sinne, sondern als „Schaltlogik“ oder ähnliches zu verstehen; Logikminimierung hat also zum Ziel, aus einer gegebenen Anforderung an eine zu konstruierende Schaltung den Aufbau dieser Schaltung (unter Verwendung vorgegebener Teile) so zu bestimmen, daß zum Bau dieser Schaltung möglichst wenig Aufwand erforderlich ist. Dieser Prozeß wird meist in mehrere Zwischenstufen unterteilt, die von der rein mathematischen Ebene zu immer schaltungsnäheren Bereichen fortschreiten, weil eine umfassende Berücksichtigung mathematischer Vereinfachungsschritte einerseits und praktischer Probleme (wie etwa dem Layout der Verbindungsleitungen) andererseits derzeit nicht möglich ist.
Im eher theoretisch–mathematischen Bereich der Logikminimierung arbeitet man mit zu realisierenden *Booleschen Funktionen* derart, daß diese Funktionen in einer bestimmten *Darstellung* vorliegen und die Darstellung so umgeformt wird, daß ihre *Größe* möglichst klein wird. Dabei ist die Größe einer Darstellung „zweckdienlich“ definiert, d. h. sie gibt einerseits einen einleuchtenden Begriff der „Handlichkeit“ einer Darstellung und andererseits eine grobe Abschätzung des Schaltungsaufwandes für eine „entsprechende“ Schaltung. Betrachtet man nicht Funktionen, sondern Funktionenfolgen, kann die Folge der Größen optimaler Darstellungen zur Definition eines Komplexitätsbegriffes dienen.
Die Vorlesung „Logikminimierung“ von Herrn Professor Meinel führt zunächst in die Theorie der Booleschen Funktionen ein, behandelt dann Repräsentationen Boolescher Funktionen und ihre Komplexität und leitet anschließend zur *zweistufigen Logikminimierung* über. Das ist die Logikminimierung unter Einschränkung auf solche Darstellungen, die höchstens zwei „Verarbeitungsschritte“ (logische Gatter) auf die Eingaben anwenden. Da für diesen Teilbereich geschlossene Theorien existieren, gibt es in der Praxis Programmpakete zur automatischen zweistufigen Logikminimierung. Eines davon, ESPRESSO II, wird zum Abschluß der Vorlesung in seinem Aufbau skizziert.

Dieses Skript liegt in Form von LaTeX–Quelltexten vor, erstellt von Detlef Menne an der Gesamthochschule–Universität Paderborn, 1992 an der Universität Trier überarbeitet von Jochen Bern, und 1995 von Stefan Krischer und Gerd Hoff. Bei der Fachschaft liegt eine Kopiervorlage bereit. Wenden Sie sich bei Rückfragen bitte an Herrn Professor Meinel, E-Mail-Adresse MEINEL@uni-trier.de.

Kapitel 1

Einführung in die Theorie der Booleschen 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:*

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

und

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

1.1 Boolesche Funktionen

Definition 1.1
$\mathbb{B}_{n,m}$ bezeichnet die Menge der Booleschen Funktionen

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

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

Satz 1.1
Es gilt $\# I\!B_{n,m} = 2^{m2^n}$.

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: $\# I\!B_{n,m} = 2^{m2^n}$. ■

Beispiel 1.1

$\# I\!B_2 = 16 \quad (= 2^4)$
$\# I\!B_3 = 256 \quad (= 2^8)$
$\# I\!B_4 = 65536 \quad (= 2^{16})$
$\# I\!B_5 > 4 * 10^9$
$\# I\!B_6 > 16 * 10^{18}$.

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

1.2 Wichtige Funktionen aus $I\!B_0 \cup I\!B_1 \cup I\!B_2$

- *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 1.2 (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) (de Morgan–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

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

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

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

1.3 Wichtige Operationen Boolescher Funktionen

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 1.2
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) $x + x * y = x$; $x * (x + y) = x$ *(Absorption)*

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

Satz 1.3
$(\mathbb{B}_n, \vee, \wedge, \neg)$ *bildet eine Boolesche Algebra.* ■

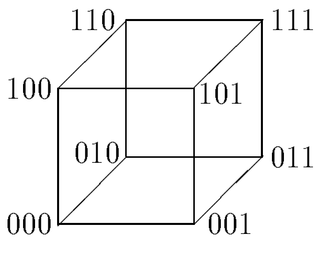
Satz 1.4 (Dualitätsprinzip)
Jede Aussage in einer Booleschen Algebra bleibt wahr, wenn man die Konstanten 0 und 1 und die Operatoren $\wedge$ und $\vee$ vertauscht.

Beweis
Überprüfung aller Axiome. ■

1.4 Geometrische Veranschaulichung Boolescher Funktionen

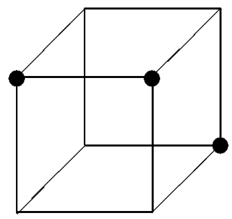
Wir fassen die Argumente $(x_1, \ldots, x_n) \in \{0, 1\}^n$ als Ecken eines *n–dimensionalen Würfels* auf.

Beispiel 1.2



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

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



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

1.5 Partielle Boolesche Funktionen

Definition 1.3
$\widetilde{\mathbb{B}}_{n,m}$ bezeichnet die Menge der *partiellen Booleschen Funktionen*

$$f : \{0,1\}^n \overset{\supset}{\rightarrow} \{0,1\}^m.$$

$dc(f) := \{0,1\}^n \backslash def(f)$ heißt *don't care Menge* von f.

Wir können partielle Boolesche Funktionen $f \in \widetilde{\mathbb{B}}_n$ beschreiben mit Hilfe von drei (totalen) Booleschen Funktionen f_{on}, f_{off}, f_{dc} mit

$$\begin{aligned} f_{on}(x_1, \ldots, x_n) = 1 \quad & \text{falls} \quad (x_1, \ldots, x_n) \in on(f), \\ f_{off}(x_1, \ldots, x_n) = 1 \quad & \text{falls} \quad (x_1, \ldots, x_n) \in off(f) \text{ und} \\ f_{dc}(x_1, \ldots, x_n) = 1 \quad & \text{falls} \quad (x_1, \ldots, x_n) \in dc(f), \end{aligned}$$

den charakteristischen Funktionen der *on*–, *off*– und *dc*–Mengen.

Bezeichnung
Sei $S \subseteq \{0,1\}^n$.

$$\mathbb{B}_{n,m}(S) := \{f : S \longrightarrow \{0,1\}^m\}.$$

Offenbar gilt $\mathbb{B}_{n,m}(S) \subseteq \widetilde{\mathbb{B}}_{n,m}$ für alle $S \subseteq \{0,1\}^n$.

Satz 1.5
Sei $S \subseteq \{0,1\}^n$, dann ist $(\mathbb{B}_n(S), \vee, \wedge, \neg)$ eine Boolesche Algebra. ■

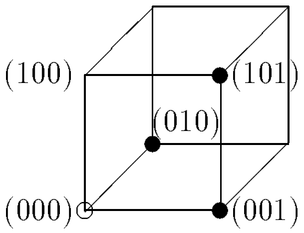
Definition 1.4
Sei $f \in \widetilde{\mathbb{B}}_n$. $F \in \mathbb{B}_n$ heißt *Cover* von f, falls

$$on(f) \subseteq on(F) \subseteq on(f) \cup dc(f).$$

Es gilt $on(f) \subseteq on(F)$ und $off(f) \cap on(F) = \emptyset$.

Beispiel 1.4
Sei f gegeben durch

x	y	z	$f(x,y,z)$
0	0	0	0
0	0	1	1
1	0	1	1
0	1	0	1



Dann ist

$$dc(f) = \{(011), (100), (110), (111)\}.$$

Seien $F(x,y,z) := y \oplus z$ und $F'(x,y,z) := x\overline{y}z + \overline{x}\,\overline{y}\,z + \overline{x}y\overline{z}$.
F und F' sind Cover von f.

1.6 Monotone und symmetrische Boolesche Funktionen

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

Definition 1.5
$f \in \mathbb{B}_n$ heißt *monoton*, falls $(x_1, \ldots, x_n) \leq (y_1, \ldots, y_n)$ stets $f(x_1, \ldots, x_n) \leq f(y_1, \ldots, y_n)$ 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 1.6
$f \in \mathbb{B}_n$ heißt *symmetrisch*, falls für beliebige Permutationen π von $\{1, \ldots, n\}$ gilt:

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

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

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

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: $\#\mathbb{S}_n = 2^{n+1}$. ■

Definition 1.7
Eine symmetrische Funktion 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.: $f = I_{i,j}$.

Bemerkung
Intervallfunktionen bilden eine Basis der symmetrischen Funktionen, da jede symmetrische Funktion eindeutig mit ihnen dargestellt werden kann.

Beispiel 1.5
Sei $f \in \mathbb{S}_9, v(f) = (1001001110)$.
Dann gilt:

$$f = I_{0,0} \vee I_{3,3} \vee I_{6,8}.$$

Definition 1.8
Schwellwertfunktionen (Thresholdfunktionen) $f \in \mathbb{B}_n$ sind Intervallfunktionen $f = I_{k,n}$.
Bez.: $f = t_k^n \in \mathbb{B}_n$.

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

Kapitel 2

Repräsentation Boolescher Funktionen

Es gibt eine Reihe verschiedener Möglichkeiten, Boolesche Funktionen zu repräsentieren. Ist die Darstellung in einem vorgegebenen Rahmen eindeutig, dann heißt sie *kanonisch*.

2.1 Algorithmische Form

Wir können Boolesche Funktionen repräsentieren durch Angabe von die Funktionen spezifizierenden Algorithmen.

Beispiel 2.1

$$
\begin{array}{lcl}
f(x_1, \ldots, x_n) = 1 & \Longleftrightarrow & \sum\limits_{i=1}^{n} x_i \equiv 0 \; mod\, 3 \\
ADD_n(x_n, \ldots, x_0, y_n, \ldots, y_0) & = & \text{Binärdarstellung der Summe der beiden durch} \\
 & & (x_n, \ldots, x_0), (y_n, \ldots, y_0) \text{ binär repräsentierten} \\
 & & \text{Zahlen } x \text{ und } y \\
SUB_n(x_n, \ldots, x_0, y_n, \ldots, y_0) & = & \text{Binärdarstellung der Differenz von } x \text{ und } y \\
MULT_n(x_n, \ldots, x_0, y_n, \ldots, y_0) & = & (p_{2n}, \ldots, p_0) \text{ Binärdarstellung des Produktes} \\
 & & \text{von } x \text{ und } y
\end{array}
$$

Algorithmus für $MULT_n$:

1. Berechne $z_i = (x_n \wedge y_i, \ldots, x_0 \wedge y_i)$ für $0 \leq i \leq n$
2. „verschiebe“ z_i um i Positionen nach links

3. „Addiere“ die so erhaltenen $\tilde{z}_i$.

2.2 Tabellendarstellung

Vollständige Aufzählung aller relevanten Argumente in Form einer Tabelle und Angabe der entsprechenden Funktionswerte.

Beispiel 2.2
Sei $f \in \widetilde{\mathbb{B}}_{3,2}$,
$dc(f) = \{(100), (011), (101)\}$.

f :

x	y	z	$f_1(x, y, z)$	$f_2(x, y, z)$
0	0	0	1	0
0	0	1	0	0
0	1	0	1	1
0	1	1	-	-
1	0	0	-	-
1	0	1	-	-
1	1	0	0	1
1	1	1	1	0

Die Größe der Tabelle für $f \in \widetilde{\mathbb{B}}_{n,m}$ ist proportional zu $\#\mathit{def}(f)$, d.h. im allgemeinen exponentiell in n.
Für $f \in \mathbb{B}_n$ ist die Größe der Tabelle $= \mathrm{O}(2^n)$.

Ist $f \in \mathbb{B}_n$, so können wir f eindeutig beschreiben durch Angabe von $on(f) = f^{-1}(1)$ oder $\mathit{off}(f) = f^{-1}(0)$.

Ist $f \in \mathbb{B}_{n,m}$, dann ist f repräsentierbar durch m verschiedene *on*- bzw. *off*-Mengen:

$$on(f_1), \ldots, on(f_m) \text{ oder } \mathit{off}(f_1), \ldots, \mathit{off}(f_m),$$

wobei $f(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))$.

Ist $f \in \widetilde{\mathbb{B}}_{n,m}$, dann ist f repräsentierbar durch $2m$ verschiedene *on*-, *off*- bzw. *dc*–Mengen, z. B.

$$\begin{array}{ll} on(f_1), \ldots, on(f_m), & dc(f_1), \ldots, dc(f_m); \\ on(f_1), \ldots, on(f_m), & \mathit{off}(f_1), \ldots, \mathit{off}(f_m); \\ dc(f_1), \ldots, dc(f_m), & \mathit{off}(f_1), \ldots, \mathit{off}(f_m). \end{array}$$

2.3 Zweistufige Repräsentationen

Definition 2.1
Sei $\omega \in \mathbb{B}_2$ assoziativ. Ein ω–*Monom* ist ein ω–Produkt von Literalen. Die *Länge* eines Monoms ist die Zahl der enthaltenen Literale. Per Definition sei die 1 das leere $\wedge$–Monom und die 0 das leere $\vee$–Monom.

Bezeichnung
$\wedge$–Monome heißen kurz *Monome* (Würfel, Produkt, ...).

Wir fassen manchmal Monome auch als Menge der in ihnen enthaltenen Literale auf (wenn die Operation klar ist). Dann können wir schreiben: $\overline{a} \in \{\overline{a}, b, \overline{c}\}$.

Monome sind besonders einfach strukturierte Boolesche Funktionen. Sie sind handliche Bausteine, aus denen verschiedene Repräsentationen Boolescher Funktionen aufgebaut werden können.

Definition 2.2
Seien $\omega, \omega' \in \mathbb{B}_2$ und sei (ω, ω') distributiv.
Ein (ω, ω')–*Polynom* ist ein ω'–Produkt von ω–Monomen.
Die *Länge* eines (ω, ω')–Polynoms ist die Summe der Längen der enthaltenen Monome.
$(\wedge, \vee)$–Polynome heißen auch *disjunktive Normalform* (DNF).
$(\vee, \wedge)$–Polynome heißen auch *konjunktive Normalform* (KNF).
$(\wedge, \oplus)$–Polynome heißen auch *Parity-Normalform* (PNF).
RSE–Darstellungen sind spezielle PNF, ohne negative Literale.

Definition 2.3
Sei $f \in \mathbb{B}_n$.
Ein kürzestes (ω, ω')–Polynom für f heißt (ω, ω')–*Minimalpolynom.*
$S_{(\omega,\omega')}(f)$ sei die Länge eines (ω, ω')–Minimalpolynoms von f.
Insbesondere ist

$$DNF(f) := S_{(\wedge,\vee)}(f)$$

$$KNF(f) := S_{(\vee,\wedge)}(f)$$

$$PNF(f) := S_{(\wedge,\oplus)}(f)$$

Eine (ω, ω')–Polynomdarstellung einer Funktion $f \in \mathbb{B}_{n,m}$ mit mehreren Ausgängen ergibt sich aus den Polynomdarstellungen der einzelnen Komponenten $f_i, f = (f_1, \ldots, f_n)$. Wieder bezeichne $S_{(\omega,\omega')}(f)$ die Länge (für $m > 1$ als Summe der $S_{(\omega,\omega')}(f_i)$ definiert) einer kleinsten (ω, ω')–Polynomdarstellung von f.

2.3.1 Kanonische disjunktive und konjunktive Normalform

Definition 2.4
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 ($\wedge$–)Monom

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

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

$$s_a(x) := x_1^{\overline{a}_1} \vee \ldots \vee x_n^{\overline{a}_n}$$

bezeichnet.

Lemma 2.1
Sei $a = (a_1, \ldots, a_n) \in \{0,1\}^n$. Es gilt:
(i) $m_a(x) = 1$ genau dann, wenn für alle $1 \leq i \leq n$ gilt: $x_i = a_i$.
(ii) $s_a(x) = 0$ genau dann, wenn für alle $1 \leq i \leq n$ gilt: $x_i = a_i$.

Beweis

$$\begin{array}{rcl}
(i)\;\; m_a(x) = 1 & \Longleftrightarrow & \text{für alle } i \text{ gilt: } x_i^{a_1} = 1 \\
 & \Longleftrightarrow & \text{für alle } i \text{ gilt: } x_i = a_i. \\
(ii)\;\; s_a(x) = 0 & \Longleftrightarrow & \text{für alle } i \text{ gilt: } x_i^{\overline{a}_i} = 0 \\
 & \Longleftrightarrow & \text{für alle } i \text{ gilt: } x_i^{a_i} = 1 \\
 & \Longleftrightarrow & \text{für alle } i \text{ gilt: } x_i = a_i. \quad \blacksquare
\end{array}$$

Satz 2.1
Sei $f \in \mathbb{B}_n$. Dann gilt:

$$\begin{array}{lcl}
(i)\;\; f(x) & = & \bigvee\limits_{a \in f^{-1}(1)} m_a(x), \\
(ii)\; f(x) & = & \bigwedge\limits_{a \in f^{-1}(0)} s_a(x).
\end{array}$$

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

Beweis
Übungsaufgabe. $\blacksquare$

Definition 2.5
Die Darstellung (i) heißt *kanonische disjunktive Normalform* (kDNF) von f.
Die Darstellung (ii) heißt *kanonische konjunktive Normalform* (kKNF) von f.

Korollar 2.1
Die Länge einer kDNF (kKNF) ergibt sich als $n \cdot \#on(f)$ $(n \cdot \#off(f))$. ■

Beispiel 2.3

$$1.)\quad f(x_1,\ldots,x_4) = x_1\overline{x}_2x_3x_4 + \overline{x}_1x_2x_3\overline{x}_4 + \overline{x}_1x_2\overline{x}_3x_4$$

Länge : $3 \cdot 4 = 12$

$$2.)\quad f(x_1,\ldots,x_5) = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5$$

$$\left(= \sum_{i=1}^{5} x_i \ \ mod\ 2\right) \qquad \textit{Parityfunktion}$$

$kDNF$:

$$\begin{aligned} f(x_1,\ldots,x_5) &= x_1\overline{x}_2\overline{x}_3\overline{x}_4\overline{x}_5 + \overline{x}_1x_2\overline{x}_3\overline{x}_4\overline{x}_5 + \cdots + \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4x_5 \\ &\quad + x_1x_2x_3\overline{x}_4\overline{x}_5 + \cdots + \overline{x}_1\overline{x}_2x_3x_4x_5 \\ &\quad + x_1x_2x_3x_4x_5 \end{aligned}$$

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 2.4

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 2.2 (Shannon'scher Entwicklungssatz)
Sei $f \in \mathbb{B}_n$. *Für alle* $1 \le i \le n$ *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 2.5

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

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

Wegen $(f \wedge g)^{-1}(1) = f^{-1}(1) \cap g^{-1}(1)$ und $(f \vee g)^{-1}(1) = f^{-1}(1) \cup g^{-1}(1)$ (bzw. $(f \wedge g)^{-1}(0) = f^{-1}(0) \cup g^{-1}(0)$ und $(f \vee g)^{-1}(0) = f^{-1}(0) \cap g^{-1}(0)$) kann eine kDNF (kKNF) für $f \wedge g$ bzw. $f \vee g$ leicht aus den kDNF (kKNF) für f und g gewonnen werden.

kDNF (kKNF) sind zwar eindeutig bestimmt, aber oft existieren kürzere Darstellungen als $(\wedge, \vee)$–Polynom ($(\vee, \wedge)$–Polynom).

Beispiel 2.6

$f(x_1, x_2, x_3) = x_1 \vee x_2$

kDNF: $f(x_1, x_2, x_3) = x_1 x_2 x_3 + x_1 x_2 \overline{x}_3 + x_1 \overline{x}_2 x_3 + x_1 \overline{x}_2 \overline{x}_3 + \overline{x}_1 x_2 x_3 + \overline{x}_1 x_2 \overline{x}_3$.

Man kann kürzere $(\wedge, \vee)$–Polynomdarstellungen ($(\vee, \wedge)$–Polynomdarstellungen) durch Anwendung der Rechenregeln aus Satz 1.2 gewinnen.

Beispiel 2.7

$$\begin{aligned}
f(x_1, x_2, x_3) &= x_1 x_2 x_3 + x_1 x_2 \overline{x}_3 + x_1 \overline{x}_2 x_3 \\
&\quad + x_1 \overline{x}_2 \overline{x}_3 + \overline{x}_1 x_2 x_3 + \overline{x}_1 x_2 \overline{x}_3
\end{aligned}$$

$$
\begin{aligned}
&= x_1x_2(x_3+\overline{x}_3)+x_1\overline{x}_2(x_3+\overline{x}_3)+\overline{x}_1x_2(x_3+\overline{x}_3)\\
&= x_1x_2+x_1\overline{x}_2+\overline{x}_1x_2 \qquad (\text{da } x_3+\overline{x}_3=1)\\
&= x_1\overline{x}_2+x_1x_2+x_1x_2+\overline{x}_1x_2\\
&= x_1(x_2+\overline{x}_2)+(x_1+\overline{x}_1)x_2\\
&= x_1+x_2 \qquad (\text{da } x_i+\overline{x}_i=1)
\end{aligned}
$$

DNF besitzen außerordentlich große Bedeutung, da sie unmittelbar zu PLA–Realisierungen (siehe Abschnitt 2.6) führen.

Fehlende algebraische Struktur erschwert die Arbeit mit DNF und KNF, z. B. sind $f \vee g = h$, $f \wedge g = h$ nicht eindeutig nach f auflösbar.

Dies ist anders bei der Arbeit mit $(\wedge, \oplus)$–Polynomen.

2.3.2 Ring–Summen–Entwicklung (RSE)

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

Definition 2.6
Eine Parity-Normalform ($(\wedge, \oplus)$–Polynom) heißt *Ring–Summen–Entwicklung* (RSE), falls alle Monome nur aus unnegierten Literalen bestehen.

Satz 2.3
Jede Boolesche Funktion $f \in \mathbb{B}_n$ besitzt eine eindeutig bestimmte Darstellung als RSE, d.h. die RSE ist eine kanonische Darstellung.

Beweis
<u>Existenz:</u>
Wir gehen von der kDNF für f aus:

$$f(x_1,\ldots,x_n) = \bigvee_{a\in f^{-1}(1)} m_a(x)$$

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:

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

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

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

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

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

Beispiel 2.8

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

Man kann eine RSE als (gewöhnliches) Polynom aus $\mathbb{Z}_2[x_1, \ldots, x_n]$ auffassen und andererseits eine RSE von f eindeutig durch den Vektor $v = v(f) = (v_I)_{I \subseteq \{1, \ldots, n\}}$,

$$v_I = \begin{cases} 1 & m_I \text{ ist Summand der RSE} \\ 0 & \text{sonst} \end{cases},$$

repräsentieren:

$$f(x_1, \ldots, x_n) = \bigoplus_{I \subseteq \{1, \ldots, n\}} v_I \cdot m_I \left(= \bigoplus_{\substack{I \subseteq \{1, \ldots, n\} \\ v_I = 1}} m_I \right).$$

2.4 Mehrstufige Repräsentationen

2.4.1 Boolesche Ω–Ausdrücke

Definition 2.7
Seien $X_n := \{x_1, \ldots, x_n\}$ Menge von n Booleschen Variablen,

$\overline{X}_n := \{\overline{x}_1, \ldots, \overline{x}_n\}$,
$\Omega \subseteq \bigcup_{k \geq 2} \mathbb{B}_k$ endlich.
Seien $A_0(X_n) := \{0, 1\} \cup X_n \cup \overline{X}_n$,

$$A_{i+1}(X_n) := A_i(X_n) \cup \{\omega(a_1, \ldots, a_k) \mid \omega \in \Omega \cap \mathbb{B}_k,\ a_j \in A_i(X_n)\}$$

Die Menge

$$A(X_n) := \bigcup_{i \geq 0} A_i(X_n)$$

heißt Menge der *Booleschen Ω–Ausdrücke.*

Gilt $\Omega = \{\wedge, \vee\}$, so heißt $A(X_n)$ die Menge der Booleschen Ausdrücke.
Die *Größe* eines Booleschen Ω–Ausdrucks ist die Zahl der in ihm vorkommenden Literale, Konstanten und Operationssymbole aus Ω.
Die *Tiefe* eines Booleschen Ω–Ausdruckes a ist das minimale t mit $a \in A_t(X_n)$ (d.h. $a \notin A_{t-1}(X_n)$).

Beispiel 2.9

$\wedge(\vee(\wedge(x_1, x_2), x_3), \vee(x_4, x_5)) = (x_1x_2 + x_3)(x_4 + x_5)$,

$\oplus(\vee(x_1, x_2), \wedge(\vee(x_3, x_4), x_5)) = (x_1 + x_2) \oplus (x_3 + x_4)x_5$,

...

Offenbar gilt :

$$\begin{aligned}(i)\ \mathit{Größe}(\omega(a_1, \ldots, a_k)) &= \sum_{i=1}^{k} \mathit{Größe}(a_i) + 1,\\ (ii)\ \mathit{Tiefe}(\omega(a_1, \ldots, a_k)) &= \max(\mathit{Tiefe}(a_i)) + 1.\end{aligned}$$

Boolesche Ω–Ausdrücke a repräsentieren Boolesche Funktionen f_a vermöge der folgenden Interpretation:

$$f_0 := 0, \quad f_1 := 1, \quad f_{x_i} := x_i, \quad f_{\overline{x}_i} := \overline{x}_i; \quad f_{\omega(a_1,\ldots,a_k)} := \omega(f_{a_1}, \ldots, f_{a_k}).$$

Verschiedene Boolesche Ω–Ausdrücke können die gleichen Funktionen repräsentieren, d.h. die Interpretation ist nicht injektiv!

Beispiel 2.10

$$\begin{array}{ccc} f_{(x_1x_2+x_3)(x_4+x_2)} & = & f_{(x_2(x_1(x_4+1)+x_3)+x_3x_4)} \\ \| & & \| \\ f_{(x_1x_2x_4+x_1x_2+x_3x_4+x_3x_2)} & = & f_{(x_2x_1x_4+x_2x_1+x_2x_3+x_3x_4)} \end{array}$$

Definition 2.8
Zwei Boolesche Ausdrücke a, a' heißen *äquivalent* genau dann, wenn gilt: $f_a = f_{a'}$.

Offenbar ist DNF (analog kDNF, KNF, kKNF, PNF, RSE) ein Boolescher Ausdruck der Tiefe 2.

Achtung: Die Größe einer DNF als $(\wedge, \vee)$–Polynom betrachtet ist anders definiert als die Größe einer DNF als Boolescher Ausdruck.

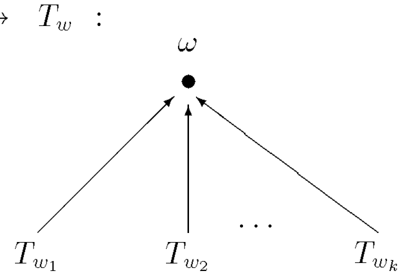
Beispiel 2.11

$f(x_1, \ldots, x_6) = x_1 x_2 + x_3 \overline{x}_5 \overline{x}_6$

Größe als $(\wedge, \vee)$–Polynom: 5.
Größe als $\{\wedge, \vee\}$–Ausdruck: 8. (Sogar 9, wenn $\wedge$ strikt als binärer Operator aufgefaßt wird!)

Man kann Boolesche Ausdrücke a graphisch darstellen als Bäume T_a:

$$\begin{array}{rcll}
0 & \to & T_0 & : \quad \bullet 0 \\
1 & \to & T_1 & : \quad \bullet 1 \\
x_i & \to & T_{x_i} & : \quad \bullet x_i \\
\overline{x}_i & \to & T_{\overline{x}_i} & : \quad \bullet \overline{x}_i \\
w = \omega(w_1, \ldots, w_k) & \to & T_w & :
\end{array}$$

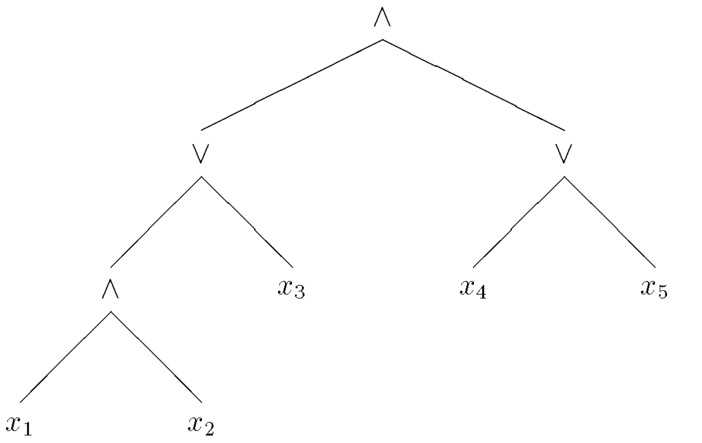


Damit entsprechen Boolesche Ω–Ausdrücke Bäumen, deren Knoten mit Elementen aus $\{0,1\} \cup X_n \cup \overline{X}_n \cup \Omega$ wie folgt markiert sind:

- Blätter des Baumes haben Markierungen in $\{0,1\} \cup X_n \cup \overline{X}_n$;
- Innere Knoten tragen Markierungen $\omega \in \Omega$ und
- haben so viele Söhne, wie die Stelligkeit von ω angibt.

Solche Bäume heißen auch Ω–*Formeln*. Die geforderte Endlichkeit von Ω impliziert, daß Ω–Formeln beschränkten Fan–in besitzen.

Beispiel 2.12
$(x_1x_2 + x_3)(x_4 + x_2)$ als Ω–Formel:



Definition 2.9
Sei $f \in \mathbb{B}_n$.
$L_\Omega(f) = \min\{\textit{Größe einer } \Omega\textit{–Formel für } f\}$,
$D_\Omega(f) = \min\{\textit{Tiefe einer } \Omega\textit{–Formel für } f\}$.

Welche Bedeutung hat die Basis Ω?

Satz 2.4
Seien Ω, Ω' vollständige, endliche Basen, dann existieren Konstanten c und d, so daß für alle $f \in \mathbb{B}_n$ gilt: $L_\Omega(f) \le cL_{\Omega'}(f)$ und $D_\Omega(f) \le dD_{\Omega'}(f)$.

Beweis
Übungsaufgabe. ∎ Jede Boolesche Ω–Formel, $\Omega \subseteq \mathbb{B}_n$, kann vollständig balanciert entworfen werden:

Satz 2.5
Seien $f \in \mathbb{B}_n$ und $\Omega \subseteq \mathbb{B}_2$ vollständig. Dann existieren Konstanten $k(\Omega)$ mit

$$\log(L_\Omega(f) + 1) \le D_\Omega(f) \le k(\Omega)\log(L_\Omega(f) + 1)$$

und folglich gilt

$$D_\Omega(f) = \Theta(\log(L_\Omega(f))). \quad \blacksquare$$

Die Tiefe $D_\Omega(f)$ einer Formel gibt den minimalen Zeitbedarf für die Berechnung von f durch einen beliebigen Schaltkreis mit Ω–Gattern und beschränktem Fan–in an.

Man erhält Formeln für Funktionen $f \in I\!B_{n,m}$ entweder durch Auszeichnung verschiedener Knoten als Ausgabeknoten oder die Bereitstellung einer gesonderten Formel für jede Komponente.

2.4.2 Boolesche Ω–Schaltkreise

Definition 2.10
Sei $\Omega \subseteq \bigcup\limits_{k\geq 0} I\!B_k$ endlich.
Ein Ω–*Schaltkreis* S ist ein Tupel $S = (n, m, G, l, o)$ mit:

- n ist die Zahl der Eingabevariablen,
- m ist die Zahl der Ausgabevariablen,
- $G = (V, E)$ ist ein gerichteter azyklischer Graph,
- $l : V \to \Omega \cup X_n \cup \overline{X}_n \cup \{0, 1\}$ ist eine Zuordnung, die jedem Knoten $v \in V$ eine Funktion (*Gatter*) $l(v) \in \Omega \cup X_n \cup \overline{X}_n \cup \{0, 1\}$ zuordnet, wobei der Ingrad von v übereinstimmt mit der Stelligkeit von $l(v)$,
- $o : \{1, \ldots, m\} \to V$ bezeichnet die Ausgabeknoten.

Definition 2.11
Die *Größe* von S ist die Anzahl der Knoten von V.
Die *Tiefe* von S ist die Länge eines längsten Weges von einer Quelle (Eingabeknoten) zu einer Senke (Ausgabeknoten).
Gilt $\Omega = I\!B_2$, so sprechen wir einfach von Schaltkreisen (SK).

Ω–Schaltkreise S *repräsentieren (berechnen)* Boolesche Funktionen:
Sei $S = (n, m, G, l, o)$, $G = (V, E)$, ein Ω–SK. Wir ordnen induktiv den Knoten $v \in V$ Funktionen $res_{S,v} \in I\!B_n$ zu:

Sei $v \in V$ Quelle, d.h. Ingrad(v)=0 und damit $l(v) \in X_n \cup \overline{X}_n \cup \{0, 1\}$.

<u>1. Fall:</u> $l(v) = 0$ (1).
Setze $res_{S,v}(x) := 0$ (1) für alle $x \in \{0, 1\}^n$.

<u>2. Fall:</u> $l(v) = x_i$ ($\overline{x}_i$).
Setze $res_{S,v}(x) := x_i$ ($\overline{x}_i$).

Sei $v \in V$ und sei allen Vorgängern $v_1, \ldots, v_k$ von v bereits eine Funktion $res_{S,v}$ zugeordnet. (Da G azyklisch ist, muß unter den noch nicht behandelten Knoten ein solcher Knoten existieren.)

Gilt Ingrad(v)=k, dann ist $l(v) = \omega \in \Omega \cap I\!B_k$.
Setze

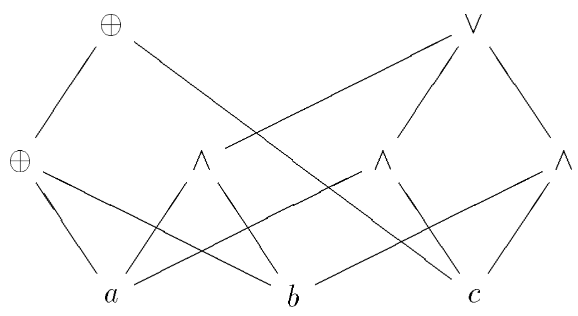
$$res_{S,v}(x) := \omega(res_{S,v_1}(x), \ldots, res_{S,v_k}(x)).$$

Wir definieren nun $res_S \in I\!B_{n,m}$ wie folgt:

$$res_S(x) := (res_{S,o(1)}(x), \ldots, res_{S,o(m)}(x)).$$

Beispiel 2.13
Volladdierer $FA \in I\!B_{3,2}$ als Ω–Schaltkreis:

$$\begin{aligned} FA(a,b,c) &= (y_1, y_0) \text{ mit } a+b+c = 2y_1 + y_0, \text{ z. B.} \\ y_0 &= a \oplus b \oplus c \\ y_1 &= ab \vee ac \vee bc \end{aligned}$$



Ω–Schaltkreisrepräsentationen sind nicht eindeutig! Verschiedene Ω–Schaltkreise repräsentieren die gleiche Funktion.

Definition 2.12
Sei $f \in I\!B_{n,m}$.
$C_\Omega(f) = \min\{$Größe eines Ω–Schaltkreises S mit $res_S = f\}$ heißt *Schaltkreisgröße*.
$D_\Omega(f) = \min\{$Tiefe eines Ω–Schaltkreises S mit $res_S = f\}$ heißt *Schaltkreistiefe*.

Bemerkung
(i) Sei $f \in I\!B_n$. Die Schaltkreistiefe und die Formeltiefe von f stimmen überein. *(ii)* Für die Schaltkreisgröße gilt: Ω, Ω' vollständige Basen $\Longrightarrow \exists c \in I\!R : \forall f \in I\!B_n : C_\Omega(f) \leq cC_{\Omega'}(f)$.

Beweis
(i) Die *Universelle Überlagerung* von S ist eine Formel mit gleicher Tiefe. Die Universelle Überlagerung eines azyklischen Graphen (mit einem Wurzelknoten) ist der Baum, der entsteht, wenn mehrfach benutzte Knoten vervielfältigt werden, so daß sie nur noch einen

Vorgänger haben.
(ii) Übungsaufgabe. ■
In der Definition der Ω–Schaltkreise war der Fan–out der Knoten nicht beschränkt. Dies ist aber nicht wesentlich:

Satz 2.6 (Hoover, Klawe, Pippenger 1984)
Zu jedem Schaltkreis S mit unbeschränktem Fan–out existiert ein Schaltkreis S' mit maximalem Fan–out 2, der die gleiche Boolesche Funktion realisiert, und für den gilt:

$$C(S') \leq 2C(S) \text{ und } D(S') \leq 2D(S). \quad ■$$

Bemerkung
Ω–Schaltkreise mit maximalem Fan–out 1 sind Ω–Formeln! Folglich gilt $C_\Omega(f) \leq L_\Omega(f)$ für alle f.

Von großer Bedeutung für paralleles Rechnen sind die sogenannten NC^i–Schaltkreise:

Definition 2.13
Ein *NC^i–Schaltkreis* über der Eingabemenge X_n ist ein Boolescher Schaltkreis polynomialer Größe der Tiefe $O(\log^i n)$.
NC–Schaltkreise sind Schaltkreise polynomialer Größe und polylogarithmischer Tiefe (d.h. Tiefe $O(\log^i n)$ für ein $i \in \mathbb{N}$).

2.5 Binäre Entscheidungsgraphen

Man kann Boolesche Funktionen auch durch *Entscheidungsbäume* und *Branching Programme* repräsentieren:

2.5.1 Entscheidungsbäume

Definition 2.14
Ein *Entscheidungsbaum (decision tree)* über X_n ist ein binärer Baum T, dessen

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

Die *Größe* eines Entscheidungsbaumes ist die Zahl der inneren Knoten.

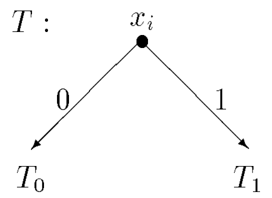
Ein Entscheidungsbaum T über X_n *repräsentiert (berechnet)* Boolesche Funktionen

$f_T \in \mathbb{B}_n$:

<u>1.Fall:</u> $T: \quad \bullet 0 \quad (1).$
Dann

$$f_T(x) = 0 \quad (1) \text{ für alle } x \in \{0,1\}^n$$

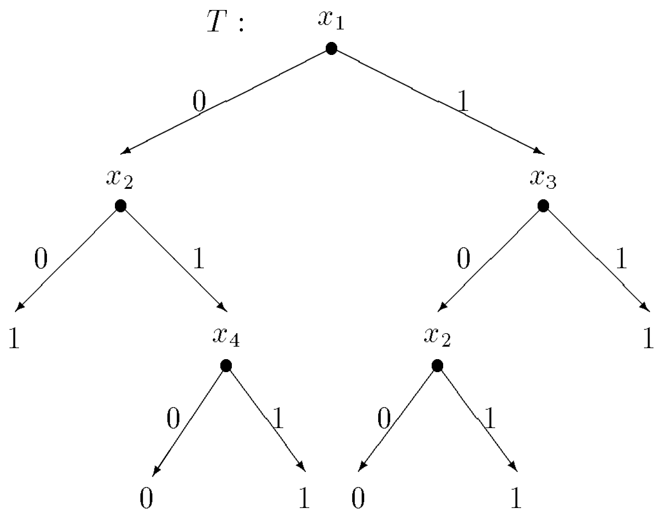
<u>2.Fall:</u>



Dann

$$\begin{aligned} f_T(x) &= \begin{cases} f_{T_0}(x) & \text{falls } x_i = 0 \\ f_{T_1}(x) & \text{falls } x_i = 1 \end{cases} \\ &= \overline{x}_i f_{T_0}(x) + x_i f_{T_1}(x). \end{aligned}$$

Beispiel 2.14



mit $f_T(1001) = 0$ und $f_T(1101) = 1$.

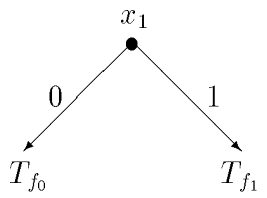
Lemma 2.2
Zu jeder Booleschen Funktion $f \in \mathbb{B}_n$ existiert ein Entscheidungsbaum T mit $f_T = f$.

Beweis
Wir erhalten einen Entscheidungsbaum für f durch „Visualisieren“ der Shannon–Entwicklung:

<u>$n = 0$:</u> Trivial.

<u>$(n-1) \rightarrow n$:</u>

$$f(x_1, x_2, x_3) \;=\; \overline{x}_1 \underset{\substack{\shortparallel \\ f_0(x_2, \ldots, x_n)}}{f(0, x_2, \ldots, x_n)} + x_1 \underset{\substack{\shortparallel \\ f_1(x_2, \ldots, x_n).}}{f(1, x_2, \ldots, x_n)}$$



■

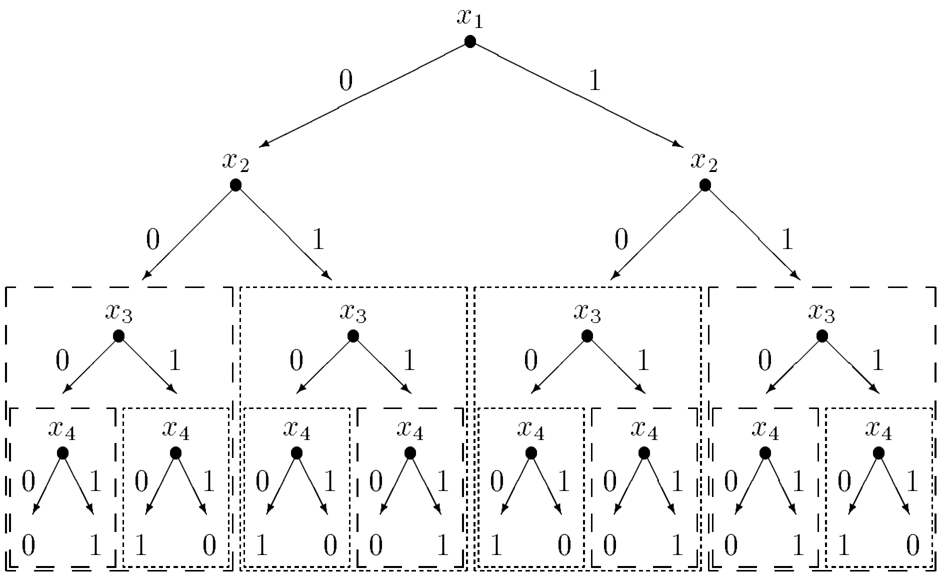
Definition 2.15
Sei $f \in \mathbb{B}_n$.
$DT(f) = \min\{$Größe eines Entscheidungsbaumes für $f\}$ heißt *Entscheidungsbaumgröße*.

Beispiel 2.15

$PAR_4(x_1, x_2, x_3, x_4) = x_1 \oplus x_2 \oplus x_3 \oplus x_4$.

Der folgende Entscheidungsbaum für PAR_4 macht deutlich, daß Entscheidungsbäume oft

- bei einfachen Funktionen schon exponentiell wachsen und dann
- viele übereinstimmende — sprich redundante — Teilbereiche aufweisen (hier durch vier verschiedene Einrahmungen markiert):



Jeder Knoten eines Entscheidungsbaums repräsentiert eine Boolesche Funktion (*Unterfunktion* von f_T). Durch Auszeichnung von m Ausgabeknoten können Entscheidungsbäume auch zur Berechnung bestimmter Funktionen aus $\mathbb{B}_{n,m}$ verwendet werden.

2.5.2 Branching Programme

Definition 2.16
Ein *Branching Programm* (BP) ist ein gerichteter azyklischer Graph, dessen

- Senken mit den Booleschen Konstanten 0 oder 1 bewertet sind,
- innere Knoten Ausgangsgrad 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* eines Branching Programms ist die Zahl der inneren Knoten.

Offensichtlich sind Entscheidungsbäume Branching Programme. In gleicher Weise wie Entscheidungsbäume *repräsentieren (berechnen)* Branching Programme Boolesche Funktionen aus $\mathbb{B}_n$.

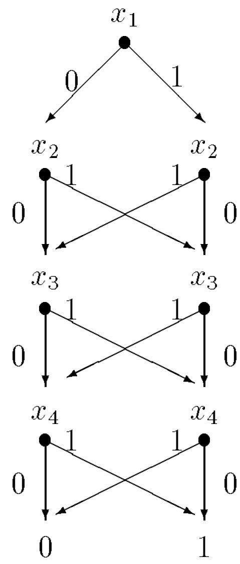
Definition 2.17
Sei $f \in \mathbb{B}_n$.
$BP(f) = \min\{$Größe eines Branching Programms für $f\}$ heißt *Branching–Programm–Größe*.

Sehr oft werden zusätzliche Bedingungen an die Struktur von BPs gestellt, z. B.

- *read–once–only*, d.h. jede Variable darf auf jedem Weg höchstens einmal getestet werden,
- *lesenormiert*, d.h. der dem Branching Programm zugrundeliegende Graph ist „gelevelt“ (jeder Knoten ist von der Wurzel eine eindeutige Zahl von Kanten entfernt, so daß man die Knotenmenge in Ebenen unterteilen kann, daher der Name) und auf jedem Niveau wird in allen Knoten die gleiche Variable getestet.

Beispiel 2.16

$PAR(x_1, x_2, x_3, x_4) = x_1 \oplus x_2 \oplus x_3 \oplus x_4$
P:



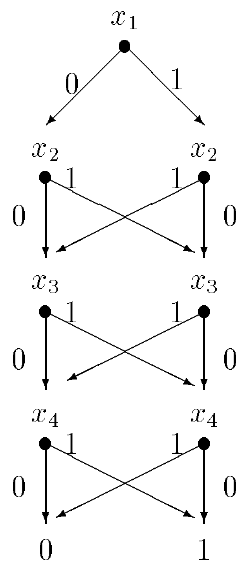
2.5.3 Reduzierte und geordnete binäre Entscheidungsgraphen (oBDD)

Sei π eine feste Ordnung auf X_n (Variablenordnung).
Betrachte lesenormierte read–once–only Branching Programme über X_n, bei denen die Variablen in der vorgegebenen Ordnung getestet werden.

Offenbar läßt sich jede Boolesche Funktion durch ein solches Branching Programm repräsentieren.

Beispiel 2.17

$PAR_4(x_1, x_2, x_3, x_4)$, $\pi = (x_1, x_2, x_3, x_4)$



oBDD liefern kanonische Repräsentationen für Boolesche Funktionen:

Definition 2.18
Zwei Branching Programme P, P' heißen *isomorph* genau dann, wenn ein Graphisomorphismus α von P nach P' existiert mit der Eigenschaft, daß für alle Knoten v aus P v und $\alpha(v)$ mit der gleichen Variablen bzw. dem gleichen Wert markiert sind und daß die Werte der Kanten (v, u) und $(\alpha(v), \alpha(u))$ übereinstimmen.

Weiter kann man isomorphe Unterprogramme untersuchen:

Definition 2.19
Sei P Branching Programm, $v \in P$. P_v bezeichnet das in v startende Unterprogramm von P. P heißt *reduziert* genau dann, wenn für alle $v, v' \in P$ gilt: P_v und $P_{v'}$ sind *nicht* isomorph.

Definition 2.20
Sei π eine Variablenordnung von X_n. Ein oBDD ist ein reduziertes BP ohne triviale Tests, bei denen die Variablen in der vorgegebenen Reihenfolge höchstens einmal getestet werden.

Satz 2.7
oBDDs liefern eine kanonische Repräsentation Boolescher Funktionen, d.h. in Bezug auf eine vorgegebene Ordnung der Variablen läßt sich jede Boolesche Funktion f eindeutig durch ein oBDD repräsentieren.

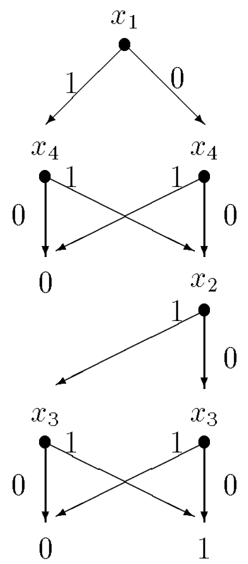
Beweis
Existenz: klar
Eindeutigkeit:
Seien P, P' zwei oBDD mit $f_P = f_{P'}$, dann genügt es, zu zeigen, daß P und P' isomorph sind. Übungsaufgabe. ■

Beachte, daß verschiedene Variablenordnungen zu verschiedenen oBDDs führen.

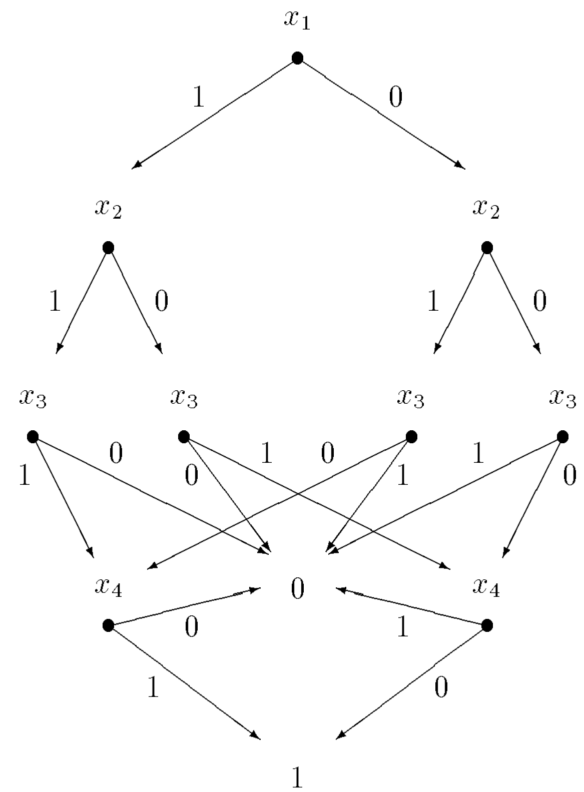
Definition 2.21
Sei π eine vorgegebene Variablenordnung.
$oBDD_\pi(f)$ ist die Größe des (bzgl. der Variablenordnung π) eindeutig bestimmten oBDD für f.

Beispiel 2.18
Sei $f(x_1, \ldots, x_4) = 1 \iff (x_1 = x_4) \wedge (x_2 = x_3)$.

Für $\pi = \{x_1, x_4, x_2, x_3\}$ ergibt sich die Größe 6:



Für $\pi = \{x_1, x_2, x_3, x_4\}$ ist dagegen die minimale Größe 9:



2.6 DNF–Repräsentation und PLA–Realisierung

Die Logiksynthese ist wesentlich gerichtet auf die Bereitstellung effizienter Realisierungen vorgegebener Boolescher Funktionen („Chiprealisierung").
Die Art der Chiprealisierung hängt entscheidend von der bereitgestellten Technologie ab. Aus der spezifischen Sicht möglicher Technologien ist die Verwendung jeweils bestimmter Darstellungen für die zu realisierende Boolesche Funktionen besonders naheliegend, wie z. B.

kDNF (Tabellenform)	für	ROM–Realisierungen
DNF	für	PLA–Realisierungen
Ω–SK	für	Realisierungen mit Ω–Gattern
BP	für	Multiplexer–Realisierungen

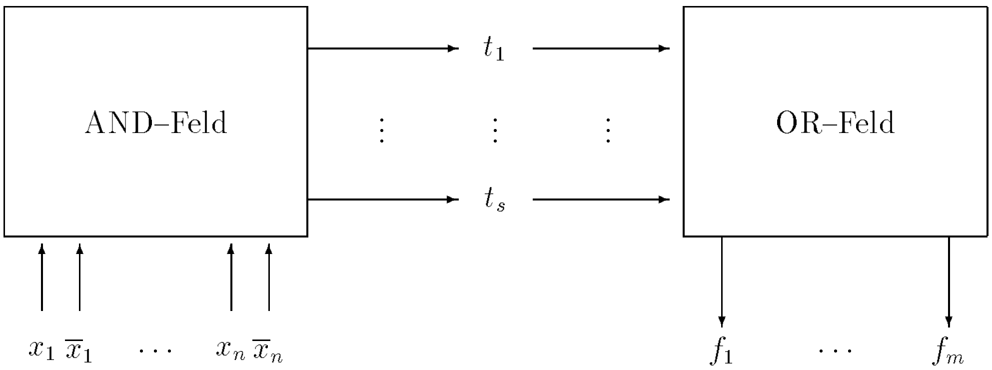
Wir wollen hier PLA–Realisierungen genauer betrachten. Sei $f \in \mathbb{B}_{n,m}$ repräsentiert durch DNF ($(\wedge, \vee)$–Polynom)

$$\begin{aligned} f_1(x) &= t_1^{(1)}(x) \vee \ldots \vee t_{k_1}^{(1)}(x) \\ &\vdots \\ f_m(x) &= t_1^{(m)}(x) \vee \ldots \vee t_{k_m}^{(m)}(x) \end{aligned}$$

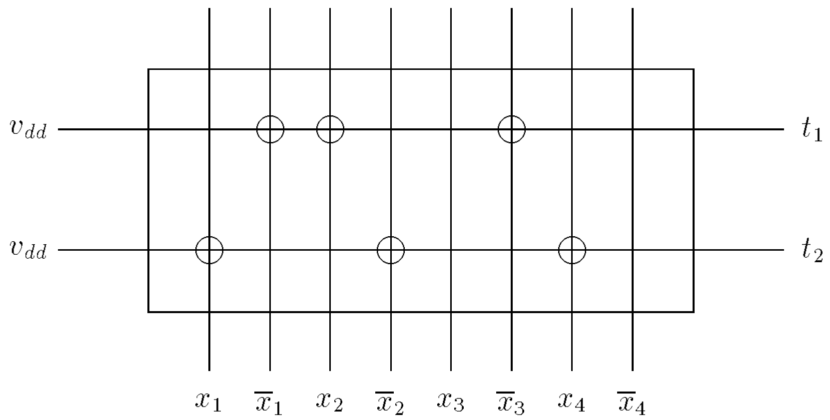
mit $x = (x_1, \ldots, x_n) \in \{0,1\}^n$, wobei $t_j^{(i)}$ Monome aus der Menge $\{t_1, \ldots, t_s\}$ sind.

Ein *programmierbares logisches Feld* (PLA) zur Realisierung von $f_1, \ldots, f_m$ besteht aus

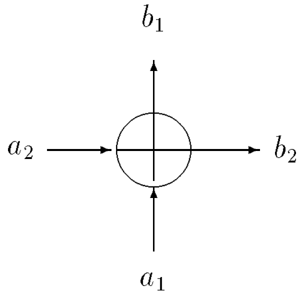
- einem AND–Feld mit $2n$ Spalten und s Zeilen und
- einem OR-Feld mit m Spalten und s Zeilen.



Beispiel 2.19
$t_1 = x_1\overline{x}_2x_3 \in \mathbb{B}_4$, $t_2 = \overline{x}_1x_2\overline{x}_4 \in \mathbb{B}_4$.
Aufbau des AND–Feldes:



Schalter I:

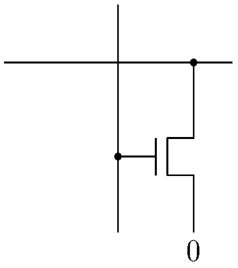


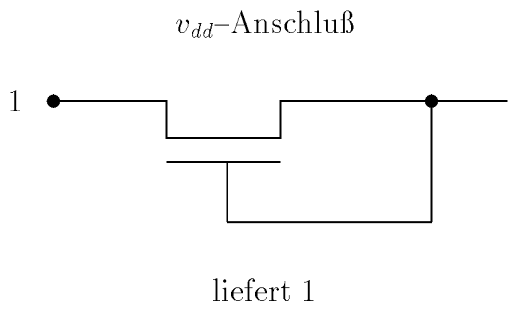
realisierte Funktion:

a_1	a_2	b_1	b_2
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	0

d. h. $b_1 := a_1, \quad b_2 := a_2\overline{a}_1$

Transistorrealisierung:

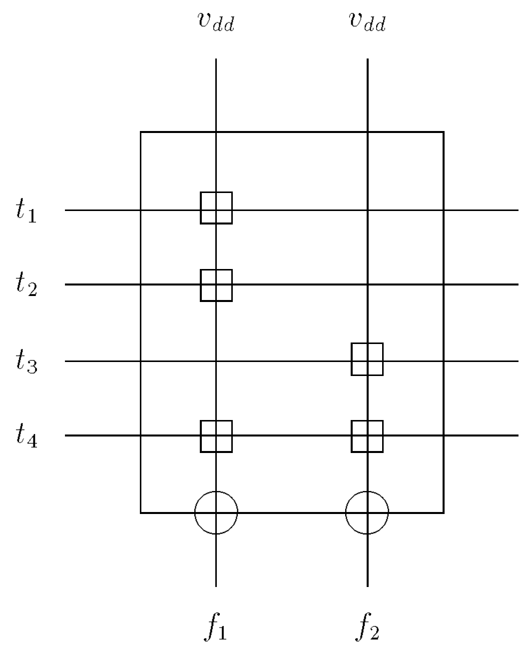




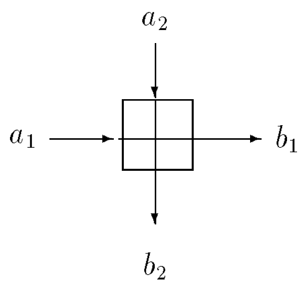
Das AND–Feld berechnet t_1, t_2 korrekt:

$$t_1 = 1 * \overline{\overline{x}_1}\,\overline{x}_2\overline{\overline{x}_3} = x_1\overline{x}_2x_3$$
$$t_2 = 1 * \overline{x}_1\overline{\overline{x}_2}\,\overline{x}_4 = \overline{x}_1x_2\overline{x}_4$$

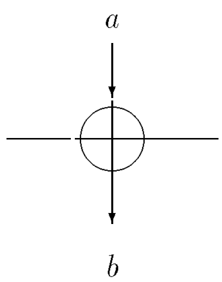
Beispiel 2.20
$f_1 = t_1 + t_2 + t_4$, $f_2 = t_3 + t_4$
Aufbau des OR–Feldes:



Schalter II:



Konverter:



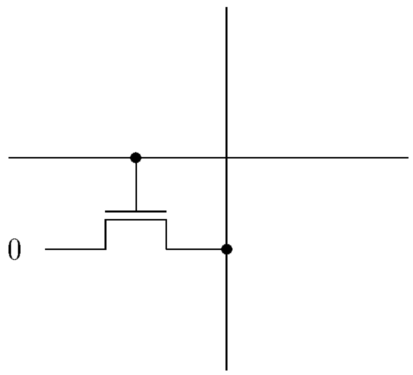
realisierte Funktionen:

a_1	a_2	b_1	b_2
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	0

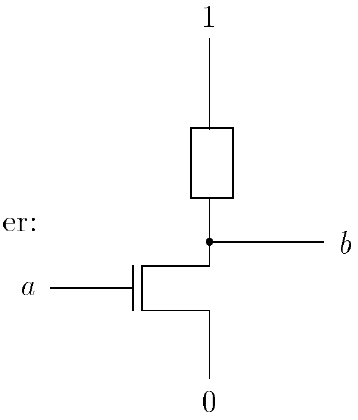
$(b_1 := a_1, \quad b_2 := a_2\overline{a}_1)$ bzw. $b := \overline{a}$

Transistorrealisierung:

Schalter II:



Konverter:



Das OR–Feld berechnet f_1, f_2 korrekt:

$$f_1 = \overline{1 * \overline{t_1}\,\overline{t_2}\,\overline{t_4}} = \overline{\overline{t_1}\,\overline{t_2}\,\overline{t_4}} = t_1 + t_2 + t_4$$
$$f_2 = \overline{1 * \overline{t_3}\,\overline{t_4}} = \overline{\overline{t_3}\,\overline{t_4}} = t_3 + t_4$$

Bemerkung
Der *Flächenbedarf* einer PLA–Realisierung ergibt sich als

$$(2n + m) * s,$$

wobei n die Zahl der Eingabebits, m die Zahl der Ausgabebits und s die Zahl der verschiedenen Monome ist. Eine Flächenoptimierung hat das Ziel, $f = (f_1, \ldots, f_m)$ mit möglichst wenigen Monomen zu realisieren.

Kapitel 3

Komplexitätsaussagen

3.1 Vorbemerkungen

- Die Komplexität der Algorithmen zur Logiksynthese und –optimierung hängt entscheidend ab von der Komplexität der Darstellung der zu verarbeitenden Booleschen Funktionen. Die Eingabegröße entspricht eigentlich nicht der Zahl der Argumente, sondern der Größe der Darstellung.

- $\left.\begin{array}{l} DNF(f), KNF(f), PNF(f), RSE(f) \\ L(f), C(f), D(f) \\ BP(f), BDD(F) \end{array}\right\}$ spiegeln minimalen Aufwand zur Darstellung einer Booleschen Funktion $f \in \mathbb{B}_n$ in Form einer $DNF, \ldots$ wider.

- Beachte den fundamentalen Unterschied zum üblichen Softwarebereich:
  Boolesche Funktionen spezifizieren eine Problemlösung für eine fest vorgegebene Eingabelänge, während Programme uniform auf Eingaben beliebiger Länge arbeiten. Berechnungsmodelle für Boolesche Funktionen heißen deshalb <u>nichtuniform</u>.

- Um Tendenzen im Aufwand bestimmen zu können, werden anstelle einer Funktion $f_n \in \mathbb{B}_n$ Funktionenfolgen $(f_n)_{n\in\mathbb{N}}$, $f_n \in \mathbb{B}_n$, betrachtet:
  <u>Z. B.</u>

  $(PAR_n)_{n\in\mathbb{N}}$ mit $PAR_n \in \mathbb{B}_n$, $PAR_n(x_1, \ldots, x_n) = x_1 \oplus \ldots \oplus x_n$,

  $(ADD_n)_{n\in\mathbb{N}}$ mit ADD_n Addition zweier n–bit Zahlen.

  Damit kann die Entwicklung des Aufwands exakter, nämlich *in Abhängigkeit von n* beschrieben werden.

- Exakte Bestimmung von DNF(f_n), ... für konkrete Funktionen ist oft äußerst schwierig. Man ist deshalb an *oberen* und *unteren Schranken* interessiert und an *typischen* Werten (sogenannten *asymptotischen Aussagen*):

  – *Obere* bzw. *untere Schranken* schätzen den Aufwand nach oben bzw. unten ab (mathematische Bezeichnung O(...) bzw. Ω(...));

– *Typische Werte* sind Aussagen der Gestalt
„für fast alle *Booleschen Funktionen einer Klasse* $\mathbb{F}_n \subseteq \mathbb{B}_n$ gilt die Eigenschaft E“

$$\frac{\#\{f \in \mathbb{F}_n \text{ mit Eigenschaft E}\}}{\#\mathbb{F}_n} \longrightarrow 1 \qquad \text{für } n \to \infty$$

- Darstellungen heißen *effizient*, falls sie von polynomialer Größe (in der Zahl der Eingabevariablen) sind.
Die folgende Tabelle macht diese Begriffsbildung plausibel:

	Maximale Eingabelänge bei vorgegebener Rechenzeit von			
T(n)	0,01 Sek.	1 Sek.	1 Min.	1 Std.
$\log(n)$	1024	$> 10^{300}$	10^{18000}	$> 10^{108000}$
n	10	1000	60000	3600000
n^2	3	31	244	1897
2^n	3	9	15	21

- Die Betrachtung von Funktionenfolgen liefert Möglichkeiten, übliche komplexitätstheoretische Konzepte anzuwenden. Z. B. kann eine Funktionenfolge $F = (f_n)_{n\in\mathbb{N}}$, $f_n \in \mathbb{B}_n$, als Sprache $L_F \subseteq \{0,1\}^*$ aufgefaßt werden vermöge

$$w \in L_F \cap \{0,1\}^n \iff f_n(w) = 1$$

und umgekehrt. Man kann dadurch die Komplexität nichtuniformer Problemlösungen der Theorie der Booleschen Funktionen mit uniformen Lösungen aus dem Softwarebereich vergleichen.
Es sei daran erinnert, daß **P** die Menge der effizient uniform lösbaren Probleme darstellt.

3.2 Komplexität Boolescher Polynome

3.2.1 Effiziente Darstellung mit Booleschen Polynomen

Definition 3.1
$P_{S_{(\omega,\omega')}}$ ist die Menge aller Folgen Boolescher Funktionen f mit (ω,ω')–Polynomdarstellung polynomialer Größe $S_{(\omega,\omega')}$.
Wir werden insbesondere folgende Mengen betrachten: P_{DNF}, P_{KNF}, P_{PNF}, P_{RSE}.

Von besonderem Interesse ist P_{DNF}:

Satz 3.1
Eine Boolesche Funktion f besitzt eine effiziente PLA–Realisierung, d.h. eine PLA–Realisierung polynomialer Fläche, genau dann, wenn $f \in P_{DNF}$.

Beweis
Sei $f \in \mathbb{B}_{n,m}$. Die PLA–Fläche ergibt sich als Produkt von $(2n + m)$ und der Zahl der Monome einer DNF–Darstellung für f, die sich höchstens um den Faktor n von der Länge dieser Darstellung unterscheiden kann. ■

Korollar 3.1
Für den minimalen Flächenbedarf A_f einer PLA–Realisierung von $f \in \mathbb{B}_{n,m}$ gilt:

$$\frac{DNF(f)}{n}(2n+m) \leq A_f \leq DNF(f)(2n+m). \quad ■$$

Beispiele für wichtige Funktionen aus P_{DNF}:

- *Boolesche Convolution:*
  $BCONV_n \in \mathbb{B}_{2n,2n}$ mit

  $$BCONV_n(x_0,\ldots,x_{n-1},y_0,\ldots,y_{n-1}) = (z_0,\ldots,z_{2n-1}) \text{ mit } z_k = \bigvee_{i+j=k} x_i y_j$$

  $BCONV_n$ ist z. B. in der Signal– oder Bildverarbeitung wichtig.
  Boolesche Convolution ist eine Boolesche Entsprechung zur Koeffizientenberechnung bei Polynommultiplikation.

- *Speicherzugriffsfunktion:*
  Betrachte einen Booleschen Speicher mit $n = 2^k$ Speicherworten der Länge l. Wir wollen das i–te Speicherwort $(0 \leq i < n)$ über eine Adresse $a = (a_{k-1},\ldots,a_0)$ mit $\text{bin}(a) = i$ ansprechen.

- *Boolesche Matrizenmultiplikationen*
  Wichtig z. B. in der Graphentheorie.

Nur wenige Funktionen gehören zu $P_{DNF}, \ldots$!

3.2.2 Unterschiedliche Berechnungskraft der verschiedenen Booleschen Polynome

Zum Nachweis von Unterschieden in der Berechnungskraft gehört der Nachweis von unteren und oberen Schranken.

obere Schranken	erfordern im allgemeinen Simulationsresultate,
untere Schranken	(sehr schwierig) erfordern den Nachweis der prinzipiellen Unmöglichkeit der Berechnung bei mangelnden Ressourcen.

Verhältnis von PNF und RSE

Offenbar gilt für alle Booleschen Funktionen f:

$$PNF(f) \leq RSE(f)$$

Betrachte $f(x_1, x_2, x_3) = \overline{x}_1 * \overline{x}_2 * \ldots * \overline{x}_n$:

Lemma 3.1
(i) $PNF(f) = n$,
(ii) $RSE(f) = 2^n$.

Beweis
(i) Klar.
(ii) Eine RSE-Darstellung für f ergibt sich aus

$$f(x_1, x_2, x_3) = (x_1 \oplus 1) * \ldots * (x_n \oplus 1) \qquad (*)$$

RSE–Darstellungen sind eindeutig.
Folglich besitzt f nur die RSE–Darstellung mit 2^n Termen, die durch Ausmultiplizieren der Faktoren von $(*)$ entsteht, d.h. $RSE(f) \geq 2^n$. ■

Verhältnis von DNF, KNF und PNF

Wir weisen nach, daß diese drei Maße paarweise verschieden sind.
Betrachte $MAJ_n, PAR_n \in \mathbb{B}_n$ und $M_{n,n-1} \in \mathbb{B}_{n^2}$ mit:

$$\begin{aligned} MAJ_n(x_1, \ldots, x_n) = 1 \quad &\Longleftrightarrow \quad \sum x_i > \left\lfloor \frac{n}{2} \right\rfloor, \\ PAR_n(x_1, \ldots, x_n) = 1 \quad &\Longleftrightarrow \quad \bigoplus x_i = 1 \text{ und} \\ M_{n,n-1}\begin{pmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & & \vdots \\ x_{n1} & \cdots & x_{nn} \end{pmatrix} = 1 \quad &\Longleftrightarrow \quad \sum_{j=1}^{n} x_{ij} \geq n-1 \qquad \text{für alle } i. \end{aligned}$$

Lemma 3.2
Für PAR_n *gilt:*

(i) $DNF(\mathrm{PAR}_n) = KNF(\mathrm{PAR}_n) = n * 2^{n-1}$,

(ii) $PNF(\mathrm{PAR}_n) = n$.

Beweis
(i) Wir erhalten eine DNF für f durch eine Würfelüberdeckung von $on(f) = f^{-1}(1)$, wobei den einzelnen Würfeln Monome entsprechen.
$on(f)$ besteht aus 2^{n-1} Punkten, von denen keine zwei auf einer Geraden (1–dimensionaler Würfel) liegen. Folglich werden 2^{n-1} 0-dimensionale Würfel, also Minterme, benötigt, um $on(f)$ zu überdecken.
Die Argumentation für die KNF verläuft analog, indem von $\{0,1\}^n$ die Ecken aus $o\!f\!f(f)$ einzeln „abgeschnitten“ werden.
(ii) Klar. ■

Lemma 3.3
Für $M_{n,n-1}$ gilt:
(*i*) $KNF(M_{n,n-1}) = \mathrm{O}(n^3)$,
(*ii*) $DNF(M_{n,n-1}) = 2^{\Omega(n)}$,
(*iii*) $PNF(M_{n,n-1}) = 2^{\Omega(n \log n)}$.

Beweis
(*i*) Offenbar gilt

$$M_{n,n-1}(x_{11} \dots x_{nn}) = \bigwedge_{i=1}^{n} \bigwedge_{1 \le j < k \le n} (x_{ij} \vee x_{ik})$$

und damit

$$KNF(M_{n,n-1}) \le n * \frac{n(n-1)}{2} * 2 \; .$$

(*ii*) Ergibt sich aus der Monotonie von $M_{n,n-1}$ und der Tatsache, daß für monotone Funktionen die minimale DNF–Darstellung eindeutig bestimmt ist.
(*iii*) Ist ein aktuelles (1991) Forschungsresultat von Damm und Meinel und basiert auf Razborovschem Approximationskalkül. ■

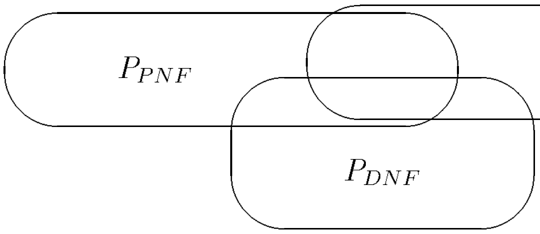
Korollar 3.2
Für $\neg M_{n,n-1}$ gilt:
(*i*) $KNF(\neg M_{n,n-1}) = 2^{\Omega(n)}$,
(*ii*) $DNF(\neg M_{n,n-1}) = O(n^3)$,
(*iii*) $PNF(\neg M_{n,n-1}) = 2^{\Omega(n \log n)}$.

Beweis
(*i*) und (*ii*) folgen aus Lemma 3.3 und de Morganschen Regeln. (*iii*) $\neg M_{n,n-1} = M_{n,n-1} \oplus 1$.
■

Lemma 3.4
Für MAJ_n gilt:
(*i*) $KNF(MAJ_n) = 2^{\Omega(n)}$,
(*ii*) $DNF(MAJ_n) = 2^{\Omega(n)}$,
(*iii*) $PNF(MAJ_n) = 2^{\Omega(n)}$.

Beweis
MAJ_n kann nicht in AC^0 reduziert werden auf PAR_n.[1] Somit ergeben sich (i)–(iii) aus entsprechenden Eigenschaften für PAR_n. ■

Satz 3.2
Die Komplexitätsmaße DNF, KNF, PNF sind paarweise unvergleichbar. Für die Klassen Boolescher Funktionen mit effizienten DNF–, KNF– bzw. PNF–Darstellungen gilt:

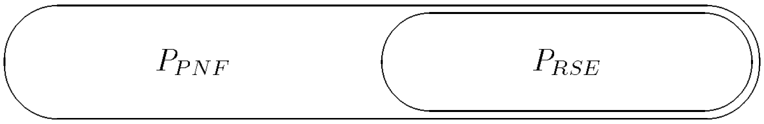


■

Satz 3.3
Die Komplexitätsmaße RSE und PNF stehen für alle Booleschen Funktionen f in der Beziehung

$$RSE(f) \geq PNF(f).$$

Für die Klassen Boolescher Funktionen mit effizienten RSE– bzw. PNF–Darstellungen gilt:



■

3.2.3 Typische Größe von Minimalpolynomen

Wir haben Funktionen mit effizienter DNF–Darstellung kennengelernt ($BCONV_n$) und mit sehr aufwendigen Minimalpolynomen ($PAR_n, \ldots$). Wie sieht die typische Größe eines Minimalpolynoms für eine Boolesche Funktion $f \in \mathbb{B}_n$ aus?

Satz 3.4
Für fast alle Booleschen Funktionen $f \in \mathbb{B}_n$ gilt für die Zahl m_f der Monome in einem Minimalpolynom für f

$$\frac{(1-\varepsilon_n)2^n}{\log n \log\log n} \leq m_f \leq \frac{1,6 * 2^n}{\log n \log\log n},$$

wobei $\varepsilon_n \longrightarrow 0$ für $n \to \infty$.

[1] D. h. MAJ_n kann nicht mit einem $\{\wedge, \vee, PAR_n\}$–Schaltkreis konstanter Tiefe berechnet werden.

Beweis
Erfordert recht aufwendige kombinatorische Untersuchungen. ■

Korollar 3.3
Für die minimale PLA–Fläche A_f zur Realisierung fast aller Boolescher Funktionen $f \in \mathbb{B}_n$ gilt:

$$\frac{(1+2n)(1-\varepsilon_n)2^n}{\log n \log\log n} \leq A_f \leq \frac{1,6 * 2^n(2n+1)}{\log n \log\log n},$$

wobei $\varepsilon_n \longrightarrow 0$ für $n \to \infty$. ■

Zum Vergleich: kDNF haben höchstens 2^n Minterme und typischerweise 2^{n-1} Minterme (für fast alle Booleschen Funktionen $f \in \mathbb{B}_n$).

Satz 3.5
Für fast alle Booleschen Funktionen $f \in \mathbb{B}_n$ übersteigt die maximale Länge eines Monoms in einem Minimalpolynom $\lfloor \log n \rfloor + 1$ nicht. ■

Korollar 3.4
Für fast alle Booleschen Funktionen $f \in \mathbb{B}_n$ gilt:

$$\frac{(1-\varepsilon_n)2^n}{\log\log n} \leq DNF(f) \leq \frac{1,6 * 2^n}{\log\log n},$$

wobei $\varepsilon_n \longrightarrow 0$ für $n \to \infty$. ■

Beachte: Die Aussagen gelten zwar für fast alle Booleschen Funktionen, Boolesche Funktionen von praktischem Interesse sind aber häufig hochstrukturiert und somit keine „typischen“ Booleschen Funktionen.

3.3 Komplexität Boolescher Formeln und Schaltkreise

3.3.1 Effiziente Darstellung mit Booleschen Formeln und Schaltkreisen

Definition 3.2
$P_{\Omega-\mathrm{F}}$ bzw. $P_{\Omega-\mathrm{SK}}$ ist die Menge aller Folgen Boolescher Funktionen f, die berechnet werden können durch Ω–Formeln bzw. Ω–SK polynomialer Größe.
Wir schreiben kurz P_{F} bzw. P_{SK} für $\Omega = \mathbb{B}_2$.

Satz 3.6
Kann $f = (f_n) \in P_{\Omega-\mathrm{SK}}$ *(mit* $\Omega \subseteq \bigcup_{i \le k} \mathbb{B}_i$*) berechnet werden durch einen Schaltkreis polynomialer Größe und logarithmischer Tiefe, dann gilt:* $f \in P_{\Omega-\mathrm{F}}$.

Beweis
Wir können den Graph der Tiefe $c \log n$, der dem Schaltkreis zugrundeliegt, zum Baum der Größe $< k^{c \log n} = k' n^c + \ldots$ entwickeln, ohne die Tiefe zu verändern.
Damit erhalten wir eine polynomiale Formel. ■

Satz 3.7
Ist $(f_n) \in P_F$*, dann gilt:*

$$D(f) = \mathrm{O}(\log n).$$

Beweis
Jeder binäre Baum T besitzt einen Knoten v, so daß für den in v wurzelnden Unterbaum T_v gilt:

$$\frac{1}{3}\,\#T \le \#T_v \le \frac{2}{3}\,\#T \qquad \text{(Übungsaufgabe).}$$

Sei F optimale Formel für f. Im F zugrundeliegenden Baum existiert ein Knoten v mit

$$\frac{\text{Größe}(F)}{3} \le \text{Größe}(F_v) \le \frac{2\text{Größe}(F)}{3} \quad .$$

Sei F_i die Formel, die sich aus F durch Ersetzen von F_v durch Konstante i, $i = 0, 1$, ergibt.
Damit ist $F' = (F_v \wedge F_1) \vee (\neg F_v \wedge F_0)$ äquivalent zu F.
Nach Induktionsvoraussetzung seien $F_v, \neg F_v, F_0, F_1$ realisierbar durch Schaltkreise der Tiefe $c_1 \log(\text{Größe}(F_v)), \ldots, c_4 \log(\text{Größe}(F_1))$.
Dann gilt:

$$\begin{aligned}
\text{Tiefe}(F') &\le \max\{\text{Tiefe}(F_v), \ldots, \text{Tiefe}(F_1)\} + 2 \\
&\le \max\{c_1 \log(\text{Größe}(F_v)), \ldots, c_4 \log(\text{Größe}(F_1))\} + 2 \\
&= \mathrm{O}(\log(\text{Größe}(F))). \quad ■
\end{aligned}$$

Korollar 3.5
$P_{\mathrm{F}} = \mathrm{NC}^1$. ■

Satz 3.8
$\mathbf{P} \subseteq P_{\mathrm{SK}}$.

Beweis
Grundvorlesung. ■

Ergänzung:

Satz 3.9
$BPP \subseteq P_{\mathrm{SK}}$, wobei $BPP = \{L | \text{es existiert prob. Polynomzeit–Turingmaschine } M \text{ mit } Prob(M(x) = \mathcal{X}_L(x)) > 1/2 + \varepsilon \ , \ \varepsilon > 0\}$. ■

(Es gilt $\mathbf{P} \subseteq BPP$. Unklar ist, ob $\mathbf{P} \overset{\subset}{\neq} BPP$).

Beispiele wichtiger Vertreter aus $P_{\mathrm{F}} \subseteq P_{\mathrm{SK}}$:

- ADD_n: Addition zweier n–Bit–Zahlen:
  $C(ADD_n) = \mathrm{O}(n)$
  $D(ADD_n) = \mathrm{O}(\log n)$

- $MULT_n$: Multiplikation zweier n–Bit–Zahlen:
  $C(MULT_n) = \mathrm{O}(n \log n \log\log n)$
  $D(MULT_n) = \mathrm{O}(\log n)$

- DIV_n: Division zweier n–Bit–Zahlen (ebenso Matrizen-Multiplikation):
  $C(DIV_n)$ ist polynomial
  $D(DIV_n) = \mathrm{O}(\log n)$

- PAR_n:
  $C(PAR_n) = n - 1$
  $D(PAR_n) = \lceil \log n \rceil$

Weiterhin gilt :

- Determinantenberechnung, Matrixinversion und die Überprüfung einer Reihe von Grapheigenschaften gehören zu P_{SK}.
  Es ist aber unklar, ob sie auch zu P_{F} gehören.

Allgemeine Vermutung:

$$P_{\mathrm{F}} \overset{\subset}{\neq} P_{\mathrm{SK}} \qquad (\text{also } \mathrm{NC}^1 \overset{\subset}{\neq} \mathbf{P})$$

3.3.2 Typische Größe von Booleschen Formeln und Schaltkreisen

3.3.2.1 Formeln:

Satz 3.10
(i) Für fast alle Booleschen Funktionen $f \in \mathbb{B}_n$ gilt:

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

(ii) Für alle Booleschen Funktionen $f \in \mathbb{B}_n$ gilt:

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

Beweis

(i) Wir benutzen das folgende

> **Lemma**
> *Höchstens*
> $$\alpha(n,b) := (n+2)^{b+1} 16^b 4^b$$
> *Funktionen $f \in \mathbb{B}_n$ können von $\mathbb{B}_2$–Formeln der Größe b berechnet werden.*
>
> **Beweis**
> Übungsaufgabe. □

Sei $\mathbb{B}_n^*$, $\#\mathbb{B}_n^* = 2^{2^n(1-\log^{-1} n)}$, 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:

$$\alpha(n, l_n^*) \geq 2^{2^n(1-\log^{-1} n)} \qquad (*)$$

$(*)$ 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 $L(f) \geq l$ und damit

$$L(f) \geq l$$

für alle $f \in \mathbb{B}_n \setminus \mathbb{B}_n^*$. Wegen

$$\#(\mathbb{B}_n \setminus \mathbb{B}_n^*) = 2^{2^n} - 2^{2^n(1-\log^{-1} n)} = 2^{2^n}(1 - 2^{-2^n \log^{-1} n}) \longrightarrow 2^{2^n} \text{ falls } n \to \infty$$

gilt

$$\frac{\#\{f \in \mathbb{B}_n \text{ mit } L(f) \leq l\}}{\#\mathbb{B}_n} \longrightarrow 0 \text{ für } n \to \infty.$$

(ii) Nachweis über Lupanowsche (k, s)–Repräsentation; Siehe hierzu den Beweis von Satz 3.11. ■

3.3.2.2 Schaltkreise:

Satz 3.11
(i) Für fast alle Booleschen Funktionen $f \in \mathbb{B}_n$ gilt:

$$C(f) = \Omega\left(\frac{2^n}{n}\right).$$

(ii) Für alle Booleschen Funktionen $f \in \mathbb{B}_n$ gilt:

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

Beweis
(i) Wir benutzen das folgende

> **Lemma**
> *Höchstens*
>
> $$\alpha(n, c) = (c + n + 1)^{2c} 16^c c/c!$$
>
> *viele Funktionen $f \in \mathbb{B}_n$ können mit $\mathbb{B}_2$–Schaltkreisen der Größe c berechnet werden.*
>
> **Beweis**
> Übungsaufgabe. □

Analog wie im Satz über Formeln sei

$$\mathbb{B}_n^* \subseteq \mathbb{B}_n, \; \#\mathbb{B}_n^* = 2^{2^n - 2^n n^{-1} \log\log n}$$

die Menge der Booleschen Funktionen $f \in \mathbb{B}_n$ mit den kleinsten Schaltkreisen.
Sei $c_n^* = \max\{c(f) | f \in \mathbb{B}_n^*\}$. Offenbar gilt:

$$\alpha(n, c_n^*) \geq \#\mathbb{B}_n^* = 2^{(2^n - 2^n n^{-1} \log\log n)} \qquad (*)$$

$(*)$ ist für große n noch nicht erfüllt für

$$c = \frac{2^n}{n}.$$

Folglich existiert ein $f \in \mathbb{B}_n^*$ mit $C(f) \geq c$ und damit $C(f) \geq c$ für alle $f \in \mathbb{B}_n \setminus \mathbb{B}_n^*$. Die Behauptung ergibt sich nun analog wie in Satz 3.10.

(ii) Lupanow (1958) und Shannon (1949) haben ein relativ einfaches Schema zur Konstruktion von Schaltkreisen der Größe $O\left(\frac{2^n}{n}\right)$ für beliebige 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 $\frac{2^n}{n} + o\left(\frac{2^n}{n}\right)$ bereitstellt.
Die Konstruktion basiert auf Lupanows (k,s)–Repräsentation.
Sei $f \in \mathbb{B}_n$.
Wir betrachten die Funktionstabelle für f als $2^k \times 2^{n-k}$–Matrix:

- Zeilen entsprechen 2^k Werten für $(x_1, \ldots, x_k)$
- 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 $p = \left\lceil \frac{2^k}{s} \right\rceil$ Blöcken $A_1, \ldots, A_p$ zusammen.
($A_1, \ldots, A_{p-1}$ enthalten genau s Zeilen, A_p enthält $s' \leq s$ Zeilen)
Betrachte f_i mit

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

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:

$n = 6, k = 3, s = 2:$

	f										
			0	0	0	0	1	1	1	1	x_4
			0	0	1	1	0	0	1	1	x_5
			0	1	0	1	0	1	0	1	x_6
0	0	0	0	1	1	0	1	1	1	0	
0	0	1	1	0	0	1	0	1	0	1	
0	1	0	1	0	1	1	0	0	1	0	
0	1	1	0	0	0	1	1	0	0	0	
1	0	0	1	1	1	0	0	1	1	0	
1	0	1	0	1	1	0	1	0	1	0	
1	1	0	1	0	0	1	1	0	0	1	
1	1	1	1	1	0	0	0	0	1	1	
x_1	x_2	x_3									

Z. B. $B_{2,\binom{0}{0}} = \{1,5,7\}$, $B_{3,\binom{1}{0}} = \{0,5\}$.
Betrachte weiter $f_{i,w}$ mit

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

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

$$f_{i,w}(x) = 1 \iff \begin{array}{l} (1)\ (x_{k+1}, \ldots, x_n) \in B_{i,w} \\ (2) \text{ Für ein } j \text{ mit } w_j = 1 \text{ ist } (x_1, \ldots, x_k)\ j\text{–te Zeile von } A_i \end{array}$$

Ist $f_{i,w}^{(1)}$ Boolesche Funktion, die (1) überprüft, und $f_{i,w}^{(2)}$ Boolesche Funktion, die (2) überprüft, dann gilt $f_{i,w}(x) = f_{i,w}^{(1)} \wedge f_{i,w}^{(2)}$.
Offenbar hängt $f_{i,w}^{(1)}$ lediglich ab von $(x_{k+1}, \ldots, x_n)$ und $f_{i,w}^{(2)}$ von $(x_1, \ldots, x_k)$.
Wir erhalten nun die *Lupanowsche (k,s)–Darstellung*:

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

<u>Konstruktion eines Booleschen Schaltkreises der Tiefe 4 für f:</u>

1.) *Konjunktion*
Bilde alle 2^k Minterme auf $(x_1, \ldots, x_k)$ und alle 2^{n-k} Minterme auf $(x_{k+1}, \ldots, x_n)$.
(Wir benötigen dazu $(k-1)2^k + (n-k-1)2^{n-k}$ AND's.)

2.) *Disjunktion*
Berechne die 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 Blöcke A_i und 2^{n-k} Minterme gibt, genügen $p2^{n-k}$ OR's.)
Berechne die kDNF für alle $f_{i,w}^{(2)}$.
(Für festes w kommt jeder $(x_1, \ldots, x_k)$–Minterm nur in einem $f_{i,w}^{(2)}$ vor. Da es 2^s Vektoren w und 2^k Minterme gibt, genügen $2^s 2^k$ OR's.)

3.) *Konjunktion*
Berechne die $f_{i,w} = f_{i,w}^{(1)} \wedge f_{i,w}^{(2)}$
(Es genügen dazu $p2^s$ AND's.)

4.) *Disjunktion*
Berechne $f = \bigvee f_{i,w}$
(Es genügen dazu $p2^s - 1$ OR's.)

<u>Übungsaufgabe:</u> Analysiere die Größe des konstruierten Schaltkreises.

Für die Wahl von $k := \lceil 3 \log n \rceil$ und $s := n - \lceil 5 \log n \rceil$ gilt die Behauptung (ii). ■

3.3.3 Formeln und Schaltkreise monotoner Funktionen

Für monotone Boolesche Funktionen $f \in \mathbb{B}_n$ gilt:

Satz 3.12
(i) Für fast alle monotonen Booleschen Funktionen $f \in \mathbb{B}_n$ gilt:

$$\begin{aligned} L(f) &= \Omega\left(\frac{2^n}{\sqrt{n}\log n}\right), \\ C(f) &= \Omega\left(\frac{2^n}{n\sqrt{n}}\right). \end{aligned}$$

(ii) Für alle monotonen Booleschen Funktionen $f \in \mathbb{B}_n$ gilt:

$$C(f) = \mathrm{O}\left(\frac{2^n}{n\sqrt{n}}\right). \quad \blacksquare$$

3.4 Komplexität von Entscheidungsgraphen

3.4.1 Effiziente Darstellung mit Entscheidungsbäumen und Branching-Programmen

Definition 3.3
P_{DT} bzw. P_{BP} sei die Menge aller Folgen Boolescher Funktionen $f_n \in \mathbb{B}_n$ mit polynomialem $DT(f_n)$ bzw. $BP(f_n)$.

Satz 3.13

$$L = DSPACE(\log n) \subseteq P_{BP}. \quad \blacksquare$$

Wegen $\mathrm{NC}^1 \subseteq L$ können alle Funktionen aus NC^1 mit Branching-Programmen polynomialer Größe berechnet werden. Es gilt sogar:

Satz 3.14
$\mathrm{NC}^1 \subseteq P_{bw-BP}$*, wobei* P_{bw-BP} *die Menge aller Folgen Boolescher Funktionen bezeichnet, die von polynomialen Branching-Programmen beschränkter Breite berechnet werden können.* $\blacksquare$

Beispiele wichtiger Vertreter aus P_{BP}:

- ADD_n, $MULT_n$, DIV_n, PAR_n

- Berechnungen mit Permutationen (Multiplikation, Potenzen)
- Depth–first–search und Breadth–first–search auf ungerichteten Wurzelbäumen
- Bestimmte Grapheigenschaften (z. B. Grapherreichbarkeitsprobleme GAP1, UGAP)
- Symmetrische Funktionen

3.4.2 Unterschiede in der Berechnungskraft von Entscheidungsbäumen und Branching-Programmen

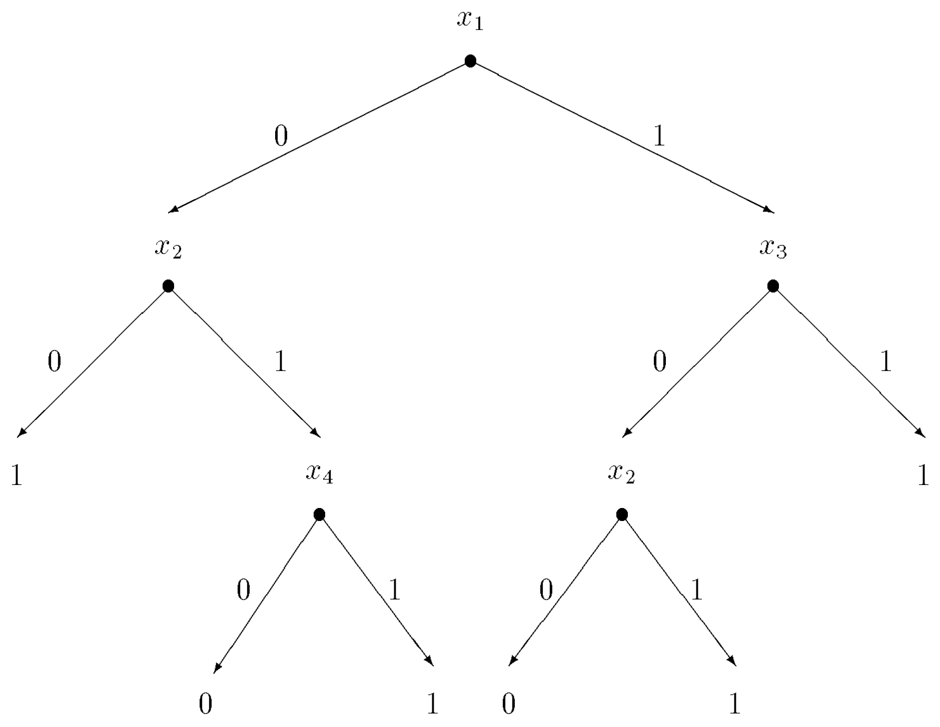
Satz 3.15
$P_{DT} \overset{\subset}{\neq} P_{BP}$.

Beweis
Offenbar gilt: $P_{DT} \subseteq P_{BP}$. Betrachte

Lemma

$$P_{DT} \subseteq P_{DNF}$$

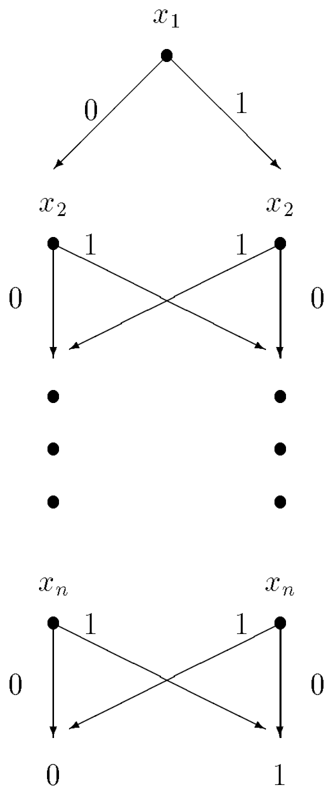
Beweis
Wir können Entscheidungsbäumen für f eine DNF für f abgewinnen:



wird beispielsweise zu $\overline{x}_1\overline{x}_2 + \overline{x}_1 x_2 x_4 + x_1 x_3 + x_1 \overline{x}_3 x_2$. Es folgt:

$$DNF(f) \leq nDT(f). \quad \square$$

Die Behauptung ergibt sich nun aus $PAR_n \notin P_{DT}$, aber $PAR_n \in P_{BP}$:



$\blacksquare$

3.4.3 Typische Größe von Branching-Programmen

Satz 3.16
(i) *Für fast alle Booleschen Funktionen $f \in \mathbb{B}_n$ gilt:*

$$BP(f) = \Omega\left(\frac{2^n}{n}\right).$$

(ii) *Für alle Booleschen Funktionen $f \in \mathbb{B}_n$ gilt:*

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

Beweis
(i) Wir werden im nächsten Abschnitt zeigen, daß

$$BP(f) \geq \frac{1}{3}C(f)$$

gilt. Damit folgt die Behauptung aus Satz 3.11.
(ii) Übungsaufgabe (Hinweis: Verwende Lupanowsche (k,s)–Repräsentation.) ∎

Für read–once–only Branching-Programme (BP_1)gilt:

Satz 3.17
(i) *Für fast alle Booleschen Funktionen $f \in \mathbb{B}_n$ gilt:*

$$BP_1(f) = \Omega\left(\frac{2^n}{n}\right).$$

(ii) *Für alle Booleschen Funktionen $f \in \mathbb{B}_n$ gilt:*

$$BP_1(f) = O\left(\frac{2^n}{n}\right). \quad \blacksquare$$

3.5 Beziehungen zwischen den verschiedenen Darstellungen

Satz 3.18
Für alle Booleschen Funktionen $f \in \mathbb{B}_n$ gilt:

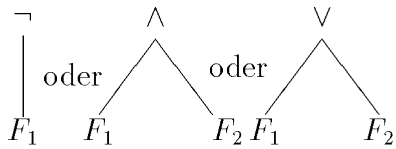
$$nDT(f) \underset{(i)}{\geq} \left\{ \begin{array}{c} DNF(f) \\ KNF(f) \\ PNF(f) \end{array} \right\} \underset{(ii)}{\geq} L_{\{\vee,\wedge,\neg\}} \underset{(iii)}{\geq} BP(f) \underset{(iv)}{\geq} \tfrac{1}{3}C(f)$$

Beweis
(i) $DT(f) \geq DNF(f)$ ist bereits gezeigt.
Noch zu zeigen: $DT(f) \geq KNF(f)$, $DT(f) \geq PNF(f)$. Übungsaufgabe.

(ii) DNF, KNF, PNF sind Formeln der Tiefe 2.

(iii) *Vollständige Induktion über die Formelgröße $l := L_{\{\vee,\wedge,\neg\}}$:*
$l = 0$: klar.
$(l-1) \rightarrow l$:

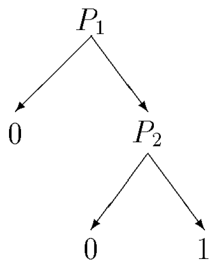
Sei F optimale Formel für f. Dann gilt

$T =$ 

wobei F_1, F_2 Formeln der Tiefe $l-1$ sind.
Nach Induktionsvoraussetzung können wir F_1 und F_2 durch Branching-Programme P_1, P_2 mit Größe(P_i) $\leq$Größe(F_i) berechnen lassen.

<u>1. Fall:</u> $F = \neg F_1$.
Betrachte das Branching-Programm $\tilde{P}_1$ mit gegenüber P_1 vertauschten 0– und 1–Senke(n). $\tilde{P}_1$ berechnet F. Größe$(\tilde{P}_1)$ =Größe(P_1) $\leq$Größe(F_1) =Größe$(F) - 1$.

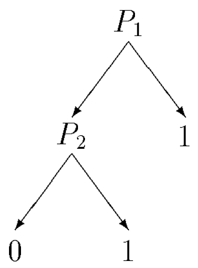
<u>2. Fall:</u> $F = F_1 \wedge F_2$.
Betrachte das folgende Branching-Programm P:



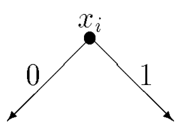
Offenbar berechnen P und F dieselbe Funktion.
Es gilt:

$$\text{Größe}(P) = \text{Größe}(P_1) + \text{Größe}(P_2) \leq \text{Größe}(F_1) + \text{Größe}(F_2) = \text{Größe}(F) - 1.$$

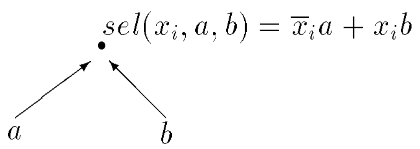
<u>3. Fall:</u> $F = F_1 \vee F_2$.
Analoge Betrachtung für P:



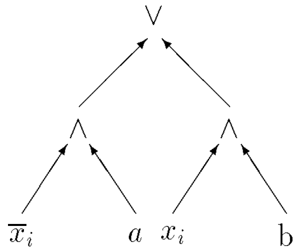
(iv) Sei P ein optimales Branching-Programm für f, d.h. $BP(f) =$ Größe(P).
Ersetze alle Knoten



durch Knoten



Wir erhalten einen $(\mathbb{B}_2 \cup \{sel\})$–Schaltkreis C' für f.
Weiter können wir die sel–Knoten von C' ersetzen durch



Damit erhalten wir einen Schaltkreis C für f der Größe

$$\text{Größe}(C) \leq 3\text{Größe}(C') = 3\text{Größe}(P). \quad \blacksquare$$

Man kann insbesondere bei den weniger kompakten Darstellungsformen explizit Effizienzunterschiede für einzelne Funktionen nachweisen. Dies ist besonders für Entscheidungen zur Auswahl konkreter Darstellungsformen von Bedeutung.

Satz 3.19
Es existiert je eine Boolesche Funktion $f \in \mathbb{B}_n$ mit:

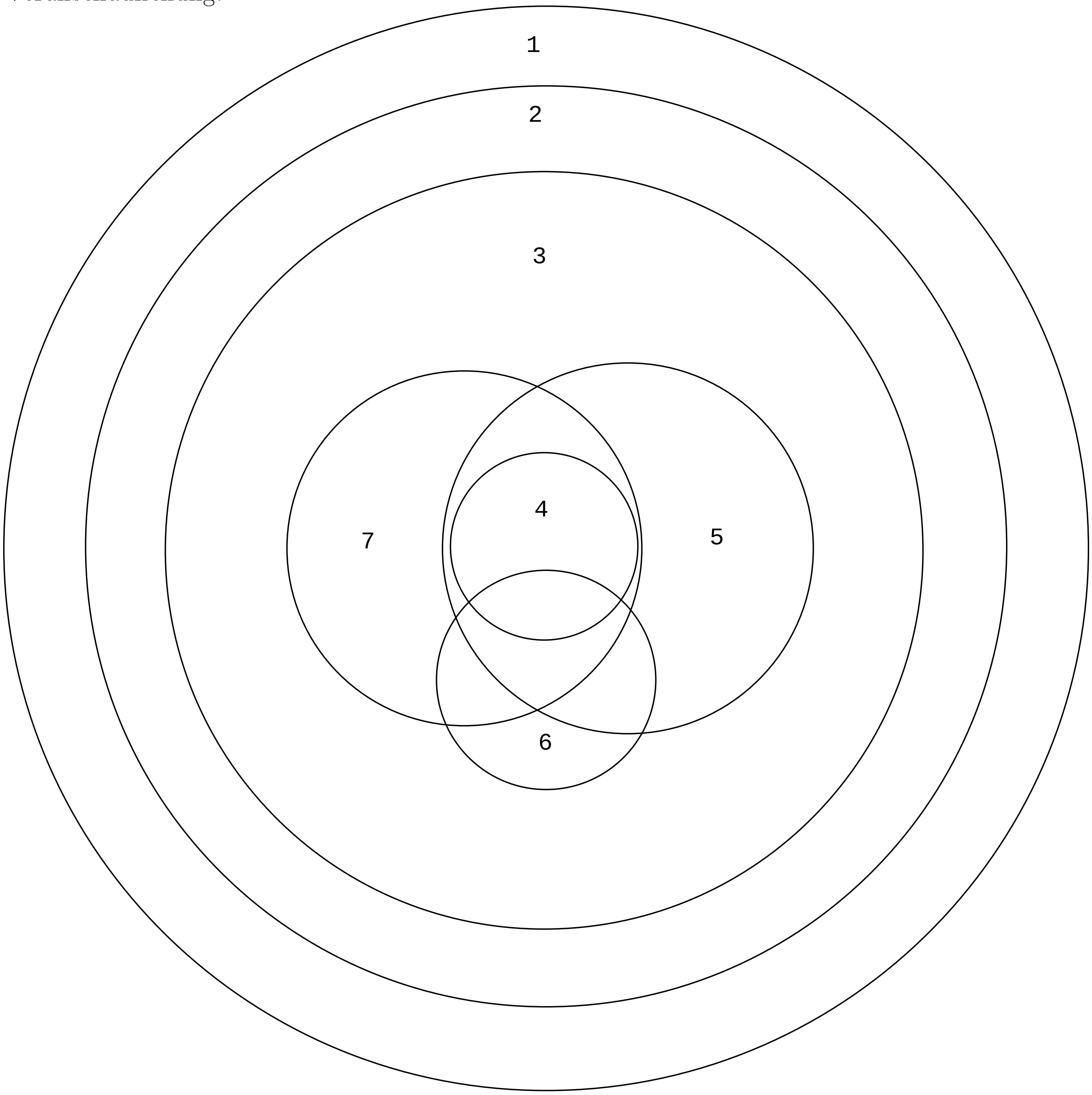
(*i*)
$$\begin{aligned} DT(f) &= 2^{\Omega(n)} \quad \textit{und} \quad KNF(f) = n^{o(1)} \\ DT(f) &= 2^{\Omega(n)} \quad \textit{und} \quad DNF(f) = n^{o(1)} \\ DT(f) &= 2^{\Omega(n)} \quad \textit{und} \quad PNF(f) = n^{o(1)} \end{aligned}$$

(*ii*)
$$\begin{aligned} DT(f) &= 2^{\Omega(n)} \quad \textit{und} \quad BDD(f) = \mathrm{O}(n) \\ BDD(f) &= 2^{\Omega(n/\log n)} \quad \textit{und} \quad DT(f) = n^{o(1)} \end{aligned}$$

(*iii*)
$$\begin{aligned} DNF(f) &= 2^{\Omega(n)} \quad \textit{und} \quad L(f) = n^{o(1)} \\ KNF(f) &= 2^{\Omega(n)} \quad \textit{und} \quad L(f) = n^{o(1)} \\ PNF(f) &= 2^{\Omega(n)} \quad \textit{und} \quad L(f) = n^{o(1)} \end{aligned}$$

Beweis
Originalarbeiten von Meinel, Damm/Meinel, Breitbart/Hunt/Rosenkrantz. ■

Veranschaulichung:



$1 = P_{\mathrm{SK}} = \mathbf{P}$
$2 = P_{BP} = L$
$3 = P_{\mathrm{F}} = \mathrm{NC}^1$
$4 = P_{DT}$
$5 = P_{KNF}$
$6 = P_{BDD}$
$7 = P_{DNF}$

Kapitel 4

Zweistufige Logikminimierung

4.1 Primimplikanten und Minimalpolynome

Definition 4.1
Sei $f \in \mathbb{B}_n$.
(i) Ein Monom m heißt *Implikant* von f genau dann, wenn gilt:

$$m(a) = 1,\ a \in \{0,1\}^n,\ \text{impliziert}\ f(a) = 1,$$

d. h. $m(a) \leq f(a)$ für alle $a \in \{0,1\}^n$.
(ii) Ein Implikant m heißt *Primimplikant* von f genau dann, wenn keine echte Verkürzung von m Implikant von f ist.

Bezeichnung

$I(f)$	sei die Menge der Implikanten von f.
$PI(f)$	sei die Menge der Primimplikanten von f.

Primimplikanten sind wichtig bei der Erzeugung von Minimalpolynomen für f. (Minimalpolynome liefern minimale PLAs!)

Satz 4.1
Minimalpolynome für f enthalten nur Primimplikanten von f.

Beweis
Offenbar enthält jedes Polynom für f nur Implikanten von f.
Sei $p = m_1 \vee \ldots \vee m_k$ Minimalpolynom für f.
Annahme: $m_1 \notin PI(f)$.
Es existiert ein echtes Teilmonom m_1' von m_1, das Implikant von f ist.

Da $m_1' \vee m_1 = m_1'$ und da für einen Implikanten m von f auch $m \vee p$ Polynom für f ist, liefert

$$m_1' \vee m_2 \vee \ldots \vee m_k = p'$$

ein Polynom für f, das kürzer ist als das Minimalpolynom p ↯. ■

Strategie zum Auffinden eines Minimalpolynoms für f:

(i) Bestimme $PI(f)$.

(ii) Suche unter allen Summen von Primimplikanten von f, die f berechnen, die billigste Summe.

4.2 Berechnung aller Primimplikanten

4.2.1 Eingabe einer Funktionstabelle (=kDNF)

Sei $f \in \mathbb{B}_n$ in Tabellenform (=kDNF) gegeben.

Quine/McCluskey–Verfahren:

Idee
Bestimme sukzessive alle Implikanten der Länge $n, n-1, \ldots$ und sortiere die Primimplikanten heraus.

Realisierung
Sei $I_k(f)$ die Menge aller Implikanten von f der Länge k. Sei $PI_k(f)$ die Menge aller Primimplikanten von f der Länge k.

<u>Offenbar gilt:</u>

$$I_n(f) = \{m_a | m_a \text{ ist Minterm von } f\} \quad \text{(d. h. } f(a) = 1).$$

Wir können Implikanten auf folgende Weise verkürzen:

Lemma 4.1 (Einfacher Konsensus)
Sei m ein Monom, das weder x_k noch $\overline{x}_k$ enthält. Dann gilt:

$$m \in I(f) \iff mx_k, m\overline{x}_k \in I(f).$$

Beweis

$$(\Longrightarrow) \qquad (mx_k)(a) = 1 \implies m(a) = 1$$
$$\implies f(a) = 1, \qquad \text{da } m \in I(f).$$

Analog folgt aus $(m\overline{x}_k)(a) = 1 \quad f(a) = 1$.

$$(\Longleftarrow) \qquad m(a) = 1 \implies (mx_k)(a) = 1 \quad (\text{falls } a_k = 1)$$
$$\text{oder } (m\overline{x}_k)(a) = 1 \quad (\text{falls } a_k = 0)$$

Wegen $mx_k, m\overline{x}_k \in I(f)$ folgt in beiden Fällen $f(a) = 1$, also $m \in I(f)$. ■

Algorithmus 4.1 (Quine)
Eingabe: Funktionstabelle von f.
Ausgabe: $PI(f)$.

$PI(f) := \emptyset$

1. *Berechne aus der Funktionstabelle $I_n(f)$.*
   (Menge aller Minterme m_a von f, also $f(a) = 1$)
   Setze $i := n$.

2. *Solange $I_i \neq \emptyset$:*
   $i := i - 1;$
   $I_i(f) := \{m | \exists k : x_k, \overline{x}_k \notin m$ *und* $mx_k, m\overline{x}_k \in I_{i+1}(f)\};$
   $PI_{i+1}(f) := \{m \in I_{i+1}(f) |$ *es existiert keine Verkürzung von* m *in* $I_i(f)\};$
   $PI(f) := PI(f) \cup PI_{i+1}(f).$

Analyse:
Zur Berechnung von $I_i(f)$ müssen alle Paare $m', m'' \in I_{i+1}(f)$ überprüft werden, ob sie sich als $mx_k, m\overline{x}_k$ schreiben lassen.
Es existieren $\#I_{i+1}(f) * \#(I_{i+1}(f) - 1)/2$ verschiedene Paare.
Man kann aber die zu überprüfenden Paare geschickt auswählen:
Sei $I_{i,l}(f)$ die Menge der Implikanten von f der Länge i mit genau l negativen Literalen.
Für $m' \in I_{i+1,l}$ muß man nur für jedes j mit $\overline{x}_j \in m'$ prüfen, ob

$$m''_j := \{x_j\} \cup \{x_i^{\epsilon_i} | x_i^{\epsilon_i} \in m' \text{ und } {}_i \neq j\} \in I_{i+1,l-1}.$$

Für jedes $m' \in I_{i+1,l}(f)$ werden also höchstens $l \leq i + 1 \leq n$ andere Monome überprüft, so daß die Zahl der zu untersuchenden Paare $\leq \#I_{i+1} * n$ ist.

Beispiel 4.1
Betrachte $f = \overline{x}_1\overline{x}_2x_3\overline{x}_4 + \overline{x}_1\overline{x}_2x_3x_4 + \overline{x}_1x_2\overline{x}_3\overline{x}_4 + x_1\overline{x}_2x_3\overline{x}_4$.

$I_{4,4}(f) = \emptyset$
$I_{4,3}(f) = \{\overline{x}_1\overline{x}_2x_3\overline{x}_4, \overline{x}_1x_2\overline{x}_3\overline{x}_4\}$
$I_{4,2}(f) = \{\overline{x}_1\overline{x}_2x_3x_4, x_1\overline{x}_2x_3\overline{x}_4\}$
$I_{4,1}(f) = \emptyset$
$I_{4,0}(f) = \emptyset$

$I_{3,3}(f) = \emptyset$
$I_{3,2}(f) = \{\overline{x}_1\overline{x}_2x_3, \overline{x}_2x_3\overline{x}_4\}$
$I_{3,1}(f) = \emptyset$
$I_{3,0}(f) = \emptyset$ $\qquad PI_4(f) = \{\overline{x}_1x_2\overline{x}_3\overline{x}_4\}$

$I_{2,2}(f) = \emptyset$
$I_{2,1}(f) = \emptyset$
$I_{2,0}(f) = \emptyset$ $\qquad PI_3(f) = \{\overline{x}_1\overline{x}_2x_3, \overline{x}_2x_3\overline{x}_4\}$

Es gilt: $PI(f) = \{\overline{x}_1\overline{x}_2x_3, \overline{x}_2x_3\overline{x}_4, \overline{x}_1x_2\overline{x}_3\overline{x}_4\}$.

Bemerkung
Da Monome als Worte über einem 3–elementigen Alphabet aufgefaßt werden können, lassen sich die $I_{r,s}(f)$ als balancierte Bäume verwalten (z. B. als 2–3–Bäume).
Der Zeitaufwand zum Suchen und Einfügen ist $\mathrm{O}(\log(\#I_{r+1,s}(f)))$.

Analyse des verbesserten Algorithmus' (Quine/McCluskey):
Es existieren $\leq 3^n$ Monome.
Für jedes $m \in I_{i+1,l}(f)$ müssen $\leq n$ Monome in $I_{i+1,l-1}$ gesucht werden.
$\mathrm{O}(n)$ Schritte genügen, um ein Monom in $I_{i+1,l-1}$ zu suchen und in $I_{i,l-1}$ einzufügen.
Folglich ist die Rechenzeit $\mathrm{O}(n^2 3^n)$ in der Zahl der Variablen bzw.
$\mathrm{O}(N^{\log 3}\log^2 N)$ in der Eingabelänge $N = 2^n$.

Satz 4.2
Der Quine/McCluskey–Algorithmus berechnet aus der Funktionstabelle einer Booleschen Funktion $f \in \mathbb{B}_n$ die Primimplikantenmenge PI(f).
Die Eingabelänge ist $N = 2^n$.
Die maximale Ausgabelänge ist $\Omega(3^n) = \Omega(N^{\log 3})$. $\quad (*)$
Die worst–case Rechenzeit ist $\mathrm{O}(3^n n^2) = \mathrm{O}(N^{\log 3}\log^2 N)$.

Beweis
Lediglich $(*)$ ist noch offen. Übungsaufgabe. ■

Bemerkung
Es existiert eine Boolesche Funktion $f \in \mathbb{B}_n$ mit $\#PI(f) = \Omega(\frac{3^n}{n})$ und Länge der Ausgabe $= \Theta(3^n) = \Theta(N^{\log 3})$.
Unter diesem Gesichtspunkt ist die Laufzeit von Quine/McCluskey sehr gut!

4.2.2 Eingabe einer DNF

Sei $f \in \mathbb{B}_n$ als DNF gegeben.

Satz 4.3
Es existieren Funktionen f, die als Summe von k Monomen dargestellt werden können, mit

$$\#PI(f) = 2^k - 1.$$

Beweis
Definiere $f_k \in \mathbb{B}_{2k-1}$ induktiv:

<u>k=1:</u>

$$f_1(x_1) = x_1$$

(f_1 kann durch einen Primimplikanten dargestellt werden).

<u>$k \to k+1$:</u>

$$f_{k+1}(x_1, \ldots, x_{2k+1}) = x_{2k} f_k(x_1, \ldots, x_{2k-1}) + \overline{x}_{2k} x_{2k+1}.$$

Da f_k aus k Monomen besteht, ist f_{k+1} Polynom aus $k+1$ Monomen.
Nach Induktionsvoraussetzung hat $f_k \quad (2^k - 1)$ PI.
Da $\overline{x}_{2k} x_{2k+1} \in PI(f_{k+1})$, genügt es zu zeigen, daß $m \in PI(f_k) \quad x_{2k}m, x_{2k+1}m \in PI(f_{k+1})$ impliziert und folglich $\#PI(f_{k+1}) \geq 2(2^k - 1) + 1 = 2^{k+1} - 1$.
Offenbar ist $x_{2k}m \in I(f_{k+1})$.
Weiter gilt $x_{2k+1}m \in I(f_{k+1})$, denn:
Sei $x_{2k+1}m(a) = 1$. Dann gilt:

$$\begin{aligned} a_{2k} = 0 &\Longrightarrow \overline{x}_{2k} x_{2k+1}(a) = 1 &&\Longrightarrow f_{k+1}(a) = 1 \\ a_{2k} = 1 &\Longrightarrow x_{2k} m(a) = 1 &&\Longrightarrow f_{k+1}(a) = 1 \end{aligned}$$

Untersuche echte Verkürzungen von $x_{2k}m, x_{2k+1}m$:
Offenbar ist $m \notin I(f_{k+1})$ (Z. B. existieren Eingaben a mit $a_{2k} = a_{2k+1} = 0$ und damit $f_{k+1}(a) = 0$, aber $m(a) = 1$). Sei m' echte Verkürzung von m.

Annahme: $x_{2k}m' \in I(f_{k+1})$
Dann gilt $m' \in I(f_k) \lightning\ m \in PI(f_k)$.

Annahme: $x_{2k+1}m' \in I(f_{k+1})$
Dann folgt ebenfalls $m' \in I(f_k) \lightning\ m \in PI(f_k)$, denn:
Sei $a' \in \{0,1\}^{2k-1}$ mit $m'(a') = 1$.
Betrachte $a = (a', 1, 1)$.
Dann gilt

$$x_{2k+1}m'(a) = 1$$

und damit

$$f_{k+1}(a) = 1,$$

da $x_{2k+1}m' \in I(f_{k+1})$. Wegen $\overline{x}_{2k}x_{2k+1}(a) = 0$ folgt $f_k(a') = 1$, also $m' \in I(f_k)$.

Insgesamt gilt also $x_{2k}m, x_{2k+1}m \in PI(f_k)$ und damit $\#PI(f) = 2^k - 1$. ∎

Bemerkung
Ist umgekehrt f darstellbar durch k Monome, dann hat f höchstens $2^k - 1$ PI.

Schlußfolgerung:
Ausgehend von einer DNF–Darstellung von f gibt es zur Berechnung von $PI(f)$ nur Algorithmen mit exponentieller Laufzeit.
(Allein die Ausgabelänge ist schon exponentiell in der Eingabelänge!)

Baummethode

Idee:
Überführe $f \in \mathbb{B}_n$ in Funktionen $f_i \in \mathbb{B}_{n'}$ mit $n' < n$ und löse das Problem für die f_i:
Sei $f \in \mathbb{B}_n$.
Definiere $f_{i,c} \in \mathbb{B}_n, \quad 1 \le i \le n, \quad c \in \{0,1\}$ als

$$f_{i,c}(x_1, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, c, x_{i+1}, \ldots, x_n)$$

$f_{i,c}$ ist unabhängig von x_i auffaßbar als Funktion in $\mathbb{B}_{n-1}$.

Betrachte die Shannon–Zerlegung:

$$f(x) = \overline{x}_i f_{i,0} \vee x_i f_{i,1}.$$

Satz 4.4

(i) Gilt $m_0 \in I(f_{i,0})$, $m_1 \in I(f_{i,1})$, dann sind $\overline{x}_1 m_0, x_i m_1, m_0 m_1 \in I(f)$.

(ii) Gilt $m \in PI(f)$, $m = x_i m_1$, dann ist $m_1 \in PI(f_{i,1})$.

(iii) Gilt $m \in PI(f)$, $m = \overline{x}_i m_0$, dann ist $m_0 \in PI(f_{i,0})$.

(iv) Gilt $m \in PI(f)$, $x_i, \overline{x}_i \notin m$, dann existieren $m_0 \in PI(f_{i,0})$, $m_1 \in PI(f_{i,1})$ mit $m = m_0 m_1$.

Beweis
Übungsaufgabe. ∎

Wir können nun $PI(f_{i,0}), PI(f_{i,1})$ berechnen und daraus $PI(f)$ gewinnen:

Algorithmus 4.2 (Baummethode)
Wir vereinbaren zunächst, daß alle entstehenden Polynome so vereinfacht werden, daß keine

- Summanden 0
- unverkürzte Monome (z.B. $x_1 x_1 x_2$)
- mehrfach auftretende Monome
- Verlängerungen anderer Monome
- Widersprüche (z.B. $x_1 \overline{x_1} x_2$)

auftreten.
Ist p ein Polynom für f, dann bezeichne p^* das Polynom aus allen PI von f.

Eingabe: DNF p für f
Ausgabe: p^*

1. Falls p nach Vereinfachung 0 oder 1 ist, gib Konstante aus und beende den Algorithmus.
2. Ansonsten wähle i mit: x_i oder $\overline{x}_i$ kommt in p vor.
3. Bilde $p_{i,c}$, $c = 0, 1$, durch formales Ersetzen von x_i durch c in p.
   Berechne rekursiv $p^*_{i,c}$ für $c = 0, 1$.
4. Bilde $p' = \overline{x}_i p^*_{i,0} \vee x_i p^*_{i,1} \vee p^*_{i,0} p^*_{i,1}$.
5. Bilde p^* durch Ausmultiplizieren und Vereinfachen.

Satz 4.5
Algorithmus 4.2 berechnet $PI(f)$.

Beweis
<u>Endlichkeit:</u>
Der Algorithmus ist endlich, da die Zahl der vorkommenden Variablen sinkt und jeder Teilschritt endlich ist.

<u>Korrektheit:</u>
Wir führen den Korrektheitsbeweis durch Induktion über die Zahl n der Indizes i, für die Literale x_i in p vorkommen.
<u>$n = 0$:</u>
trivial.
<u>$(n-1) \to n$:</u>
Es komme ein Literal x_i in p vor.

$$PI(f_{i,0}) = p^*_{i,0},\ PI(f_{i,1}) = p^*_{i,1} \quad \text{nach Induktionsvoraussetzung.}$$

Vermöge der Shannon–Entwicklung ist $p'' = (\overline{x}_i p^*_{i,0} \vee x_i p^*_{i,1})$ Polynom für f.
Nach Satz 4.4 enthält $p^*_{i,0} p^*_{i,1}$ nur Implikanten von f, so daß p' ein Polynom für f ist.

Annahme: Es existiert $m \in PI(f)$, $m \notin p'$.
Nach Satz 4.4 und der Induktionsvoraussetzung gilt

$$\begin{aligned} m = \overline{x}_i m_0 &\quad \text{und} \quad m_0 \in p^*_{i,0} \text{ oder} \\ m = x_i m_1 &\quad \text{und} \quad m_1 \in p^*_{i,1} \text{ oder} \\ m = m_0 m_1 &\quad \text{und} \quad m_0 \in p^*_{i,0},\ m_1 \in p^*_{i,0}. \end{aligned}$$

In jedem Fall existiert also ein PI, der weder in $p^*_{i.0}$ noch in $p^*_{i,1}$ vorkommt $\lightning$.
Es gilt $p^* = PI(f)$, da PI nicht durch Verkürzungen wegfallen, aber alle anderen Implikanten wegfallen lassen. ■

Satz 4.6
Es existieren Funktionen $f_n \in \mathbb{B}_n$, $n = m^2$, für die der Algorithmus 4.2 bei Eingabe des Polynoms p^* aus allen m Primimplikanten und bei beliebiger Wahl der Zerlegungsvariablen mindestens 2^m Schritte benötigt. ■

4.2.3 Methode des iterierten Konsensus

Definition 4.2
Seien $m\overline{x}_i$, $m'x_i \in I(f)$, wobei m und m' weder x_i noch $\overline{x}_i$ enthalten. Gilt $mm' \neq 0$, so heißt mm' *Konsensus* von $m\overline{x}_i$ und $m'x_i$.
Gilt $m = m'$, so heißt m *einfacher Konsensus* von $m\overline{x}_i$ und $m'x_i$.

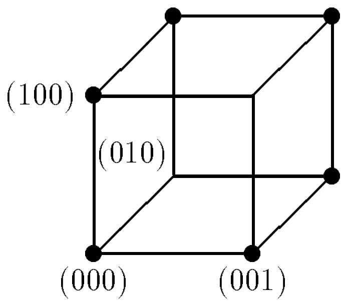
Satz 4.7
Jeder Konsensus zweier Implikanten von f ist Implikant von f. Mehr noch, es gilt:

$$m\overline{x}_i \vee m'x_i = m\overline{x}_i \vee m'x_i \vee mm'.$$

Beweis
Sei $mm'(a) = 1$.
Aus $a_i = 0$ folgt $m\overline{x}_i(a) = 1$.
Aus $a_i = 1$ folgt $m'x_i(a) = 1$.
Da $m\overline{x}_i, m'x_i \in I(f)$, gilt $f(a) = 1$ in beiden Fällen, d. h. $mm' \in I(f)$. ■

Bemerkung
Da Quine/McCluskey von einer kDNF ausgeht, genügt die Betrachtung des einfachen Konsensus'.
(Allgemein ist jeder Konsensus zu betrachten.)

Beispiel 4.2



Sei $p(x) = \overline{b}\overline{c} \vee ab \vee \overline{a}c$.
In p ist kein einfacher Konsensus enthalten, aber
$p^*(x) = \overline{b}\overline{c} \vee ab \vee \overline{a}c \vee bc \vee a\overline{c} \vee \overline{a}\overline{b}$.

Algorithmus 4.3 (Methode des iterierten Konsensus)
Eingabe: DNF p für f
Ausgabe: p^*

1. Vereinfache das Polynom.

2. Solange es einen Konsensus gibt, für den noch keine Verkürzung im Polynom enthalten ist, füge diesen Konsensus hinzu und vereinfache das Polynom.

Beispiel 4.3

$$\begin{aligned}
p_1 &= \underbrace{\overline{b}\overline{c} \vee ab}_{\rightarrow a\overline{c}} \vee \overline{a}c \\
p_2 &= \underbrace{\overline{b}\overline{c}}_{\rightarrow} \vee ab \vee \underbrace{\overline{a}c}_{\overline{a}\overline{b}} \vee a\overline{c} \\
p_3 &= \overline{b}\overline{c} \vee \underbrace{ab \vee \overline{a}c}_{\rightarrow bc} \vee a\overline{c} \vee \overline{a}\overline{b} \\
p_4 &= \overline{b}\overline{c} \vee ab \vee \overline{a}c \vee a\overline{c} \vee \overline{a}\overline{b} \vee bc \qquad \text{STOP.}
\end{aligned}$$

Satz 4.8
Die Methode des iterierten Konsensus berechnet $PI(f)$ für die durch das Polynom p gegebene Funktion f.

Beweis
Endlichkeit:
Für jedes entfernte Monom m existiert eine Verkürzung m' im Polynom.
Folglich wird Monom m nie wieder zum Polynom hinzugefügt, da dieses stets eine Verkürzung (m' oder Verkürzungen davon) enthält.
Da jedes Monom höchstens einmal als Konsensus zum Polynom hinzugefügt wird und es nur endlich viele Monome gibt, bricht der Algorithmus ab.

<u>Korrektheit:</u>
Nach Satz 1.4 stellt das aktuelle Polynom stets f dar.
Es genügt zu zeigen, daß p^* alle PI enthält, da das aktuelle Polynom stets nur verkürzte Monome enthält.
Induktion über die Anzahl der Indizes i, für die x_i–Literale in p^* vorkommen:
<u>$n = 0$:</u>
Klar.
<u>$(n-1) \to n$:</u>
Sei p^* so, daß x_i–Literale oBdA für $i \in \{1, \ldots, n\}$ vorkommen.
Sei $m \in PI(f)$. Zeige, daß m in p^* vorkommt.
<u>1. Fall:</u> $m \neq 1$:
Dann enthält m ein Literal, oBdA x_n;
D. h. $m = x_n m^*$. Wegen $m \in PI(f)$ gilt $m^* \in PI(f_{n,1})$.
Sei $p^* = \overline{x}_n p_1 \vee x_n p_2 \vee p_3$ (x_n bzw. $\overline{x}_n$ ausklammern).
Dann ist $p_2 \vee p_3$ ein Polynom für $f_{n,1}$. In $p_2 \vee p_3$ kommen $x_n, \overline{x}_n$ nicht vor.
<u>Annahme:</u> Der Algorithmus stoppt bei Eingabe $p_2 \vee p_3$ nicht im ersten Schritt.
Dann existiert ein Konsensus $m'm''$ in $p_2 \vee p_3$ aus $x_i m'$ und $\overline{x}_i m''$, für den es keine Verkürzung von $m'm''$ in $p_2 \vee p_3$ gibt.
Setze

$$M' := \begin{cases} x_n m' & \text{falls } x_i m' \text{ in } p_2 \text{ vorkommt} \\ m' & \text{sonst} \end{cases}$$

und

$$M'' := \begin{cases} x_n m'' & \text{falls } \overline{x}_i m'' \text{ in } p_2 \text{ vorkommt} \\ m'' & \text{sonst} \end{cases} .$$

Dann ist $M'M''$ Konsensus in p^* von $x_i M'$ und $\overline{x}_i M''$.
Da der Algorithmus bei p^* stoppt, existiert in p^* eine echte Verkürzung von $M'M''$. Wegen $\overline{x}_n \notin M'M''$ hat $m'm''$ eine Verkürzung in $p_2 \vee p_3$ $\lightning$.
Also stoppt der Algorithmus bei Eingabe $p_2 \vee p_3$ nach dem ersten Schritt.
Wir wissen nun, daß m^* ein Monom in p_2 ist und folglich ist $x_n m^* = m$ ein Monom in p^*.

<u>2. Fall:</u> $m = 1$ und folglich $m^* = 1$.
Sei oBdA $x_n \in p^*$.
Wegen $f_{1,0} = f_{1,1} = 1$ kommt 1 sowohl in $p_1 \vee p_3$ als auch in $p_2 \vee p_3$ vor.
Gilt $p_3 = 1$, dann ist $p^* = 1$. Gilt $p_3 \neq 1$, dann ist $p_1 = p_2 = 1$.
Damit enthält p^* den Konsensus 1 aus $\overline{x}_n$ und x_n. ■

Satz 4.9
Es gibt Funktionen $g_k \in \mathbb{B}_{2k}$, für die die Methode des iterierten Konsensus bei Eingabe eines Polynoms aus (k+2) Monomen für g_k zur Berechnung des einzigen PI mehr als 2^k Schritte benötigt. ■

4.3 Berechnung von Minimalpolynomen

Eingabe einer Funktionstabelle (=kDNF)

Sei p^* und $f \in \mathbb{B}_n$ in kDNF gegeben.

Definition 4.3
Die *Primimplikantentafel* $M_{PI(f)}$ (PI–Tafel) von f ist eine 0–1–Matrix, die für jeden PI $m_1^*, \ldots, m_k^*$ von f eine Zeile und für jeden Minterm $m_{a_1}, \ldots m_{a_l}$ der kDNF eine Spalte hat mit

$$M_{PI(f)}(i,j) = m_i^*(a_j).$$

Bezeichnung
Gilt $m_i(a_j) = 1$, so *überdeckt* m_i^* den Minterm m_{a_j} bzw. die Eingabe a_j.

Bemerkung
$p = m_{i_1}^* \vee \ldots \vee m_{i_r}^*$ stellt f dar genau dann, wenn jede Eingabe a_j von wenigstens einem der PI $m_{i_j}^*$ überdeckt wird.
Wir sagen dann, daß p f *überdeckt.*
Ein Minimalpolynom ist eine kürzeste, f überdeckende Summe von PI.

Problem:
Finde eine kürzeste, f überdeckende Summe von PI von f.

Beispiel 4.4

	0010	0011	0100	1010
$\overline{x}_1\overline{x}_2x_3$	1	1	0	0
$\overline{x}_2x_3\overline{x}_4$	1	0	0	1
$\overline{x}_1x_2\overline{x}_3\overline{x}_4$	0	0	1	0

Wir können die PI–Tafel reduzieren:

(a) Existieren Spalten mit genau einer 1 in der i–ten Zeile, so können die i–te Zeile und alle Spalten, die in der i–ten Zeile eine 1 haben, gestrichen werden.
Denn: Spalten mit genau einer 1 können nicht anders überdeckt werden als mit dem zu der betreffenden Zeile gehörenden PI, auf den wir deshalb nicht verzichten können. Wir wählen also diesen PI aus und können seine Zeile sowie alle Spalten, die er bereits überdeckt, aus der PI–Tafel ausblenden.

(b) Existiert eine Spalte j, die an keiner Stelle Einträge enthält, die kleiner sind als die Einträge in einer festen anderen Spalte k, so kann die j–te Spalte gestrichen werden.
Denn: Wenn wir die andere Spalte mit PI i überdecken, d. h. $M_{PI(f)}(i,k) = 1$, gilt $M_{PI(f)}(i,j) \geq M_{PI(f)}(i,k) = 1$, d. h. Spalte j wird dann automatisch mitüberdeckt. Sie kann daher bereits jetzt ausgeblendet werden.

(c) Existieren Zeilen z_{i_1}, z_{i_2} mit $z_{i_1} \leq z_{i_2}$ (komponentenweise) und gilt $|m^*_{i_1}| \geq |m^*_{i_2}|$, dann kann die i_1–te Zeile gestrichen werden.

Denn: $m^*_{i_2}$ überdeckt alle Eingaben, die von $m^*_{i_1}$ überdeckt werden, und ist außerdem kürzer *(Achtung: $z_{i_1} \leq z_{i_2} \Longrightarrow |m^*_{i_1}| \geq |m^*_{i_2}|$ gilt nur vor der ersten Vereinfachung!)*; Wir werden $m^*_{i_1}$ also bestimmt nicht auswählen und können seine Zeile ausblenden.

Definition 4.4
Eine Matrix $\widetilde{M}_{PI(f)}$, die aus $M_{PI(f)}$ durch Anwendung von (a) und (b) entsteht, heißt *reduzierte PI–Tafel.* Die Primimplikanten, die zu Spalten mit genau einer 1 gehören, heißen *Kernimplikanten.*

Satz 4.10
(i) Die Kernimplikanten gehören zu jedem Minimalpolynom.
(ii) Wir können eine kürzeste, f überdeckende Summe von PI von f durch die Untersuchung der reduzierten PI–Tafel berechnen.

Beweis
(i) Sei m^*_i Kernimplikant von f.
Dann existiert eine Spalte j in der PI–Tafel, die — eventuell nach vorhergehender Vereinfachung — genau eine 1 enthält, und zwar in Zeile i.
D. h., m^*_i ist einziger PI, der die Eingabe a_j überdeckt.
Folglich muß m^*_i zu jedem Minimalpolynom von f gehören.
(ii) Nach (i) gehört jeder Kernimplikant zu jedem Minimalpoynom von f.
Die Summe der Kernimplikanten überdeckt jede Eingabe, die in einer der entsprechenden Zeilen eine 1 hat.
Wir können alle Spalten, die zu solchen Eingaben gehören, streichen (Reduktion (a)).
Reduktion (b): Da jeder Implikant, der a_k überdeckt, auch a_j überdeckt, genügt die Untersuchung der k–ten Spalte, die j–te Spalte braucht nicht betrachtet zu werden. ∎

Beispiel 4.5
Reduktion (a):

	0010	0011	0100	0101	0111	1010	1011	1110	1111
$\boxed{\overline{a}b\overline{c}}$	0	0	$\boxed{1}$	1	0	0	0	0	0
$\overline{a}bd$	0	0	0	1	1	0	0	0	0
$\boxed{\overline{b}c}$	$\boxed{1}$	1	0	0	0	1	1	0	0
cd	0	1	0	0	1	0	1	0	1
$\boxed{ac}$	0	0	0	0	0	1	1	$\boxed{1}$	1
	$\Uparrow$	$\uparrow$	$\Uparrow$	$\uparrow$		$\uparrow$	$\uparrow$	$\Uparrow$	$\uparrow$

Die mit „$\Uparrow$“ markierten Spalten enthalten nur je eine 1, so daß wir als Kernimplikanten erhalten: $\overline{a}b\overline{c}$, $\overline{b}c$ und ac.
Da die zu diesen Kernimplikanten gehörenden Zeilen außerdem noch in den mit „$\uparrow$“ markierten Spalten Einsen haben, reduziert sich die PI–Tafel auf

	0111	
$\overline{a}bd$	1	$(p_1 = \overline{a}b\overline{c} \vee \overline{b}c \vee ac \vee \overline{a}bd)$
cd	1	$(p_2 = \overline{a}b\overline{c} \vee \overline{b}c \vee ac \vee cd)$

Also $p^* = p_2 = \overline{a}b\overline{c} \vee \overline{b}c \vee ac \vee cd$.

<u>Reduktion (b):</u>

	0010	0011	0100	0101	0111	1010	1011	1110	1111
$\overline{a}b\overline{c}$	0	0	1	1	0	0	0	0	0
$\overline{a}bd$	0	0	0	1	1	0	0	0	0
$\overline{b}c$	1	1	0	0	0	1	1	0	0
cd	0	1	0	0	1	0	1	0	1
ac	0	0	0	0	0	1	1	1	1
		*		*		*	*		*

Die mit einem Stern markierten Spalten sind komponentenweise größer oder gleich einer anderen Spalte, so daß wir sie ausblenden können. Es bleibt die PI–Tafel

	0010	0100	0111	1110
$\overline{a}b\overline{c}$	0	1	0	0
$\overline{a}bd$	0	0	1	0
$\overline{b}c$	1	0	0	0
cd	0	0	1	0
ac	0	0	0	1

aus der sich auch noch recht leicht das Minimalpolynom $p^* = \overline{b}c \vee \overline{a}b\overline{c} \vee ac \vee cd$ ablesen läßt.

Beispiel 4.6
MD_6 hat 90 PI.
$\#MD_6^{-1}(1) = 50$.
Folglich ist die PI–Tafel $M = M_{PI(MD_6)}$ eine 90×50–Matrix.
M ist reduziert, aber das Minimalpolynom enthält 15 PI!
Die Ermittlung des Minimalpolynoms durch Reduktion einer PI–Tafel ist also schwierig.

Das Problem der Bestimmung eines Minimalpolynoms aus der reduzierten PI–Tafel kann auf ein NP–vollständiges Problem „reduziert“ werden:

Definition 4.5
Seien X, Y endliche Mengen, $A : X \times Y \to \{0,1\}$ eine Abbildung mit:
(i) Für alle $y \in Y$ existiert $x \in X$: $A(x,y) = 1$. (D. h. in jeder Spalte ex. eine 1.)
(ii) Für alle $x \in X$ existiert $y \in Y$: $A(x,y) = 1$. (D. h. in jeder Zeile ex. eine 1.)
$Z \subseteq X$ heißt *Überdeckung* von Y bezüglich A genau dann, wenn für alle $y \in Y$ ein $z \in Z$ existiert mit $A(z,y) = 1$.
Z heißt *kürzeste Überdeckung* von Y bezüglich A genau dann, wenn Z eine Überdeckung von Y bezüglich A ist und $\#Z \leq \#Z'$ für alle Überdeckungen Z' von Y bezüglich A.

Satz 4.11 („kleinste–Überdeckung–Problem“)
Das Problem, eine kürzeste Überdeckung einer Menge Y bezüglich einer Matrix $A : X \times Y \to \{0,1\}$ zu finden, ist NP–vollständig. ■

Betrachte das Überdeckungsproblem für reduzierte Martrizen:
A heißt *reduziert* genau dann, wenn

(i) jede Spalte wenigstens zwei Einsen enthält,

(ii) jede Zeile wenigstens eine Eins enthält,

(iii) keine zwei Spalten j, j' existieren mit $A(\cdot, j) \leq A(\cdot, j')$.

Korollar 4.1
Das kleinste–Überdeckung–Problem für reduzierte Matrizen ist ebenfalls NP–vollständig.

Beweis
Übungsaufgabe. ■

Wir können die Bestimmung eines Minimalpolynoms aus einer reduzierten PI–Tafel zurückführen auf das kleinste–Überdeckung–Problem für reduzierte Matrizen.

Satz 4.12
Zu jeder reduzierten 0–1–Matrix A existiert eine Boolesche Funktion f, deren reduzierte PI–Tafel $\widetilde{M}_{PI(f)}$ übereinstimmt mit A. ■

Algorithmus 4.4 (Naiver Probieralgorithmus)
Eingabe: $\widetilde{M}_{PI(f)}$
Ausgabe: Minimalpolynom

Überprüfe alle 2^k Teilmengen der k PI darauf, ob sie f überdecken, und wähle die billigste aus.

Es existieren Heuristiken, um den Suchraum wesentlich zu verkleinern:

Algorithmus 4.5 (Branch–and–Bound)
Subroutinen:
(i) Lower Bound:
Berechne (möglichst gute) untere Schranke L für die Länge eines Minimalpolynoms.
(ii) Upper Bound:
Berechne (möglichst gute) obere Schranke U für die Länge eines Minimalpolynoms. Gebe ein Polynom für f der Länge U an.
(iii) Branching:
Zerlege das Polynom in disjunkte Teilprobleme, die wieder vom Typ des Überdeckungsproblems sind.

(i) Subroutine Lower Bound:

1.) Sei z_i die Zahl der Eingaben, die m_i^* überdeckt.
Eine untere Schranke für die Zahl der PI im Minimalpolynom ist die kleinste Anzahl r von z_i–Werten, deren Summe $\geq \#(f^{-1}(1))$ ist.
2.) Sei c_i die Länge von m_i^*.

$$L := \text{Summe der } r \text{ kleinsten } c_i$$

(ii) Subroutine Upper Bound:

(Greedy–Algorithmus)
Wähle solange den PI, der die meisten der verbleibenden Eingaben überdeckt, hinzu, bis alle Eingaben überdeckt sind.

$$\begin{aligned} P &:= \text{Das aus diesen PI bestehende Polynom} \\ U &:= \text{Länge von } P \end{aligned}$$

(iii) Subroutine Branching:

Bilde zwei Teilprobleme durch Zerlegung an „kritischen“ PI.
Wir nehmen den PI m_i^*, der die meisten Eingaben überdeckt.

1. Teilproblem:
Suche minimales überdeckendes Polynom ohne Verwendung von m_i^*. (Exklusion von m_i^*, d. h. wir streichen die zu m_i^* gehörende Zeile)

2. Teilproblem:
Suche minimales Polynom, das m_i^* enthält. (Inklusion von m_i^*, d. h. wir streichen die zu m_i^* gehörende Zeile und alle Spalten, die durch m_i^* überdeckt werden)

Das kürzere der beiden Polynome ist das Minimalpolynom.

Branch–and–Bound Algorithmus
Eingabe: Reduzierte PI–Tafel für f
Ausgabe: Minimalpolynom

Vereinbarung:
Reduziere zunächst jedes enthaltene Überdeckungsproblem P_i und berechne Schranken L_i' und U_i' für reduzierte P_i.
Ermittle L_i und U_i, indem zu L_i' und U_i' ggf. die Längen aller durch Inklusion vorab ausgewählten PI hinzuaddiert werden.

Benutzte Datenstruktur:
BBB (Branch–and–Bound Baum): In den Knoten stehen die Teilprobleme.
Beachte: Die Probleme in den Blättern bilden eine disjunkte Zerlegung des Gesamtproblems.
Damit ist eine kleinste untere (obere) Schranke in den Blättern eine untere (obere) Schranke für das Gesamtproblem.

Algorithmus:
1.) Initialisiere BBB als Baum mit einem Knoten, der das Gesamtproblem P_0 darstellt.

2.) Berechne aktuelle Werte für L und U als Minimum aller L_i bzw. U_i der Probleme an den Blättern des BBB.

3.) Ist $L = U$, so ist das zugehörige Polynom ein Minimalpolynom. STOP.

4.) Ist $L < U$, so wird das zu L gehörende Problem P_i in zwei Teilprobleme zerlegt, die zu Söhnen von P_i im BBB werden. Gehe zurück zu 2.)

Satz 4.13
Der vorgestellte Branch–and–Bound–Algorithmus berechnet aus einer reduzierten PI–Tafel für f ein Minimalpolynom.

Beweis
Endlichkeit: Klar.
Korrektheit: Klar durch Induktion über die Zahl der PI. ■

Die Güte des Algorithmus' erklärt sich aus der Tatsache, daß nicht unnötig viele Polynome untersucht werden:

- Keine Zerlegung von Problemen P_i, für die L_i größer als U_j für ein anderes Problem P_j ist.

- Vorrangige Untersuchung von Teilproblemen mit niedrigster unterer Schranke sichert oft Rechenersparnis.

Beispiel 4.7
Seien P_1, P_2 Teilprobleme mit $L_1 = a_1, U_1 = b_1, L_2 = a_2, U_2 = b_2$ und $a_1 < a_2 < b_1 < b_2$.
Würde P_2 zuerst untersucht werden, so müßte in jedem Fall P_1 untersucht werden, da die Länge des Minimalpolynoms im Intervall $[a_2, b_2]$ liegt.
Wird P_1 zuerst untersucht, braucht P_2 nur im Falle eines ermittelten Polynoms der Länge $> a_2$ untersucht zu werden.
Vorteil: Wir können bei Erreichen einer bestimmten Schranke $U - L < \varepsilon$ abbrechen, das „richtige"Minimalpolynom ist dann höchstens um ε kürzer.

4.4 Karnaugh–Diagramme

Wir können für $n \leq 4$ ($n = 5$, $n = 6$ auch, aber eingeschränkt) $PI(f)$ und ein Minimalpolynom aus einer speziellen Tabellendarstellung ablesen:

$wx \setminus yz$	00	01	11	10
00	0	**1**	0	0
01	0	1	0	0
11	1	*1*	1	1
10	**1**	0	**1**	1

Auf dem Hyperwürfel „benachbarte" Eingaben (d. h. Eingaben, die sich nur in einer Stelle unterscheiden) sind auch in der Tabelle benachbart, wenn man sich die Ränder torusförmig geschlossen vorstellt. Monome erscheinen als Rechtecke voller Einsen mit Seitenlängen 1, 2 oder 4. Den maximalen Monomen (1–Rechtecken) entsprechen PI, hier (von oben nach unten) $\overline{w}\,\overline{y}\,z$, $x\overline{y}z$, wy, wx und $w\overline{z}$.
Kernimplikanten sind solche PI, die eine Eins als einziger PI überdecken (als „**1**" markiert), hier also $\overline{w}\,\overline{y}\,z$, wy und $w\overline{z}$.
Lediglich die als „*1*" markierte Eins wird von keinem Kernimplikanten überdeckt.
Folglich ist $p^* = \overline{w}\,\overline{y}\,z \vee wy \vee w\overline{z} \vee wx$ Minimalpolynom.

<u>n=5:</u> Zwei benachbarte Diagramme.
(<u>n=6:</u> 4 quadratisch angeordnete Diagramme.)

4.5 Partielle Boolesche Funktionen

Definition 4.6
Sei $S \subseteq \{0,1\}^n$. $f : S \to \{0,1\}^m$ heißt *partielle Boolesche Funktion* mit m Ausgängen.

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

$\widetilde{\mathbb{B}}_{n,m} := \{f | f : \{0,1\}^n \overset{\supset}{\to} \{0,1\}^m\}$.

Beispiel 4.8
Kodierung eines endlichen Automaten:
Sei $\#\{\text{Buchstaben des Eingabealphabets}\} = 20$, $\#\{\text{Zustände}\} = 10$.
Kodierung: Wir benötigen 5 Bits zur Kodierung des Eingabealphabets, 4 Bits zur Kodierung der Zustandsmenge.
Die Anzahl der möglichen Eingaben ist $2^9 = 512$. Tatsächlich kommen aber nur $10 * 20 = 200$ Eingaben vor. Der Rest ist ohne Bedeutung.

Definition 4.7
Ein Polynom p beschreibt $f \in \widetilde{\mathbb{B}_n}$ genau dann, wenn für alle a, für die $f(a)$ definiert ist, gilt: $p(a) = f(a)$.

Definition 4.8
$f' \in \widetilde{\mathbb{B}_n}$ heißt *Erweiterung* von $f \in \widetilde{\mathbb{B}_n}$ genau dann, wenn

$$def(f) \subseteq def(f') \text{ und } f(a) = f'(a) \text{ für alle } a \in def(f)$$

$f^1 \in \mathbb{B}_n$ sei Erweiterung von f mit $f^1(a) = 1$ für alle $a \notin def(f)$.
$f^0 \in \mathbb{B}_n$ sei Erweiterung von f mit $f^0(a) = 0$ für alle $a \notin def(f)$.

Satz 4.14
Minimalpolynome für $f \in \widetilde{\mathbb{B}_n}$ enthalten nur Primimplikanten von f^1.

Beweis
Sei $p = m_1 \vee \ldots \vee m_k$ Minimalpolynom für f.
p ist Minimalpolynom für eine Erweiterung $f' \in \mathbb{B}_n$ von f und enthält nur PI von f'.
Gilt $m_i(a) = 1$, dann ist $f'(a) = 1$, da $m_i \in PI(f')$, und $f^1(a) = 1$ wegen $f' \le f^1$.
Folglich ist $m_i \in I(f^1)$ und es existieren PI m_i' von f^1, die Verkürzung von m_i sind.
Damit ist $p' = m_1' \vee \ldots \vee m_k'$ ein Polynom für f. Falls $f(a) = 1$, ist $p(a) = 1$ und erst recht $p'(a) = 1$. Falls $f(a) = 0$, ist $f^1(a) = 0$ und, da $m_i' \in PI(f^1)$, $m_i'(a) = 0$, also $p'(a) = 0$. Da p Minimalpolynom, muß $m_i' = m_i$ gelten für alle i. Also enthält p nur Primimimplikanten von f^1. ■

Es genügt also, PI von f^1 zu bestimmen. Jede Überdeckung von Erweiterungen f' überdeckt alle Eingaben von f^0.
Somit genügt es, alle Eingaben von f^0 mit PI von f^1 zu überdecken, um f zu überdecken. Wir haben damit das Problem für partielle Funktionen auf das Problem für volldefinierte Funktionen zurückgeführt.

4.6 Funktionen mit mehreren Ausgängen

Benutze ein modifiziertes Kostenmaß, das Monome, die in mehreren Ausgabefunktionen vorkommen, nur einmal Kosten verursachen läßt.

Definition 4.9

(i) Ein Tupel $p = (p_1, \dots p_k)$ von Polynomen stellt $F = (f_1, \dots, f_k) \in \mathbb{B}_{n,k}$ dar, falls für alle i p_i ein Polynom für f_i ist.

(ii) Die *Größe* von p ist die Summe der Längen der verschiedenen Monome in $p_1, \dots, p_k$.

(iii) Ein Minimalpolynom für F ist ein F darstellendes Tupel von Polynomen mit minimaler Größe.

Achtung:
Minimalpolynome p_i für f_i, $1 \leq i \leq k$, bilden im allgemeinen kein Minimalpolynom $p = (p_1, \dots, p_k)$ für $f = (f_1, \dots, f_k)$.
Ein Minimalpolynom $p^* = (p_1^*, \dots, p_k^*)$ für F kann z. B. in p_i^* Implikanten von f_i enthalten, die keine PI sind:

Beispiel 4.9
Sei $F = (f_1, f_2)$ mit

$$\begin{aligned} f_1(x, y, z) &= \overline{y}z \vee xz = p_1 \\ f_2(x, y, z) &= \overline{x}y \vee yz = p_2 \end{aligned}$$

Offenbar sind die p_i Minimalpolynome für f_i.
Die Größe von $p = (p_1, p_2)$ ist 8.
Seien nun

$$\begin{aligned} p_1^* &= \overline{y}z \vee xyz \\ p_2^* &= \overline{x}y \vee xyz \end{aligned}$$

$p^* = (p_1^*, p_2^*)$ stellt F dar, aber die Größe von p^* ist 7.
Überdies ist p^* einziges Minimalpolynom von F, aber die p_i^* sind keine Minimalpolynome für f_i.

Definition 4.10
Ein Monom m heißt *multipler PI* von $F = (f_1, \ldots, f_k) \in \mathbb{B}_{n,k}$ genau dann, wenn eine Menge $I \subseteq \{1, \ldots, k\}$ existiert, so daß m PI von $\bigwedge_{i \in I} f_i$ ist.

Bez.: $MPI(F)$ sei die Menge der multiplen PI von F, $MPI_I(F) := PI\left(\bigwedge_{i \in I} f_i\right)$.

Satz 4.15
Minimalpolynome für $F \in \mathbb{B}_{n,k}$ enthalten nur multiple PI von F.

Beweis
Sei $p = (p_1, \ldots, p_k)$ Minimalpolynom von F, m ein Monom, das in p mindestens einmal vorkommt.
OBdA komme m in $p_1, \ldots, p_i$ vor.
Dann ist m Implikant von $f_1, \ldots, f_i$, d. h. $m(a) = 1$ impliziert $f_1(a) = \ldots = f_i(a) = 1$.
Dann ist m Implikant von $\bigwedge_{j=1}^{i} f_j$.
Ist m kein PI von $\bigwedge_{j=1}^{i} f_j$, dann existiert eine echte Verkürzung m' von m, die PI von $\bigwedge_{i=1}^{i} f_i$ ist.
Ersetze m in $p_1, \ldots, p_i$ durch m' und erhalte kürzere Polynome $p'_1, \ldots, p'_i$.

Behauptung:
$p' = (p'_1, \ldots, p'_i, p_{i+1}, \ldots, p_k)$ ist Polynom für F.
Es genügt zu zeigen, daß p'_j f_j darstellt.
Sei also $p'_j(a) = 0$, dann gilt $\widetilde{m}(a) = 0$ für alle Monome aus p'_j.
Insbesondere gilt $m'(a) = 0$, folglich $m(a) = 0$ und damit $p_j(a) = 0 = f(a)$.
Ist $p'_j(a) = 1$, dann existiert ein Monom $\widetilde{m}$ mit $\widetilde{m}(a) = 1$.

1. Fall: $\widetilde{m} \neq m'$.
Dann kommt $\widetilde{m}$ in p_j vor. Also $p_j(a) = 1$ und folglich $f_j(a) = 1$.

2. Fall: $\widetilde{m} = m'$.
Da m' PI von $\bigwedge_{j=1}^{i} f_j$, ist m' Implikant von f_j und somit $f_j(a) = 1$.
Also stellt $(p'_1, \ldots, p'_i, p_{i+1}, \ldots, p_k)$ f dar $\lightning$ p Minimalpolynom und länger als p'. ∎

Satz 4.16
Sei $f = f_1 \wedge \ldots \wedge f_k$, $f_i \in \mathbb{B}_n$.

(i) Für $m_i \in I(f_i)$ ist $m = m_1 \wedge \ldots \wedge m_k \in I(f)$.

(ii) Für $m \in PI(f)$ gibt es $m_i \in PI(f_i)$ mit $m = m_1 \wedge \ldots \wedge m_k$.

Beweis
Übungsaufgabe. ■

Berechnung von $MPI(F)$

1. Alle PI von f_i, $1 \leq i \leq k$, sind multiple PI für F für $I = \{i\}$.

2. Sind I', I'' disjunkt, so kann man aus $MPI_{I'}(F), MPI_{I''}(F)$ die Menge $MPI_{I' \cup I''}(f)$ bestimmen:
   Bilde für alle $m' \in MPI_{I'}(F), m'' \in MPI_{I''}(F)$ $m = m'm''$.
   Streiche m, falls $m = 0$ oder eine echte Verkürzung existiert.
   Speichere zu m die Information, daß m Implikant aller f_i, $i \in I$, ist.

Berechnung des Minimalpolynoms:
Die Auswahl einer minimalen Überdeckung von F aus $MPI(F)$ ist möglich mit einer Modifikation unseres Branch–and–Bound–Algorithmus' oben (der Funktionen mit einem Ausgang verarbeitete).
Wir haben damit die Behandlung von Funktionen mit mehreren Ausgängen auf die Behandlung von Funktionen mit einem Ausgang zurückgeführt.

4.7 Monotone und symmetrische Funktionen

4.7.1 Monotone Funktionen

Definition 4.11
Sei $f \in \mathbb{B}_n$.
f heißt *monoton*, falls

$$a \leq b \quad (a_i \leq b_i \text{ für alle } i \in \mathbb{N}) \Longrightarrow f(a) \leq f(b).$$

Bez.: $\mathbb{M}_n$ Menge aller monotonen Funktionen aus $\mathbb{B}_n$.

Satz 4.17
Sei $f \in \mathbb{M}_n$ und $m \in I(f)$ mit $m = m'\overline{x}_i$. Dann gilt: $m' \in I(f)$.

Beweis
Sei $m'(a) = 1$.
Definiere b durch:

$$b_j := \begin{cases} a_j & j \neq i \\ 0 & sonst \end{cases}$$

Es gilt $m(b) = 1$ und damit $f(b) = 1$, da $m \in I(f)$.
Wegen $b \leq a$ ist $f(a) = 1$ und damit $m' \in I(f)$. ■

Korollar 4.2
Jede monotone Funktion läßt sich durch ein Polynom ohne negative Literale darstellen.

Beweis
Ausgehend von einem beliebigen Polynom für f erhält man das gewünschte Polynom durch Streichen aller negativen Literale. ■

Definition 4.12
Ein Polynom p heißt *monoton* genau dann, wenn p keine negativen Literale enthält.

Korollar 4.3

(i) Jede monotone Funktion läßt sich durch ein monotones Polynom beschreiben.

(ii) Ist p ein monotones Polynom, so ist f_p monoton. ■

Korollar 4.4
Sei $f \in I\!M_n$. Jeder PI von f besteht nur aus positiven Literalen.

Beweis
Ansonsten wäre er nach Satz 4.17 verkürzbar. ■

Satz 4.18
Überdeckt ein monotones Polynom $p = m_1 \vee \ldots \vee m_s$ ein monotones Monom m, dann existiert ein i, $1 \leq i \leq s$, so daß m bereits von m_i überdeckt wird.

Beweis
Da p, m monoton, gilt $m(1, \ldots, 1) = 1$, $m_i(1, \ldots, 1) = 1$ für alle i.
Sei $a \in m$ mit $\sum a_i$ minimal.
Offenbar gilt: $\sum a_i =$ Länge von m.
Da p m überdeckt, existiert ein m_i mit $a \in m_i$.
Da m_i nur positive Literale enthält, überdeckt m_i alle Eingaben von m und damit m. ■

Satz 4.19
Jeder PI einer monotonen Funktion ist wesentlich (d.h. Kernimplikant).

Beweis
Sei $m \in PI(f)$.
Es ist zu zeigen, daß m Kernimplikant ist.
OBdA gelte: $m = x_1 \wedge \ldots \wedge x_i$.
Definiere a durch:

$$a_j = \begin{cases} 1 & j \leq i \\ 0 & \text{sonst} \end{cases}$$

Offensichtlich gilt $m(a) = 1$.
<u>Annahme:</u> Es existiert $m' \in PI(f)$ mit $m'(a) = 1$ und $m \neq m'$.
Nach Satz 4.18 enthält m nur positive Literale $x_j, j \leq i$.
Wegen $m \neq m'$ ist m' echte Verkürzung von m ↯ $m \in PI(f)$. ■

Korollar 4.5
Das Minimalpolynom einer monotonen Funktion ist eindeutig bestimmt. Es besteht aus allen PI. ■

Wir müssen für monotone Funktionen lediglich $PI(f)$ bestimmen, um ein Minimalpolynom zu erhalten.

Algorithmus 4.6
Eingabe: p Polynom für f
Ausgabe: p^*

1. Streiche in p aus allen Monomen die negierten Literale. Es entsteht ein Polynom p', das f darstellt.
2. Streiche alle echten Verlängerungen von vorhandenen Monomen, da alle monotonen und überflüssigen Monome vollständig von einem anderem Monom überdeckt werden.
   Es entsteht das Polynom p^*, denn da alle Variablen unnegiert sind, enthält das Ergebnis alle PI von f (Methode des iterierten Konsensus).

Verallgemeinerung der Monotonie

Definition 4.13
$F = (f_1, \ldots, f_m) \in \mathbb{B}_{n,m}$ heißt *monoton steigend (fallend) im i–ten Argument*, wenn für alle $(x_1, \ldots, x_n) \in \{0,1\}^n$ gilt:

$$F(x_1, \ldots x_{i-1}, 0, x_{i+1}, \ldots, x_n) \leq (\geq) F(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n).$$

F heißt *(allgemein) monoton*, falls F in jedem Argument monoton steigend oder fallend ist.

Achtung:
Die Definition ist nicht gleichwertig mit „F ist monoton, wenn alle f_i (allgemein) monoton sind“!

Satz 4.20
F ist genau dann monoton steigend im j–ten Argument, wenn $\overline{F} = (\overline{f}_1, \ldots, \overline{f}_m)$ fallend im j–ten Argument ist.

Beweis
Übungsaufgabe. ■

Korollar 4.6
Das Komplement einer monotonen Funktion ist monoton. ■

Definition 4.14
Ein Polynom $p = (p_1, \ldots, p_m)$ heißt im i–ten Argument *monoton steigend (fallend)*, falls $\overline{x}_i$ (x_i) in keinem seiner Monome vorkommt.
p heißt *monoton*, falls p in jedem Argument monoton steigend oder fallend ist.

Satz 4.21
Eine durch ein monotones Polynom beschriebene Funktion $F \in \mathbb{B}_{n,k}$ ist monoton.

Beweis
OBdA sei p im i–ten Argument monoton steigend.
Dann enthält kein Monom m von p ein Literal $\overline{x}_i$.
Somit verändert die Änderung der i–ten Eingabe von 0 auf 1 einzelne Funktionswerte höchstens von 0 auf 1. ■

Achtung: Die Umkehrung gilt nicht! Zumindest gilt aber:

Satz 4.22
Ist F monoton steigend (fallend) im i–ten Argument, so ist jeder multiple Primimplikant von F monoton steigend (fallend) im i–ten Argument.

Beweis
OBdA sei F monoton steigend im i–ten Argument.
Annahme: Es existiert ein multipler PI m von F mit $m = m'\overline{x}_i$. Dann existiert $J \subseteq \{1, \ldots, k\}$ mit $m \in PI(\bigwedge_{j \in J} f_j)$.
Für alle a mit $m(a) = 1$ und $a_i = 0$ gilt: $f_j(a) = 1$ für alle $j \in J$.
Da F im i–ten Argument monoton steigend ist, gilt $f_j(b) = 1$ für

$$b_k = \begin{cases} a_k & \text{für } k \neq i \\ 1 & \text{sonst.} \end{cases}$$

Folglich ist $m'(a) = m'(b) = 1$, d. h. $m' \in I(\bigwedge_{j \in J} f_j)$, und m' ist kürzer als m $\lightning$. ■

Korollar 4.7
Minimalpolynome monotoner Funktionen sind monoton. ■

Satz 4.23
Das Minimalpolynom einer monotonen Funktion $F = (f_1, \ldots, f_m) \in \mathbb{B}_{n,m}$ ist eindeutig bestimmt. Es besteht aus den Primimplikanten der einzelnen Komponenten f_i.

Beweis
Wir wissen bereits:

— Minimalpolynome von F bestehen aus multiplen Primimplikanten.

— Alle multiplen Primimplikanten sind monoton.

— Alle Primimplikanten der f_i sind für f_i wesentlich.

OBdA sei F in allen Komponenten monoton steigend.
Sei p Minimalpolynom für F.

<u>Annahme:</u> Es existiert ein PI m eines f_j, der nicht in p vorkommt.
Sei a so gewählt, daß $m(a) = 1$ und $\sum a_i$ minimal.
Dann existiert ein multipler PI $m' \in I(f_i)$ mit $m'(a) = 1$.
Da m' monoton ist, wird m von m' überdeckt $\lightning$ (Denn: m' ist kürzer als m.).
Folglich kommen alle PI der f_i in p vor.
Da alle Eingaben von diesen überdeckt werden, enthält p keine weiteren Monome. ■

Wir können nun das Minimalpolynom von F bestimmen:
1. Berechne die Primimplikanten der f_i.
2. Lasse mehrfach vorkommende PI weg.

4.7.2 Symmetrische Funktionen

Definition 4.15
(i) $f \in \mathbb{B}_n$ heißt *symmetrisch*, falls ein „Wertevektor“ $v(f) := (v_0, \ldots, v_n) \in \{0,1\}^{n+1}$ existiert mit:

$$f(a) = v_{\sum a_i}$$

Bez.: $\mathcal{S}_n$ sei die Menge aller symmetrischen Funktionen in $\mathbb{B}_n$, $\mathcal{S} := \bigcup_{n \in \mathbb{N}} \mathcal{S}_n$.
(ii) *Intervallfunktionen* $I_{k,l}$, $0 \leq k \leq l \leq n$, sind symmetrische Funktionen mit Wertevektor $v(I_{k,l}) = (v_1, \ldots, v_n)$ so, daß

$$v_i = 1 \iff k \leq i \leq l.$$

Lemma 4.2
Intervallfunktionen bilden eine Basis für alle symmetrischen Funktionen $f \in \mathcal{S}$.

Beweis
Suche in $v(f)$ maximale 1–Intervalle $[k(i), l(i)]$, $1 \leq i \leq r$, $k(i) \leq l(i) < k(i+1) - 1$ $(k(r+1) := n+2)$. Dann gilt: $f = \bigvee_{1 \leq i \leq r} I_{k(i),l(i)}$. ■

Diese Darstellung heißt *Intervalldarstellung*. Die Intervalldarstellung ist eindeutig bestimmt.

Satz 4.24

(i) $PI(I_{k,l})$ besteht aus allen Monomen mit genau k positiven und n-l negativen Literalen.

(ii) Jedes Minimalpolynom für $f \in \mathcal{S}$ ist Summe von Minimalpolynomen der Intervallfunktionen seiner Intervalldarstellung.

Beweis

(i) $m \in I(I_{k,l})$ genau dann, wenn m nicht weniger als k positive und nicht weniger als $n-l$ negative Literale enthält. Folglich ist m unverkürzbar genau dann, wenn m genau k positive und $n-l$ negative Literale enthält.

(ii) Wir zeigen, daß $m \in I(f)$ höchstens Eingaben a mit $k(i) \leq \sum a_i \leq l(i)$ für ein maximales 1–Intervall $[k(i), l(i)]$ überdecken kann.
<u>Annahme:</u> m überdecke a mit $k(i) \leq \sum a_i \leq l(i)$ und b mit $k(j) \leq \sum b_i \leq l(j)$, $i \neq j$.
Sei oBdA $l(i) < k(j) - 1$.
Da m a überdeckt, enthält m höchstens $\sum a_i$ positive Literale. Da m auch b überdeckt, enthält m höchstens $n - \sum b_i$ negative Literale.
Wir wählen die restlichen Inputs geschickt und erhalten ein c mit $\sum c_i = l(i) + 1$, das von m überdeckt wird ↯ (Denn: $v_{l(i)+1} = 0$, also $f(c) = 0$.). ■

Minimalpolynome für $I_{k,l}$ lassen sich effizient berechnen:

Ein Minimalpolynom enthält $\max\left\{\binom{n}{k}, \binom{n}{l}\right\}$ PI.
Für die Länge L dieses Minimalpolynoms gilt:

$$L = (k + n - l) \max\left\{\binom{n}{k}, \binom{n}{l}\right\}.$$

Rechenzeit zur direkten Bestimmung eines Minimalpolynoms:

$$\mathrm{O}\left((k + n - l) \max\left\{\binom{n}{k}, \binom{n}{l}\right\}\right).$$

Kapitel 5

Heuristische Verfahren zur zweistufigen Logikminimierung

5.1 Zweistufige Logikminimierung mit ESPRESSO II

Exakte Verfahren sind praktikabel höchstens für Funktionen $f \in \mathbb{B}_{n,m}$ mit $n + m \leq 15$.

Problematisch:

1. Berechnung aller PI (es existieren bis $\frac{3^n}{n}$ verschiedene PI für eine Funktion mit einem Ausgang)

2. Lösen bzw. Hinschreiben des Überdeckungsproblems.

Ideen für große n und m: (implementiert in ESPRESSO)

Gegeben: Anfangsüberdeckung
Gesucht: möglichst effiziente Überdeckung
Vorgehen: Würfel der Anfangsüberdeckung „aufklappen", in der Hoffnung, zu PI zu gelangen, ohne alle PI berechnen zu müssen. Genauer:

1. Expandiere Würfel der Anfangsüberdeckung (d. h. mache in der Überdeckung enthaltene Implikanten prim) [EXPAND]

2. Entferne alle wesentlichen PI (verkleinert das Problem) [ESS–PRIMES]

3. Mache die Überdeckung irredundant (d. h. entferne überdeckte Implikanten) [IRR–COVER]

4. Suche alternative irredundante Überdeckung. [REDUCE]

5.2 Matrixdarstellung von Implikanten und Überdeckungen

Sei $F = (f_1, \ldots, f_m)$, $f_j \in \mathbb{B}_n$.
Wir sprechen nun von (n, m)–Würfeln anstelle von Implikanten. Schreibe (n, m)–Würfel w als Zeilenvektor c_w:

$$c_w = (\underbrace{c_1, \ldots, c_n}_{E(c_w)}, \underbrace{c_{n+1}, \ldots, c_{n+m}}_{A(c_w)}).$$

$E(c_w)$:

$$c_i = \begin{cases} * & \overline{x}_i, x_i \notin w \\ 0 & \overline{x}_i \in w \\ 1 & x_i \in w \end{cases} \qquad (i = 1, \ldots, n)$$

$A(c_w)$:

$$c_{n+j} = \begin{cases} T & \text{falls } w \text{ in der Darstellung von } f_j \text{ vorkommt} \\ F & \text{sonst} \end{cases} \qquad (j = 1, \ldots, m)$$

Wir nennen $E(c_w)$ den *Eingabeteil* und $A(c_w)$ den *Ausgabeteil.*

Wir können nun Polynome P als Matrizen M_P schreiben. Im folgenden sagen wir für Matrix M_P einfach Polynom P.

Beispiel 5.1
$F = (f_1, f_2)$, $f_j \in \mathbb{B}_3$ mit $f_1 = x_1\overline{x}_2 + x_3 = p_1$ und $f_2 = \overline{x}_1 x_3 + x_2\overline{x}_3 = p_2$.

$$M_{p_1} = \begin{pmatrix} 1 & 0 & * & T & F \\ * & * & 1 & T & F \end{pmatrix} \qquad M_{p_2} = \begin{pmatrix} 0 & * & 1 & F & T \\ * & 1 & 0 & F & T \end{pmatrix}$$

Definition 5.1
Seien c, d zwei (n, m)–Würfel.
$c \subseteq d, c \not\subseteq d, c \subset d, c \cap d$ sind komponentenweise gemäß den folgenden Tabellen definiert:

$\subseteq, \subset, \not\subseteq$:

		d_i: 0	1	*
c_i	0	$\subseteq$	$\not\subseteq$	$\subset$
	1	$\not\subseteq$	$\subseteq$	$\subset$
	*	$\not\subseteq$	$\not\subseteq$	$\subseteq$

$(1 \leq i \leq n)$

		d_j: F	T
c_j	F	$\subseteq$	$\subset$
	T	$\not\subseteq$	$\subseteq$

$(n + 1 \leq j \leq n + m)$

$\cap$:

$$\begin{array}{cc|ccc} & & & d_i & \\ & & 0 & 1 & * \\ \hline & 0 & 0 & \emptyset & 0 \\ c_i & 1 & \emptyset & 1 & 1 \\ & * & 0 & 1 & * \end{array} \qquad \begin{array}{cc|cc} & & & d_j \\ & & F & T \\ \hline c_j & F & F & F \\ & T & F & T \end{array}$$

$$(1 \leq i \leq n) \qquad\qquad (n+1 \leq j \leq n+m)$$

Wir definieren $c \cap d = \emptyset$ gdw. $\exists i : c_i \cap d_i = \emptyset$.

Lemma 5.1
Es gilt: $c \subseteq d$ genau dann, wenn $c \cap d = c$.

Beweis
Übungsaufgabe. ■

5.3 Kofaktoren und nochmals Shannon–Entwicklung

Definition 5.2
Seien c, d zwei (n, m)–Würfel. Ein (n, m)–Würfel c_d mit

$$c_d = \emptyset \text{ falls } c \cap d = \emptyset$$

und

$$(c_d)_i = \begin{cases} * & \text{falls } d_i = 0, 1 \\ T & \text{falls } d_i = F \\ c_i & \text{falls } d_i = *, T \end{cases}$$

heißt *Kofaktor von c bzgl. d*. Der *Kofaktor p_d eines Polynoms p bzgl. des (n, m)–Würfels* d ist das Polynom, das aus den Kofaktoren der Monome von p bzgl. d besteht.

Beispiel 5.2
Seien

$$M_P = \begin{pmatrix} 1 & 0 & * & 1 & T & F \\ 0 & * & 1 & 1 & F & T \\ 1 & 1 & * & * & T & T \end{pmatrix} \qquad \text{und} \qquad d = \begin{pmatrix} 1 & * & 0 & * & F & T \end{pmatrix}.$$

Dann gilt

$$M_{P_d} = \begin{pmatrix} * & 0 & * & 1 & T & F \\ * & 1 & * & * & T & T \end{pmatrix}.$$

Wichtiger Spezialfall: Kofaktoren bzgl. $x_i, \overline{x}_i \quad (x_i^\varepsilon)$:

$$M_P = \begin{pmatrix} 1 & 0 & * & 1 & T & F \\ 0 & * & 1 & 1 & F & T \\ 1 & 1 & * & * & T & T \end{pmatrix} \qquad x_1 = (1, *, *, *, T, T) \qquad \overline{x}_1 = (0, *, *, *, T, T)$$

Dann gilt:

$$M_{P_{x_1}} = \begin{pmatrix} * & 0 & * & 1 & T & F \\ * & 1 & * & * & T & T \end{pmatrix}$$

und

$$M_{P_{\overline{x}_1}} = \begin{pmatrix} * & * & 1 & 1 & F & T \end{pmatrix}.$$

Satz 5.1
(i) $P_{x_i^\varepsilon x_j^\delta} = (P_{x_i^\varepsilon})_{x_j^\delta}$ *mit* $x_i^\varepsilon x_j^\delta = x_i^\varepsilon \cap x_j^\delta$.

(ii) $P_{x_i^\varepsilon x_j^\delta} = P_{x_j^\delta x_i^\varepsilon}$.

Beweis
Folgt aus der Definition, da die Komponenten unabhängig voneinander behandelt werden.
■

Korollar 5.1
Für beliebige $l \leq n$ *und Permutationen* π *von* $\{1, \ldots, n\}$ *gilt*

(i) $P_{x_{i_1}^{\varepsilon_{i_1}} \cdots x_{i_l}^{\varepsilon_{i_l}}} = \left(P_{x_{i_1}^{\varepsilon_{i_1}} \cdots x_{i_{l-1}}^{\varepsilon_{i_{l-1}}}} \right)_{x_{i_l}^{\varepsilon_{i_l}}}$.

(ii) $P_{x_{i_1}^{\varepsilon_{i_1}} \cdots x_{i_l}^{\varepsilon_{i_l}}} = P_{x_{\pi(i_1)}^{\varepsilon_{\pi(i_1)}} \cdots x_{\pi(i_l)}^{\varepsilon_{\pi(i_l)}}}$. ■

Satz 5.2 (Rechenregeln für Kofaktoren)
Seien $F, G \in \mathbb{B}_{n,m}$. *Dann gilt*

(i) $(FG)_{x_i^\varepsilon} = F_{x_i^\varepsilon} G_{x_i^\varepsilon}$,

(ii) $\overline{(F_{x_i^\varepsilon})} = (\overline{F})_{x_i^\varepsilon}$.

Beweis
(i) Wegen $x_i\overline{x}_i = \emptyset$ liefert die Shannon–Entwicklung

$$\begin{aligned} x_i(FG)_{x_i} + \overline{x}_i(FG)_{\overline{x}_i} &= FG \\ &= (x_iF_{x_i} + \overline{x}_iF_{\overline{x}_i})(x_iG_{x_i} + \overline{x}_iG_{\overline{x}_i}) \\ &= x_iF_{x_i}G_{x_i} + \overline{x}_iF_{\overline{x}_i}G_{\overline{x}_i} \end{aligned}$$

Angenommen, es gelte $(FG)_{x_i} \neq F_{x_i}G_{x_i}$.
Dann existiert $a \in \{0,1\}^n$ so, daß für

$$\begin{aligned} a^{(0)} &= (a_1, \ldots, a_{i-1}, 0, a_{i+1}, \ldots, a_n) \\ a^{(1)} &= (a_1, \ldots, a_{i-1}, 1, a_{i+1}, \ldots, a_n) \end{aligned}$$

gilt:

$$(FG)_{x_i}(a^{(1)}) \neq F_{x_i}G_{x_i}(a^{(1)})$$

und damit $x_i(FG)_{x_i}(a^{(1)}) \neq x_iF_{x_i}G_{x_i}(a^{(1)})$ $\lightning$ $\overline{x}_i(FG)_{\overline{x}_i}(a^{(1)}) = 0 = \overline{x}_iF_{\overline{x}_i}G_{\overline{x}_i}(a^{(1)})$.

Analog erhält man $(FG)_{x_i} = F_{x_i}G_{x_i}$.

(ii) OBdA sei $\varepsilon = 1$.

Es gilt

$$\begin{aligned} x_i(\overline{F})_{x_i} + \overline{x}_i(\overline{F})_{\overline{x}_i} &= \overline{F} \\ &= \overline{x_iF_{x_i} + \overline{x}_iF_{\overline{x}_i}} \\ &= (\overline{x}_i + \overline{(F_{x_i})})(x_i + \overline{(F_{\overline{x}_i})}) \\ &= \overline{x}_i\overline{(F_{\overline{x}_i})} + x_i\overline{(F_{x_i})} + \overline{F}_{x_i}\overline{F_{\overline{x}_i}} \\ &= \overline{x}_i\overline{(F_{\overline{x}_i})} + x_i\overline{(F_{x_i})} \end{aligned}$$

Wir können nun analog wie bei (i) schließen. ■

Wir definieren nun Kofaktoren von Booleschen Funktionen:

Satz 5.3
Seien P, P' zwei Polynome für $F \in \mathbb{B}_{n,m}$ und sei d ein (n,m)–Würfel, $d \neq \emptyset$. Dann stimmen die durch die Kofaktoren P_d und P'_d beschriebenen Booleschen Funktionen F_{P_d} und $F_{P'_d}$ überein.

Beweis
Betrachte $m = 1$ (für beliebiges m analog).
Vollständige Induktion über die Länge l (des Eingabeteils) von d.

<u>$l = 1:$</u>
OBdA sei $d = x_1$, d. h. in P_{x_1} bzw. P'_{x_1} werden alle Monome m mit $\overline{x}_1 \in m$ gestrichen

und alle Vorkommen von x_1 durch $*$ ersetzt.

1. Fall: $a \in on(x_1)$, d. h. $a_1 = 1$.

Dann gilt
$$F_{P_{x_1}}(a) = F_P(a) = F_{P'}(a) = F_{P'_{x_1}}(a).$$

2. Fall: $a \notin on(x_1)$, d. h. $a_1 = 0$.

Dann gilt
$$F_{P_{x_1}}(a) = F_{P_{x_1}}(0, a_2, \ldots, a_n) = 1$$
$$\Longleftrightarrow F_P(1, a_2, \ldots, a_n) = 1 = F_{P'}(1, a_2, \ldots, a_n)$$
$$\Longleftrightarrow F_{P'_{x_1}}(0, a_2, \ldots, a_n) = F_{P'_{x_1}}(a) = 1.$$

$(l-1) \to l:$

OBdA sei $d = (x_1, \ldots, x_l)$.

1. Fall: $a \in on(d)$.

Dann gilt
$$\begin{array}{ccc} F_P(a) & = & F_{P_d}(a) \\ \| & & \\ F_{P'}(a) & = & F_{P'_d}(a) \end{array}$$

2. Fall: $a \notin on(d)$.

Dann existiert ein $i < l$ mit $a_i = 0$.
Es gilt

$$\begin{array}{rl} & F_{P_{x_1,\ldots,x_l}}(a) = F_{(P_{x_1,\ldots,x_{i-1},\hat{x}_i,x_{i+1},\ldots,x_l})_{x_i}}(a_1, \ldots, a_{i-1}, 0, a_{i+1}, \ldots a_n) = 1 \\ \Longleftrightarrow & F_{P_{x_1,\ldots,x_{i-1},\hat{x}_i,x_{i+1},\ldots,x_l}}(a_1, \ldots, a_{i-1}, 1, a_{i+1}, \ldots a_n) = 1 \\ & \qquad\qquad\qquad \| \\ & F_{P'_{x_1,\ldots,x_{i-1},\hat{x}_i,x_{i+1},\ldots,x_l}}(a_1, \ldots, a_{i-1}, 1, a_{i+1}, \ldots a_n) = 1 \text{ (nach Induktionsvor.)} \\ \Longleftrightarrow & F_{P'_{x_1,\ldots,x_{i-1},\hat{x}_i,x_{i+1},\ldots,x_l}}(a_1, \ldots, a_{i-1}, 0, a_{i+1}, \ldots a_n) = F_{P'_d}(a) = 1. \qquad \blacksquare \end{array}$$

Definition 5.3
Sei $F \in \mathbb{B}_{n,m}$ und d ein (n,m)–Würfel.
Der *Kofaktor* $F_d \in \mathbb{B}_{n,m}$ von F bzgl. d ist die durch den Kofaktor P_d eines Polynoms P für F bzgl. d beschriebene Funktion.

Satz 5.4 (Shannon–Entwicklung für Polynome)
Sei P ein beliebiges Polynom über n Literalen. Dann gilt für alle i, $1 \le i \le n$:

$$F_P = F_{(\overline{x}_i P_{\overline{x}_i} + x_i P_{x_i})}.$$

Beweis
Es gilt

$$\begin{aligned} F_{(\overline{x}_i P_{\overline{x}_i} + x_i P_{x_i})} &= F_{\overline{x}_i P_{\overline{x}_i}} + F_{x_i P_{x_i}} \\ &= \overline{x}_i F_{P_{\overline{x}_i}} + x_i F_{P_{x_i}} \\ &= \overline{x}_i (F_P)_{\overline{x}_i} + x_i (F_P)_{x_i} \\ &= F_p. \quad \blacksquare \end{aligned}$$

Wir müssen also Shannon–Entwicklungen für Polynome einerseits und Shannon–Entwicklung für (zugehörige) Funktionen andererseits nicht mehr unterscheiden.

Beispiel 5.3

$$M_P = \begin{pmatrix} 1 & 0 & * & 1 & T & F \\ 0 & * & 1 & 1 & F & T \\ 1 & 1 & * & * & T & T \\ * & 0 & 0 & 0 & T & F \end{pmatrix}$$

$$M_{P_{x_1}} = \begin{pmatrix} * & 0 & * & 1 & T & F \\ * & 1 & * & * & T & T \\ * & 0 & 0 & 0 & T & F \end{pmatrix}$$

$$M_{P_{\overline{x}_1}} = \begin{pmatrix} * & * & 1 & 1 & F & T \\ * & 0 & 0 & 0 & T & F \end{pmatrix}$$

$$M_{x_1 P_{x_1}} = \begin{pmatrix} 1 & 0 & * & 1 & T & F \\ 1 & 1 & * & * & T & T \\ 1 & 0 & 0 & 0 & T & F \end{pmatrix}$$

$$M_{\overline{x}_1 P_{\overline{x}_1}} = \begin{pmatrix} 0 & * & 1 & 1 & F & T \\ 0 & 0 & 0 & 0 & T & F \end{pmatrix}$$

$$M_{(\overline{x}_1 P_{\overline{x}_1} + x_1 P_{x_1})} = \begin{pmatrix} 1 & 0 & * & 1 & T & F \\ 1 & 1 & * & * & T & T \\ 0 & * & 1 & 1 & F & T \\ 0 & 0 & 0 & 0 & T & F \\ 1 & 0 & 0 & 0 & T & F \end{pmatrix}$$

$$\Longrightarrow F_P = F_{(\overline{x}_1 P_{\overline{x}_1} + x_1 P_{x_1})}.$$

5.4 Teile–und–Herrsche–Strategie

Die Tatsache, daß Kofaktoren die Untersuchung wichtiger Eigenschaften einer Booleschen Funktion erlauben, legt die folgende Teile–und-Herrsche–Strategie zur Bearbeitung von Polynomen P nahe:

$$\underline{\text{operate}}(P)=\underline{\text{merge}}(x_i\ \underline{\text{operate}}(P_{x_i}),\ \overline{x}_i\ \underline{\text{operate}}(P_{\overline{x}_i})).$$

operate und merge haben in unterschiedlichen Zusammenhängen unterschiedliche Bedeutung.
Die Strategie liefert effiziente Algorithmen, wenn die Auswahl der x_i geeignet getroffen wird.
Da viele Probleme für monotone Funktionen besonders einfach lösbar sind, versucht ESPRESSO, die x_i so auszuwählen, daß möglichst schnell monotone Kofaktoren zu bearbeiten sind.

5.4.1 Erste Anwendung: Ermittlung eines irredundanten Polynoms

Sei $f \in \mathbb{B}_n$.
operate(f) liefere eine irredundante Überdeckung von f aus PI.
Wir wollen aus *operate*$(f_{\overline{x}_i})$, *operate*(f_{x_i}) eine irredundante Überdeckung von f berechnen.
Wie ist *merge* zu organisieren?

Gegeben: P_0, P_1 irredundante Polynome aus PI für $f_{\overline{x}_i}, f_{x_i}$.
Gesucht: Irredundantes Polynom P von $\overline{x}_i P_0 + x_i P_1$ aus lauter PI.

Beobachtung:
Sei $m^{(1)} = (m_1, \ldots, m_{i-1}, 1, m_{i+1}, \ldots, m_{n+m}) \in x_i P_1$ PI.
Wenn $m^{(1)}$ vergrößert werden kann, dann lediglich zu

$$m' = (m_1, \ldots, m_{i-1}, *, m_{i+1}, \ldots, m_{n+m}) \qquad (\text{da } m^{(1)} \text{ PI}).$$

Dazu muß $m^{(0)} = (m_1, \ldots, m_{i-1}, 0, m_{i+1}, \ldots, m_{n+m})$ von $\overline{x}_i P_0$ überdeckt werden.
Wir müssen also prüfen, ob $m^{(0)}$ von $\overline{x}_i P_0$ überdeckt wird.
Dies ist im allgemeinen sehr aufwendig!

ESPRESSO realisiert folgende Teillösung:

1.) MERGE_WITH_IDENTITY

2.) MERGE_WITH_CONTAINMENT

Prozedur 1:

Überprüft, ob es Monome m gibt, die in P_0 und P_1 vorkommen.
$(\Longrightarrow m^{(0)} \in \overline{x}_i P_0, m^{(1)} \in x_i P_1)$.

Laufzeit: Lösung "straightforward": $O(\#P_0 * \#P_1)$.
Bessere Methode: $O(n \log n)$ mit $n = \#P_0 + \#P_1$.

Prozedur 2:

Überprüft, ob Monome aus P_1 von einem Monom aus P_0 überdeckt werden.
Laufzeit: $O(\#P_0 * \#P_1)$.

5.4.2 Zweite Anwendung: Überdeckung und Tautologie

Überdeckungstest
Wir können mit Hilfe von Kofaktoren überprüfen, ob ein Monom m von einem Polynom P überdeckt wird.

Satz 5.5
Sei P ein Polynom und m ein Monom. Dann gilt:

$$m \cdot P_m \equiv m \cdot P.$$

Beweis
Sei $c \in P$ mit $m \cap c \neq \emptyset$.
Dann gilt

$$(m \cap c_m)_j = \begin{cases} m_j & \text{falls } m_j \in \{0, 1, F\} \\ c_j & \text{falls } m_j \in \{*, T\} \end{cases}$$

und

$$(m \cap c)_j = \begin{cases} m_j & \text{falls } m_j \in \{0, 1, F\} \\ c_j & \text{falls } m_j \in \{*, T\} \end{cases}$$

Nun ist entweder

$$m \cap P = \emptyset \iff P_m = \emptyset$$

oder

$$(m \cap c)_j \equiv (m \cap c_m)_j \text{ für alle } j \text{ und alle } (m \cap c) \in (m \cap P). \quad \blacksquare$$

Satz 5.6 (Überdeckungskriterium)
Ein Monom m wird von einem Polynom P überdeckt genau dann, wenn $P_m \equiv 1$ (D. h. P_m ist eine „Tautologie").

Beweis
m wird von P überdeckt $\iff$ $m \cdot P \equiv m$.
($\Longrightarrow$):
Es gelte $m \cdot P \equiv m$.

Dann ist

$$\begin{array}{l} (m * P)_m \equiv m_m = (*, \ldots, *, T, \ldots, T) \equiv 1 \\ \quad ||| \\ m_m P_m \equiv 1 * P_m \equiv P_m. \end{array}$$

($\Longleftarrow$):
Sei $P_m \equiv 1$.

Dann ist

$$\begin{array}{l} mP_m \equiv m \\ \quad ||| \\ m * P. \quad \blacksquare \end{array}$$

Der Tautologietest kann mit der Teile–und–Herrsche–Strategie ausgeführt werden:

Satz 5.7
Sei $F = x_i F_{x_i} + \overline{x}_i F_{\overline{x}_i} \in I\!B_{n,m}$.
F *ist Tautologie genau dann, wenn* $F_{x_i}, F_{\overline{x}_i}$ *Tautologien sind.*

Beweis
($\Longleftarrow$):
Trivial, denn

$$x_i F_{x_i} + \overline{x}_i F_{\overline{x}_i} = x_i + \overline{x}_i = 1.$$

($\Longrightarrow$):
Annahme: Es existiere $a \in \{0,1\}^n$ mit $(F_{x_i}(a))_j \neq 1$ für ein j, $1 \leq j \leq n$.
Für $a' = (a_1, \ldots, a_{i-1}, 1, a_{i+1}, \ldots, a_{n+m})$ gilt

$$F_{x_i}(a') \neq 1,$$

denn F_{x_i} ist vom i–ten Argument unabhängig.
Folglich ist $x_i F_{x_i}(a') \neq 1$ und damit $(x_i F_{x_i}(a') + \overline{x}_i F_{\overline{x}_i}(a'))_j \neq 1$.
Also $(F(a'))_j \neq 1 \Longrightarrow F$ keine Tautologie. $\blacksquare$

Tautologietest für monotone Funktionen

Bemerkung

Wir vernachlässigen von hier an den Ausgabeteil der Würfel, sofern er nicht von Belang ist, und notieren dann nur noch den Eingabeteil.

Satz 5.8
Ein monotones Polynom P mit $f_P \in \mathbb{B}_n$ beschreibt eine Tautologie genau dann, wenn $(*, \ldots, *) \in P$.

Beweis
OBdA sei f_P monoton steigend.
Folglich enthalten alle Monome von P nur 1 und $*$.
Annahme: $(*, \ldots, *) \notin P$.
Dann gilt $(0, \ldots, 0) \notin m$ für alle $m \in P$ und folglich $f_P(0, \ldots, 0) \neq 1$ ↯. ■

Wir können damit Tautologietests für monotone Funktionen mit einem Ausgang ausführen durch einen Test für $m = (*, \ldots, *)$.

Verallgemeinerung für Polynome P von Funktionen $F \in \mathbb{B}_{n,m}$:
Teste, ob für alle j, $1 \leq j \leq m$, gilt:

$$(\underbrace{*, \ldots, *}_{n}, \ldots, \underset{n+j}{T}, \ldots) \in P.$$

Beschleunigung des allgemeinen Tautologietests:

Erzeuge möglichst schnell monotone Kofaktoren.
Idee: Faktorisiere dazu stets nach nicht-monotonen Variablen (also nach Variablen mit echtem Auftreten von x_i und $\overline{x}_i$).

Algorithmus zur Auswahl der „am wenigsten monotonen" Variablen:

Suche unter allen Spalten, die sowohl 1 als auch 0 enthalten, eine mit möglichst vielen solchen Einträgen $\neq *$.

Modifizierter Tautologietest:

- Wende Teile–und–Herrsche–Strategie jeweils für eine am wenigsten monotone Variable an.
- Die schließlich entstehenden monotonen Kofaktoren können dann mit effizientem monotonem Tautologietest bearbeitet werden.

Weiter verkürztes Verfahren:

1. Existiert eine Zeile mit lauter $*$, dann ist die Funktion eine Tautologie.
2. Für $n \leq 7$ kann der Tautologietest leicht von Hand ausgeführt werden.
3. Ist die Summe der von den einzelnen Würfeln überdeckten Minterme kleiner als 2^n (obere Schranke für die Zahl der überdeckten Minterme), dann ist die beschriebene Funktion keine Tautologie.

Beispiel 5.4

$$P = \begin{pmatrix} * & 1 & * & 0 \\ 1 & * & 0 & * \\ 1 & * & 0 & 1 \\ * & 1 & * & * \end{pmatrix}$$

Obere Schranke für die Zahl der überdeckten Minterme:

$$4 + 4 + 2 + 8 = 18 > 2^4 = 16.$$

Daraus können wir noch nichts schließen. Die 4. Spalte ist am wenigsten monoton.

$$P_{x_4} = \begin{pmatrix} 1 & * & 0 & * \\ * & 1 & * & * \end{pmatrix}$$

Obere Schranke für die Zahl der überdeckten Minterme:

$$4 + 8 = 12 < 16,$$

d. h. P_{x_4} ist keine Tautologie, und damit auch nicht P.

5.4.3 Dritte Anwendung: Komplementbildung

Die Komplementbildung ist im allgemeinen schwierig, aber einfach bei monotonen Funktionen, da das Komplement einer monotonen Funktion monoton ist.

Satz 5.9
Ist P ein monotones Polynom für eine monotone Funktion $F \in I\!B_{n,m}$, dann ist das nachfolgend definierte Polynom $\widehat{P}$ ein monotones Polynom für $\overline{F}$:

$$\widehat{P} := \begin{cases} \overline{x}_i \widehat{P_{\overline{x_i}}} + \widehat{P_{x_i}}, \text{ falls } P \text{ monoton steigend im } i\text{–ten Argument ist} \\ x_i \widehat{P_{x_i}} + \widehat{P_{\overline{x_i}}}, \text{ falls } P \text{ monoton fallend im } i\text{–ten Argument ist} \end{cases}$$

Beweis
OBdA sei $P(F)$ monoton steigend im i–ten Argument.
Dann hat P an der i–ten Position 1 oder $*$, d. h.

$$P_{\overline{x}_i} \subseteq P_{x_i}$$

denn P_{x_i} entsteht aus P durch Setzen aller i-ten Komponenten auf $*$ und $P_{\overline{x}_i}$ entsteht durch Übernahme aller Würfel mit $*$ in der i–ten Komponente.
Folglich gilt

$$P_{\overline{x}_i} * P_{x_i} = P_{\overline{x}_i} \text{ und } P_{x_i} = P_{x_i} + P_{\overline{x}_i},$$

d. h.

$$\begin{aligned}
P &= x_i P_{x_i} + \overline{x}_i P_{\overline{x}_i} \\
&= x_i(P_{x_i} + P_{\overline{x}_i}) + \overline{x}_i P_{x_i} P_{\overline{x}_i} \\
&= x_i(P_{x_i} + P_{x_i} P_{\overline{x}_i}) + \overline{x}_i P_{x_i} P_{\overline{x}_i} \\
&= x_i P_{x_i} + x_i P_{x_i} P_{\overline{x}_i} + \overline{x}_i P_{x_i} P_{\overline{x}_i} \\
&= x_i P_{x_i} + P_{x_i} P_{\overline{x}_i} \\
&= (x_i + P_{\overline{x}_i}) P_{x_i}.
\end{aligned}$$

Also

$$\begin{aligned}
\overline{P} &= \overline{(x_i + P_{\overline{x}_i}) P_{x_i}} \\
&= \overline{x}_i \overline{P_{\overline{x}_i}} + \overline{P_{\overline{\overline{x}}_i}} \\
&= \overline{x}_i \overline{P_{\overline{x}_i}} + \overline{P_{x_i}} \\
&= \widehat{P}.
\end{aligned}$$

Da die Operation $\widehat{P}$ ein Polynom für $\overline{F}$ liefert, schreibt man sie meist auch $\overline{P}$. ∎

Algorithmus zur Komplementbildung monotoner Polynome:

P sei oBdA in allen Argumenten monoton steigend.
Rekursives Vorgehen:

$$\begin{aligned}
\overline{P} &= \overline{x}_1 \overline{P_{\overline{x}_1}} + \overline{P_{x_1}} \\
&= \overline{x}_1 \overline{P_{\overline{x}_1}} + \overline{x}_2 \overline{P_{x_1 \overline{x}_2}} + \overline{P_{x_1 x_2}} \\
&= \dots \\
&= \sum_{i=1}^{k} (\overline{x}_i \overline{P_{x_1 \dots x_{i-1} \overline{x}_i}}) + \overline{P_{x_1 \dots x_k}}
\end{aligned}$$

bis $P_{x_1 \dots x_k}$ Tautologie und damit

$$\overline{P} = \sum_{i=1}^{k} \overline{x}_i \overline{P_{x_1 \dots x_{i-1} \overline{x}_i}}.$$

Die Effizienz ($\cong$ Rekursionstiefe) ist von der Auswahl der Variablen x_i abhängig:

Wähle x_i so, daß möglichst schnell Tautologien entstehen:

> Wähle solche Spalten, die im größten Würfel (kleinsten Monom) einen Wert $\neq *$ besitzen.
> Unter diesen wähle eine Spalte, die so oft wie möglich in anderen Würfeln vorkommt.

Wir können abbrechen, wenn

1. das Polynom ein Monom aus lauter $*$ enthält
   (dann ist das Polynom eine Tautologie, und das Komplement ist leer),

2. das Polynom leer ist (dann ist das Komplement eine Tautologie),

3. das Polynom nur aus *einem* Term besteht.
   Wir können dann nach deMorgan das Komplement bilden:

$$\overline{x_{i_1}^{\varepsilon_1} \dots x_{i_n}^{\varepsilon_n}} = x_{i_1}^{1-\varepsilon_1} \vee \dots \vee x_{i_n}^{1-\varepsilon_n}.$$

Beispiel 5.5

$$P = \begin{pmatrix} 0 & 1 & * & * \\ 0 & * & * & * \\ * & 1 & * & 1 \\ * & 1 & 0 & * \\ * & 1 & 0 & 1 \end{pmatrix}$$

Faktorisiere nach dem 1. Argument:

$$\overline{P} = x_1 \overline{P_{x_1}} + \overline{P_{\overline{x}_1}}$$

mit

$$P_{x_1} = \begin{pmatrix} * & 1 & * & 1 \\ * & 1 & 0 & * \\ * & 1 & 0 & 1 \end{pmatrix}$$

$$P_{\overline{x}_1} = \begin{pmatrix} * & 1 & * & * \\ * & * & * & * \\ * & 1 & * & 1 \\ * & 1 & 0 & * \\ * & 1 & 0 & 1 \end{pmatrix}$$

$P_{\overline{x}_i}$ ist Tautologie, also $\overline{P_{\overline{x}_1}} = \emptyset$.

Wir müssen $\overline{P_{x_1}}$ berechnen:
Faktorisiere nach dem 2. Argument:

$$\overline{P_{x_1}} = \overline{x}_2 \overline{P_{x_1 \overline{x}_2}} + \overline{P_{x_1 x_2}}$$

mit

$$P_{x_1 \overline{x}_2} = \emptyset, \text{ also } \overline{P_{x_1 \overline{x}_2}} \text{ Tautologie } \Longrightarrow \overline{P_{x_1}} = \overline{x}_2 + \overline{P_{x_1 x_2}}$$

$$P_{x_1 x_2} = \begin{pmatrix} * & * & * & 1 \\ * & * & 0 & * \\ * & * & 0 & 1 \end{pmatrix}.$$

Wir müssen $\overline{P_{x_1x_2}}$ berechnen:
Faktorisiere nach dem 4. Argument:

$$\overline{P_{x_1x_2}} = \overline{x}_4\overline{P_{x_1x_2\overline{x}_4}} + \overline{P_{x_1x_2x_4}}$$

mit

$$\begin{aligned} P_{x_1x_2\overline{x}_4} &= \begin{pmatrix} * & * & 0 & * \end{pmatrix} \\ P_{x_1x_2x_4} &= \begin{pmatrix} * & * & * & * \\ * & * & 0 & * \end{pmatrix} \text{ ist Tautologie.} \end{aligned}$$

Dann gilt

$$\overline{P_{x_1x_2}} = \overline{x}_4\overline{P_{x_1x_2\overline{x}_4}} = \begin{pmatrix} * & * & 1 & 0 \end{pmatrix},$$

also

$$\overline{P_{x_1}} = \begin{pmatrix} * & 0 & * & * \\ * & * & 1 & 0 \end{pmatrix}$$

und

$$\overline{P} = \begin{pmatrix} 1 & 0 & * & * \\ 1 & * & 1 & 0 \end{pmatrix}.$$

Komplementbildung beliebiger Funktionen:
Experimente zeigen, daß bei mehrausgängigen Funktionen sinnvollerweise jeder Ausgang für sich komplementiert wird.

Abbruchkriterium:

1. Existieren Zeilen aus lauter $*$, dann ist die Funktion eine Tautologie und das Komplement ist leer.

2. Ist die Beschreibung monoton, dann rufe die Prozedur zur Komplementbildung monotoner Funktionen auf.

3. Existiert eine Spalte aus lauter 0 bzw. 1, dann betrachten wir $c = (c_1, \ldots, c_n)$ mit

$$c_i = \begin{cases} 0 & i\text{–te Spalte besteht nur aus } 0 \\ 1 & i\text{–te Spalte besteht nur aus } 1 \\ * & \text{sonst} \end{cases}$$

   Es gilt:

$$P = P_c * c$$

   Also $\overline{P} = \overline{c} + \overline{P_c}$ ($\overline{c}$ ist einfach zu berechnen).
   Es genügt also, $\overline{P_c}$ zu berechnen.

Algorithmus zur Komplementierung von Polynomen mit einem Ausgang:

Arbeite wieder mit der Teile–und–Herrsche–Strategie, da

$$\overline{P} = x_i\overline{P_{x_i}} + \overline{x}_i\overline{P_{\overline{x}_i}}.$$

Faktorisiere nach den am wenigsten monotonen Variablen.

5.5 Ablaufskizze von ESPRESSO II

Eingabe: Beschreibung einer Booleschen Funktion F durch On–Menge $on(F)$ und don't–care–Menge $dc(F)$.

Gesucht: Möglichst effiziente Polynomdarstellung.

Schritte:

1. Berechnung der Off–Menge $off(F)$.
2. Expandiere alle Würfel der On–Menge $on(F)$ zu PI.
   Wir benötigen dazu die Off–Menge $off(F)$, um zu prüfen, ob der Würfel in einer bestimmten Dimension expandiert werden kann [vergleiche 5.5.1].
3. Sortiere die restlichen PI aus [vergleiche 5.5.2].
4. Mache die restliche Überdeckung irreduzibel [vergleiche 5.5.3].
5. Reduziere irredundante Überdeckung, d. h. Würfel werden kleiner, also nicht prim gemacht. (Abhängig von der Reihenfolge der Würfel!)
   Gehe zurück zu Schritt 2, bis keine Veränderungen mehr eintreten.
6. Letzter, von der Reihenfolge unabhängiger Versuch der Verkleinerung.
   Zurück zu Schritt 2, bis keine Veränderung mehr eintritt.
7. Schluß: Ausdünnen der PLA–Matrix.

5.5.1 Expandieren von Würfeln zu Primimplikanten

Gegeben:

(i) Polynom $P_{ON} = (m_1, \ldots, m_{t_{ON}})$ für $on(F)$.

(ii) Polynom $P_{OFF} = (r_1, \ldots, r_{t_{OFF}})$ für $off(F)$.

OBdA sei jeder Würfel in P_{OFF} nur für einen Ausgang gültig (sonst Vervielfachung).

(iii) $c = m_i$ für ein $1 \leq i \leq t_{ON}$.

Gesucht:
PI c^+ von $f_{P_{ON}}$ mit

1.) $c \subseteq c^+$

2.) $\#\{j | m_j \subseteq c^+\}$ ist maximal unter 1.)

3.) Länge(c^+) ist minimal unter 1.) und 2.)

D. h. die Prozedur muß PI c^+ und Referenz–Parameter für die von c^+ überdeckten Würfel von P_{ON} ausgeben.

Hilfsmittel:

- „Blockiermatrix“
- „Überdeckungsmatrix“

Blockiermatrix:

$B(P_{OFF}, c) \in \mathbb{M}_{n+m,t_{OFF}}\{0,1\}$

$$1 \leq i \leq n: \qquad B_{ji} = \begin{cases} 1 & \text{falls } (c_i = 1 \text{ und } r_{ji} = 0) \text{ oder } (c_i = 0 \text{ und } r_{ji} = 1) \\ 0 & \text{sonst} \end{cases}$$

$$n+1 \leq i \leq n+m: \qquad B_{ji} = \begin{cases} 1 & \text{falls } c_i = F \text{ und } r_{ji} = T \\ 0 & \text{sonst} \end{cases}$$

Zur Berechnung von c^+ ist die Menge $L_c \subseteq \{1, \ldots, n+m\}$ gesucht, so daß L_c Spaltenüberdeckung von $B(P_{OFF}, c)$ ist, d. h. für alle $j \in \{1, \ldots, t_{OFF}\}$ existiert $k \in L_c$ mit $B_{jk} = 1$. Betrachte

$$c_i^+ = \begin{cases} c_i & \text{falls } i \in L_c \\ * & \text{sonst für } i \leq n \\ T & \text{sonst für } i > n \end{cases}$$

Offenbar gilt:

$$c^+ \in I(P_{ON}), c \subseteq c^+$$

und

$$c^+ \cap P_{OFF} = \emptyset.$$

Satz 5.10

(i) Zu jedem Implikanten $c' \in I(F_{P_{ON}})$ mit $c \subseteq c'$ korrespondiert eineindeutig eine Spaltenüberdeckung von $B(P_{OFF}, c)$.

(ii) Ist die Spaltenüberdeckung minimal, so ist der zugehörige Implikant ein Primimplikant. ■

Beispiel 5.6

	x_1	x_2	x_3
c	0	0	*
P_{OFF}	1	1	*
	*	1	1
$B(P_{OFF}, c)$	1	1	0
	0	1	0

$L_c = \{x_2\}$ ist minimal und damit $c^+ = (*, 0, *)$.

Ein minimales L_c erfüllt 1.) und 3.), aber nicht 2.). Betrachte deshalb:

<u>Überdeckungsmatrix:</u>

$C(P_{ON}, c) \in I\!M_{n+m,t_{ON}}\{0,1\}$
Die Überdeckungsmatrix lenkt die Spaltenüberdeckung L_c von $B(OFF, c)$ gemäß 2.), sichert also, daß maximal viele Würfel aus P_{ON} von c^+ überdeckt werden.

$$1 \leq i \leq n: \qquad C_{ij} = \begin{cases} 1 & \text{falls } c_i = 1 \text{ und } m_{ji} \neq 1 \\ 1 & \text{falls } c_i = 0 \text{ und } m_{ji} \neq 0 \\ 0 & \text{sonst} \end{cases}$$

$$n+1 \leq i \leq n+m: \qquad C_{ij} = \begin{cases} 1 & \text{falls } c_i = F \text{ und } m_{ji} \neq T \\ 0 & \text{sonst} \end{cases}$$

Überdeckt L_c die j–te Zeile, so wird m_j nicht von c^+ überdeckt.
Suche deshalb eine Spaltenüberdeckung L_c von $B(P_{OFF}, c)$, die minimal viele Zeilen von $C(P_{ON}, c)$ überdeckt.

Beispiel 5.7

	x_1	x_2	x_3
c	0	0	*
P_{ON}	1	0	1
	0	*	0
	0	0	*
	1	0	0
$C(P_{ON}, c)$	1	0	0
	0	1	0
	0	0	0
	1	0	0

$L_c = \{x_2\}$ überdeckt genau eine Zeile von $C(P_{ON}, c)$.
c^+ überdeckt die anderen drei Würfel.

Wir können nun L_c mit Hilfe von $B(P_{OFF}, c)$ und $C(P_{ON}, c)$ durch den Greedy–Algorithmus berechnen.

5.5.2 Berechnung wesentlicher Primimplikanten

Erinnerung:
$m \in PI(F)$ heißt *wesentlich* genau dann, wenn eine Eingabe a mit $a \leq m$ existiert, aber $a \not\leq m'$ für alle $m' \in PI(F), m \neq m'$.

Wir wissen, daß wesentliche PI in jeder minimalen Überdeckung vorkommen. Daher können wir das Problem „reduzieren“ durch Ermittlung und Elimination aller wesentlichen PI und Bearbeitung des „Restproblems“.

Definition 5.4
Die *Distanz* δ zweier (n, m)–Würfel $a, b \in \{0, 1\}^n \times \{T, F\}^m$ ist definiert durch:

$$\delta(a, b) = \delta(E(a), E(b)) + \delta(A(a), A(b))$$

mit

$$\delta(E(a), E(b)) = \#\{i | a_i b_i = \emptyset\}$$

$$\delta(A(a), A(b)) = \begin{cases} 0 & \exists i \in \{n+1, \ldots, n+m\} : a_i b_i = T \\ 1 & \text{sonst} \end{cases}$$

Der Konsensus $\odot$ zweier (n, m)–Würfel $a, b \in \{0, 1, *\}^n \times \{T, F\}^m$ ist definiert durch:

(i) $\delta(a, b) \neq 1$:

$$a \odot b = \begin{cases} (a_1 b_1, \ldots, a_{n+m} b_{n+m}) & \text{falls } \delta(a, b) = 0 \\ \emptyset & \text{sonst} \end{cases}$$

(ii) $\delta(E(a), E(b)) = 1$ und $\delta(A(a), A(b)) = 0$:

$$(a \odot b)_i = \begin{cases} a_i b_i & \text{falls } a_i b_i \neq \emptyset \text{ oder } i > n \\ * & \text{sonst} \end{cases}$$

(iii) $\delta(E(a), E(b)) = 0$ und $\delta(A(a), A(b)) = 1$:

$$(a \odot b)_i = \begin{cases} a_i b_i & \text{falls } i \leq n \\ a_i \vee b_i & \text{falls } i > n \end{cases}$$

Satz 5.11
Sei P eine minimale Überdeckung (bzgl. "single cube containment") aus PI von F.

$$H := \{P \cup P_{DC}(F)\} \setminus \{m\} \textit{ für ein } m \in P \textit{ und } H' := \{h \odot m | h \in H\}.$$

Dann ist m wesentlich genau dann, wenn $m \not\subseteq H'$. ■

Algorithmus:

1. Bilde H, H' wie in Satz 5.11 beschrieben.
2. Überprüfe das Überdeckungsproblem $m \subseteq H'$ mit Tautologietest für H'_m.

5.5.3 Berechnung irredundanter Überdeckungen

Wir können bisher prime Überdeckungen bestimmen. Wir wollen diese Überdeckungen irredundant machen, d. h. minimale Teilüberdeckungen auswählen.

<u>1. Prozedur:</u>
Berechne ausgehend von $on(F)$, $dc(F)$ die Menge der *sekundär wesentlichen PI.*
($m \in P_{ON}(F) \cup P_{DC}(F)$ heißt „sekundär wesentlich“, wenn $P_{ON} \setminus \{m\}$ keine Überdeckung der On–Menge ist.)
Die Menge E der sekundär wesentlichen PI muß zu jeder zulässigen Teilüberdeckung gehören.

Idee:
Tautologietest für $((P_{ON} \cup P_{DC}) \setminus \{m\})_m$

<u>2. Prozedur:</u>
Berechnet aus der Menge der nicht sekundär wesentlichen PI die Menge R_t der „total redundanten“ PI.
(m heißt *total redundant*, wenn $m \in E \cup P_{DC}$.)
Die PI aus R_t können bestimmt weggelassen werden.

Idee:
Tautologietest für $(E \cup P_{DC})_m$.

<u>3. Prozedur:</u>
Berechne minimale Teilmenge R_c von $R_p = P_{ON} \setminus (E \cup R_t)$ mit: $R_c \cup E$ ist Überdeckung von $on(F)$ (sehr aufwendig!)

Idee:
Sei $R_p = \{r_1, \ldots, r_s\}$.
Betrachte

$$g : \{0,1\}^s \to \{0,1\}$$

mit

$$g(y) = 1 \iff on(F) \subseteq E \cup \{r_i | y_i = 1\}$$

g ist monoton.
Eine minimale Überdeckung aus R_p zu finden, heißt, einen maximalen Implikanten von g zu bestimmen.

Idee:

1. Berechnung einer monotonen Überdeckung P von $\overline{g}$.

2. Berechnung des Komplements.

5.5.4 Reduzieren von Würfeln

Wir wollen die lokal optimale Lösung aufgeben, um eine (global) bessere zu erreichen.

Gegeben: Überdeckung P von F, $m \in P$.

Gesucht: Kleinster Würfel m', so daß $m' \cup (P \setminus \{m\})$ Überdeckung von F ist, d. h. m' ist minimal mit:

$$m * \overline{(P \setminus \{m\}) \cup dc(F)} \subseteq m'.$$

Die Berechnung von m' wird mit Hilfe der Prozedur REDUCE ausgeführt.

Anhang A

Literatur

Boolesche Funktionen

G. Boole: *An introduction of the laws of thought, on which are founded the mathematical theories of logic and probabilities,* Dover, 1851 (Neuauflage).

I. Wegener: *The complexity of Boolean functions,* Wiley, 1987. 72 / HK / f 3210

I. Wegener: *Effiziente Algorithmen für grundlegende Funktionen,* Teubner, 1989. 72 / HA / f 5271 772 / HA / f 5271:...

C. Meinel: *Modified branching programs and their computational Power,* Springer, 1989. (Band 370 der *Lecture Notes in Computer Science (LNCS);* Überarbeitete Fassung einer Habilitationsschrift.) 72 / HA / f 5437

C. Meinel: *Effiziente Algorithmen,* Seminarbericht der Humboldt-Universität Berlin, Sektion Mathematik 61, 1984. 72 / HA / f 6540 772 / HA / f 6540:...

E. J. McCluskey: *Logic Design Principles,* Prentice Hall, 1986. 72 / KD / f 6628 772 / KD / f 6628:...

Boolesche Algebren

M. A. Breuer, A. D. Friedman: *Diagnosis and Reliable Design of Digital Systems,* Computer Science Press (Woodland Hills), 1976.

Chiprealisationen

T. Lengauer: *Combinational algorithms for integrated circuit Layout,* Teubner, 1990. 72 / KD / f 6512 772 / KD / f 6512:...

Komplexität von Funktionenfolgen

I. Wegener: *The complexity of Boolean functions*, a. a. o..

C. Meinel: *Modified Branching Programs and their Computational Power*, a. a. o..

Listen NP-vollständiger Probleme

M. R. Garey, D. S. Johnson: *Computers and Intractability*, Freeman (New York), 1988. 72 / HK / f 2792 772 / HK / f 2792:...

D. S. Johnson: *The NP-completeness Column*, seit 1981 im *Journal of algorithms* 55 / z 10457.

Heuristische Verfahren

R. K. Brayton, G. P. Hachtel, C. T. McMullen, A. L. Sangiovanni–Vincentelli: *Logic minimization algorithms for VLSI synthesis*, Kluwer Academic Publishers (Boston, The Hague, Dordrecht, Lancaster), 1989. 72 / KD / f 6620 772 / f 6620:...

I. Wegener: *Effiziente Algorithmen für grundlegende Funktionen*, a. a. o..

Anhang B

Exkurse

B.1 Ladner–Fischer–Addierer

B.1.1 Verschiedene Addiererprinzipien

Um zwei n–Bit–Zahlen zu addieren, verfährt man meist nach der sogenannten *Schulmethode:*

$$
\begin{array}{cccc}
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0
\end{array}
\quad
\begin{array}{c}
\text{Überträge} \\
\Longrightarrow \\
\text{bilden}
\end{array}
\quad
\begin{array}{cccc}
0 & 1 & 1 & 1 \\
{}_1 1 & {}_1 0 & {}_1 1 & {}_0 0 \\
\hline
\end{array}
$$

$$
\begin{array}{c}
\text{Parity-} \\
\Longrightarrow \\
\text{rechnung}
\end{array}
\quad
\begin{array}{ccccc}
 & 0 & 1 & 1 & 1 \\
 & {}_1 1 & {}_1 0 & {}_1 1 & {}_0 0 \\
\hline
1 & 0 & 0 & 0 & 1
\end{array}
$$

Dieses Berechnungsverfahren hat zwar unter allen die kleinste Größe, aber (wegen der Art und Weise, in der der Übertrag „durchgereicht“ wird) lineare Tiefe. Um dieses Problem zu umgehen, wurden im Laufe der Zeit andere Bauformen von Addierern entwickelt:

- Der *Krapchenko–Addierer* hat Größe $\mathrm{O}(n)$ und Tiefe $\mathrm{O}\left(\log n + \sqrt{\log n}\right)$. Damit ist er der derzeit schnellste bekannte Addierer. Sein Aufbau ist allerdings nicht modular.
- Der sogenannte *Conditional Sum Adder* hat Größe $\mathrm{O}(n \log n)$ und Tiefe $2\lceil \log n \rceil + 1$.
- Der in der Praxis erfolgreiche *Ladner-Fischer-Addierer* hat Größe $\mathrm{O}(n)$ und Tiefe $2\lceil \log n \rceil + 2$.

Literatur:

- I. Wegener: *Effiziente Algorithmen für grundlegende Funktionen*, a. a. o..
- I. Wegener: *The complexity of Boolean functions*, a. a. o..

B.1.2 Das Präfix-Problem

Sei M eine Menge, $\circ : M \times M \longrightarrow M$ eine (binäre) assoziative Verknüpfung (und $(M, \circ)$ somit Halbgruppe). Sei $n \in \mathbb{N}$, $x_0, \dots, x_{n-1} \in M$. Das *Präfix–Problem* besteht darin, alle $p_i = x_i \circ \cdots \circ x_0$, $i = 0, \dots, n-1$, zu berechnen.

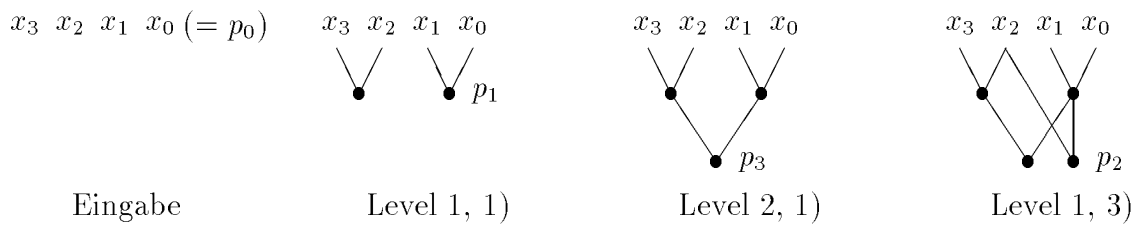
Rekursive Methode: $p_0 = x_0$, $p_i = x_i \circ p_{i-1}$ ist mit dem geringsten Aufwand durchzuführen, aber nicht parallelisierbar.

Binäre Methode: Sei oBdA $n = 2^k$.

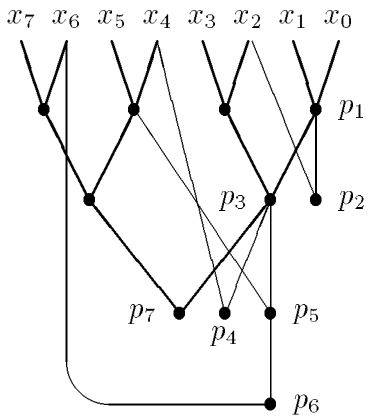
1) Berechne parallel die $\frac{n}{2}$ Paare $x_{n-1} \circ x_{n-2}, \dots, x_1 \circ x_0$.
2) Wende den Algorithmus rekursiv auf die $\frac{n}{2}$ Paare an. Damit werden bereits alle p_i mit ungeradem i berechnet.
3) Alle p_i mit geradem i können dann parallel durch $p_i = x_i \circ p_{i-1}$ berechnet werden.

Die Effektivität dieses Algorithmus' ist: Größe $O(n)$, Tiefe $O(\log n)$.

Beispiel B.1
$k = 2$:



$k = 3$:



Beachte, daß 5 „gerade“ für den inneren Durchlauf des Algorithmus' ist, denn seine Eingaben sind: (3) $7 \circ 6$, $\boxed{(2)\ 5 \circ 4}$, (1) $3 \circ 2$ und (0) $1 \circ 0$.

B.1.3 Überträge als Präfix–Problem

Seien $x = (x_{n-1}, \ldots, x_0)$ und $y = (y_{n-1}, \ldots, y_0)$ zwei zu addierende n–Bit–Zahlen. Betrachte die Situation *lokal an der i–ten Stelle:* Dort sind x_i und y_i bekannt, der Übertrag c_i an dieser Stelle ist zu berechnen, der von der $(i-1)$–ten Stelle kommende Übertrag c_{i-1} ist *nicht* lokal bekannt. Es gilt aber:

$$c_i = \begin{cases} 0 & x_i = y_i = 0 \\ c_{i-1} & (x_i = 0 \text{ und } y_i = 0) \text{ oder } (x_i = 0 \text{ und } y_i = 1) \\ 1 & x_i = y_i = 1 \end{cases} \qquad (c_{-1} := 0)$$

Dementsprechend definieren wir die Funktionen $E(c) \equiv 0$ (Eliminate), $P(c) = c$ (Propagate) und $G(c) \equiv 1$ (Generate), die mit der Operation $\circ$ eine Halbgruppe bilden (Beweis: Abgeschlossenheit und Assoziativität zeigen). Sei $A_i \in \{E, P, G\}$ die an der Stelle i auf den Übertrag anzuwendende Funktion, dann ist der Übertrag an der j–ten Stelle gerade

$$c_j = (A_j \circ A_{j-1} \circ \cdots \circ A_0)\,(0).$$

Die Bestimmung der Überträge läßt sich nun mit dem oben angegebenen Algorithmus für allgemeine Präfixprobleme effizient durchführen, anschließend werden Eingaben und Überträge der einzelnen Stellen mit der Parity-Funktion zum Endergebnis verarbeitet:

$$e = (e_n, \ldots, e_0) \qquad \text{mit} \qquad e_i = x_i \oplus y_i \oplus c_{i-1}$$

B.2 Test von PLAs

Literatur:

R. Kolla, P. Molitor, G. Osthof: *Einführung in den VLSI–Entwurf,* Teubner, 1989. 72 / KD / f 5404

B.2.1 Fehlermodelle

Es kann aus verschiedenen Gründen erwünscht sein, einen nach einer PLA–Vorlage gefertigten Chip auf seine Funktion zu prüfen:

- Um den Entwurf auf seine Korrektheit zu prüfen
- Um eine Fehlerstelle zu finden und sie materialtechnisch untersuchen zu können
- Zur laufenden Produktionskontrolle

Insbesondere im letzten Fall ist es zu aufwendig, alle 2^n möglichen Eingaben durchzuprobieren. Sobald Entwurffehler ausgeschlossen sind, werden auftretende Fehler ausschließlich durch Beschädigungen des Chips bei seiner Produktion verursacht. Es handelt sich also um Fehler, bei denen eine *lokale* Fehlfunktion fehlerhafte Ausgaben, wenn überhaupt, nur *durch eine höchstwahrscheinlich fehlerfreie Restschaltung hindurch* bewirken kann. Damit kann man die Auswirkungen eines angenommenen Fehlers berechnen und so wesentlich effizientere Testreihen entwickeln.

Für PLA–Entwürfe treten in der Praxis fast ausschließlich die folgenden Fehler auf:

- *“Stuck–at”–Fehler:* Unter einem Stuck–at–x–Fehler versteht man einen Fehler, der das „offene Ende“ einer (Eingabe–, Monom– oder Ausgabe–) Leitung unabhängig von den Eingaben den Wert $x \in \{0, 1\}$ annehmen läßt. (Der Fehler pflanzt sich auf alle Leitungen fort, die zwischen der Fehlstelle und dem offenen Ende mit der fehlerhaften Leitung verbunden sind.)
- *“Crosspoint”–Fehler:* Unter einem Crosspoint–T–Fehler versteht man eine fehlenden Schalter des Typs $T \in \{I, II\}$ an der Kreuzungsstelle zweier Leitungen.[1]

[1] Der Name PLA (“Programmable Logic Array”) kommt daher, daß AND– und OR–Feld fester Größe hergestellt und an *allen* Kreuzungspunkten mit Schaltern versehen werden; Die gewünschte Schaltung kommt dann dadurch zustande, daß die nicht gewünschten Schalter nachträglich zerstört werden. Die versehentliche Zerstörung eines eigentlich erwünschten Schalters ist daher wesentlich wahrscheinlicher als eine andere Fehlfunktion eines Schalters, z. B. durch falsche Dotierung. Gleichzeitig erklärt dies, warum ein *nicht* zerstörter Schalter kein Fehler, oder zumindest kein unbehebbarer Fehler, ist.

Außerdem ist es bei den derzeit erreichten Fehlerraten möglich, ausschließlich *einfache Fehler* zu betrachten, d. h. ein PLA–Chip soll nie mehrere Fehler gleichzeitig haben. Damit erhalten wir für ein PLA mit n Eingaben, m Ausgaben und s Monomen $2s+(2+s)(2n+m)$ relevante Fehler.

B.2.2 PLAs mit Testschaltung

Wir versehen die PLAs mit zusätzlichen Schaltelementen, anhand derer wir die Tests *weitgehend unabhängig von der vorliegenden Schaltung* durchführen können:

- Die *Eingaben* werden nicht mehr wie bisher mit Hilfe eines Inverters auf ein positives und ein negatives Literal aufgespalten, sondern mit einer „D_2“ genannten Schaltung, deren zwei zusätzliche Eingänge mit zwei neuen Eingaben y_1 und y_2 verbunden sind. Wird der Eingang y_1 (y_2) auf 0 gesetzt, zwingt dies auch alle mit positiven (negativen) Literalen besetzten Eingabeleitungen auf 0.

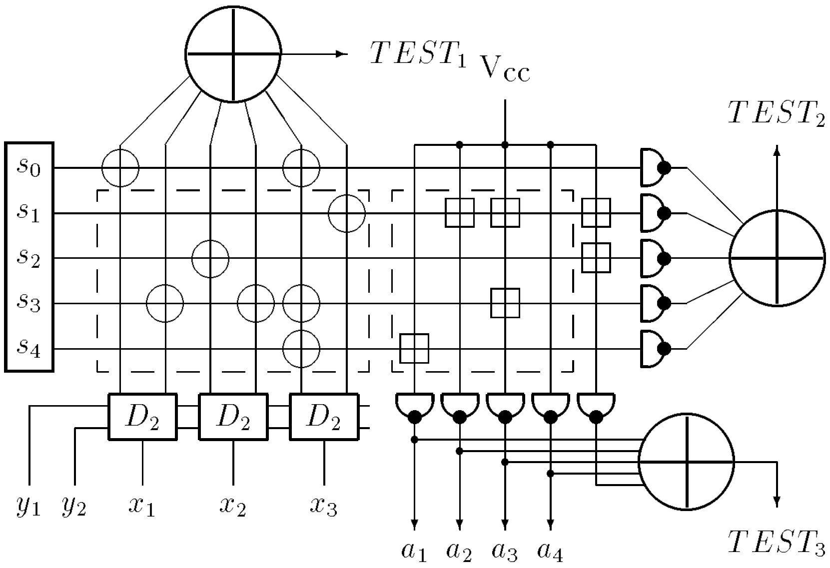
- Die *offenen Enden der Eingabeleitungen* werden in ein XOR–Gatter (Parity-Funktion) geführt. Der Ausgang dieses Gatters wird zu einer neuen Ausgabe $TEST_1$.

- Die *linken Enden der Monom–Leitungen* werden nicht mehr auf 1 gesetzt, sondern mit Werten $s_1, \ldots, s_s$ belegt. Außerdem wird eine neue Monomleitung mit Eingabewert s_0 eingefügt. Diese Leitung hat am Kreuzungspunkt mit einer Eingabeleitung genau dann einen Typ–I–Schalter, wenn diese Eingabeleitung bisher mit einer *geraden* Zahl von Typ–I–Schaltern versehen war. Die Werte $s_0, \ldots, s_s$ liefert eine Zusatzlogik so, daß wir jede Kombination mit $\sum_{i=0}^{s} s_i \in \{0, 1, s+1\}$ abrufen können.

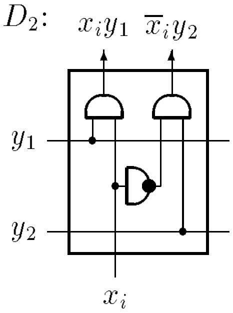
- Die *offenen Enden der Monomleitungen* werden mit Invertern versehen, deren Ausgaben wiederum in ein XOR–Gatter geleitet werden. Der Ausgang dieses Gatters wird zu einer neuen Ausgabe $TEST_2$.

- Eine neue *Ausgabeleitung* wird hinzugefügt, die am Kreuzungspunkt mit einer Monomleitung genau dann einen Typ–II–Schalter besitzen soll, wenn diese bisher mit einer *geraden* Zahl von Typ–II–Schaltern versehen war. Diese Ausgabeleitung wird mit dem üblichen Inverter versehen. Die Ausgaben *aller* Inverter auf Ausgabeleitungen werden dann in ein drittes XOR–Gatter mit Ausgabe $TEST_3$ geleitet. (Die bisherigen Ausgaben der Ausgabeleitungen bleiben natürlich bestehen — wir wollen den Chip nach den Tests ja auch noch benutzen!)

Beispiel B.2



wobei die gestrichelten Rahmen das ursprüngliche AND– bzw. OR–Feld anzeigen und



Der eigentliche Test geschieht dann mit den folgenden Testeingaben:

Test No.	s_j	y_1	y_2	x_i	# Tests	Korrektes Ergebnis für $TEST_1$	$TEST_2$	$TEST_3$
1_i	1_s	1	0	e_i	n	1	$s \; mod\, 2$	*
2_i	1_s	0	1	$1_n - e_i$	n	1	$s \; mod\, 2$	*
3_j	e_j	0	0	beliebig	$1+s$	0	$(s-1) \; mod\, 2$	$m \; mod\, 2$
4	0_s	0	0	beliebig	1	0	0	$(m-1) \; mod\, 2$
5	1_s	0	0	beliebig	1	0	$(s-1) \; mod\, 2$	*
					$2n+s+3$			

Einzelfehler sind nun wie folgt zu erkennen:

- Ein Stuck–at–0 bei einer Leitung für x_i führt bei Test 1_i zu $TEST_1 = 0$, sonst bleibt $TEST_1 = 1$.
- Ein Stuck–at–1 bei x_i führt bei Test 1_i zu $TEST_1 = 1$, sonst $TEST_1 = 0$.
- Analog für $\overline{x}_i$ bei Test 2_i.
- Ein Stuck–at–0 bei der Leitung für Monom j führt bei Test 3_j zu $TEST_2 = 0$, sonst bleibt $TEST_2 = 1$.
- Analog für Stuck–at–1 mit vertauschten Werten für $TEST_2$.
- Ein Stuck–at–0 auf einer Ausgabeleitung führt bei Test 4 zu $TEST_3 = m \; mod\, 2$.
- Ein Stuck–at–1 auf einer Ausgabeleitung führt bei Test 5 zu einem veränderten Wert von $TEST_3$, *es sei denn, die Ausgabeleitung hat keinen Typ–II–Schalter.* Die Einschränkung ist unwesentlich, da solche Leitungen auch bei korrekter Funktion stets 1 liefern. Der korrekte Wert für $TEST_3$ muß in Abhängigkeit von der implementierten Funktion berechnet werden.
- Ein Crosspoint–I–Fehler auf der Leitung für x_i ($\overline{x}_i$) führt bei Test 1_i (2_i) zu $TEST_2 \neq s \; mod\, 2$.
- Ein Crosspoint–II–Fehler auf der Leitung für Monom j führt bei Test 3_j zu $TEST_3 = m \; mod\, 2$.

B.3 Minimalpolynome disjunkt konjugierter Funktionen

Seien $f \in \mathbb{B}_n$, $g \in \mathbb{B}_m$, $f \owedge g(x_1, \ldots, x_n, y_1, \ldots, y_m) := f(x_1, \ldots, x_n) \wedge g(y_1, \ldots, y_m) \in \mathbb{B}_{n+m}$ („disjunkte Konjunktion“). Dann gilt $PI(f \owedge g) = \{p_f \owedge p_g | p_h \in PI(h)\}$, aber für die Minimalpolynome p_h^* gilt i. a. *nicht* $p^*_{f \owedge g} = p^*_f \owedge p^*_g$! Dazu ein Gegenbeispiel von Voigt und Wegener:

$$
\begin{aligned}
f(x_1, \ldots, x_4) \;:=\; & \overline{x}_1\overline{x}_2x_3\overline{x}_4 + \overline{x}_1x_2\overline{x}_3\overline{x}_5 + x_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1x_2x_3\overline{x}_4 \\
& + x_1\overline{x}_2x_3\overline{x}_4 + x_1x_2\overline{x}_3\overline{x}_4 + \overline{x}_1\overline{x}_2x_3x_4 + \overline{x}_1x_2\overline{x}_3x_4 \\
& + x_1\overline{x}_2\overline{x}_3x_4 + x_1x_2x_3\overline{x}_4 + x_1x_2x_3x_4
\end{aligned}
$$

Es folgt:

$$
\begin{aligned}
PI(f) &= \{\underbrace{\overline{x}_1\overline{x}_2x_3, \overline{x}_1x_2\overline{x}_3, x_1\overline{x}_2\overline{x}_3, x_1x_2x_3}_{\text{Kernimplikanten}}, x_3\overline{x}_4, x_2\overline{x}_4, x_1\overline{x}_4\} \\
p^*_f &= \underbrace{\overline{x}_1\overline{x}_2x_3 + \overline{x}_1x_2\overline{x}_3 + x_1\overline{x}_2\overline{x}_3 + x_1x_2x_3}_{=:\, P_1} + \underbrace{r\overline{x}_4 + s\overline{x}_4}_{=:\, P_2}
\end{aligned}
$$

mit $r \neq s$, $r, s \in \{x_1, x_2, x_3\}$ beliebig. Wir erhalten damit

$$p_f^*(x) \mathbin{\textcircled{\wedge}} p_f^*(y) = \underbrace{P_1(x)P_1(y) + P_2(x)P_1(y) + P_1(x)P_2(y)}_{176 \text{ Literale } (16 \times 6 + 16 \times 5)} + \underbrace{P_2(x)P_2(y)}_{16 \text{ Literale } (4 \times 4)}$$

Wir versuchen nun, $P_2(x)P_2(y)$ zu verkürzen. Dabei müssen wir die neun Eingaben (x, y) mit $x, y \in \{a, b, c\}$, $a := (1100)$, $b := (1010)$, $c := (0110)$ überdecken. Dies ist möglich mit dem 12 Literale (3×4) umfassenden Polynom

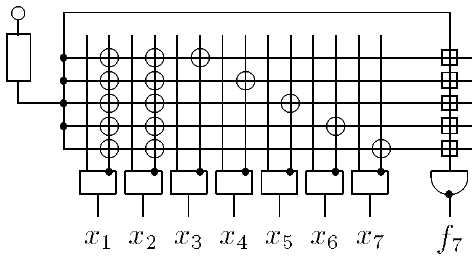
$$\hat{p}(x, y) := x_1\overline{x}_4 y_1\overline{y}_4 + x_2\overline{x}_4 y_2\overline{y}_4 + x_3\overline{x}_4 y_3\overline{y}_4$$

B.4 Weinberger–Arrays

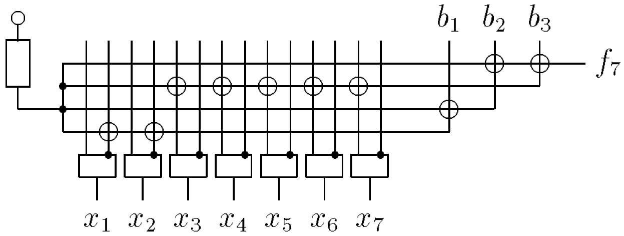
Betrachte die folgende Funktion:

$$f_n(x) := \bigwedge_{i=3}^{n} x_1 x_2 x_i = x_1 x_2 (x_3 + \cdots + x_n)$$

Um diese Funktion als PLA zu realisieren, ist die Fläche $(2n + 1)(n - 2)$ notwendig. Für $n = 7$ etwa ergibt sich



Dieselbe Funktion läßt sich in einer abgewandelten Bauform der PLA, den *Weinberger–Arrays,* auf der Fläche $4(2n + 4)$ realisieren:



Es gilt hierbei: $b_1 = \overline{\overline{x}_1 + \overline{x}_2} = x_1 x_2$, $b_2 = \overline{x_1 x_2}$, $b_3 = \overline{x_3 + \cdots + x_n}$, $f_n = \overline{b_2 + b_3} = \overline{b}_2 \overline{b}_3 = x_1 x_2 (x_3 + \cdots + x_n)$. Allgemein ist die Ausgabe eines Weinberger–Arrays ein f mit:

$$\begin{array}{rcl} b_i & = & \overline{b_1^{(i)} + \cdots + b_{k_i}^{(i)}} \qquad (1 \leq i \leq s) \\ b_l^{(i)} & \in & X_n \cup \overline{X}_n \cup \{b_1, \ldots, b_{i-1}\} \qquad (1 \leq l \leq k_i) \\ f & = & b_s \end{array}$$

Weil $\overline{+}$ eine Basis von $\mathbb{B}_n$ ist und $x \overline{+} y = \overline{x + y}$ gilt, läßt sich jede Funktion $f \in \mathbb{B}_n$ durch ein Weinberger–Array berechnen.