

DigiSem

Wir beschaffen und
digitalisieren



^b
**UNIVERSITÄT
BERN**

Dieses Dokument steht Ihnen online zur
Verfügung dank DigiSem, einer Dienstleistung
der Universitätsbibliothek Bern.

Kontakt: Gabriela Scherrer

Koordinatorin Digitale Semesterapparate

mailto: digisem@ub.unibe.ch Telefon 031 631 93 26

Rechneraufbau und Rechnerstrukturen

Von
Walter Oberschelp,
Gottfried Vossen

10., überarbeitete und erweiterte Auflage



Kt 16754

A.3926961

Oldenbourg Verlag München Wien

Prof. Dr. Gottfried Vossen lehrt seit 1993 Informatik am Institut für Wirtschaftsinformatik der Universität Münster. Er studierte, promovierte und habilitierte sich an der RWTH Aachen und war bzw. ist Gastprofessor u.a. an der University of California in San Diego, USA, an der Karlstad Universität in Schweden, an der University of Waikato in Hamilton, Neuseeland sowie am Hasso-Plattner-Institut für Softwaresystemtechnik in Potsdam. Er ist europäischer Herausgeber der bei Elsevier erscheinenden Fachzeitschrift *Information Systems*.

Prof. Dr. Walter Oberschelp studierte Mathematik, Physik, Astronomie, Philosophie und Mathematische Logik. Nach seiner Habilitation in Hannover lehrte er als Visiting Associate Professor an der University of Illinois (USA). Nach seiner Rückkehr aus den USA übernahm er den Lehrstuhl für Angewandte Mathematik an der RWTH Aachen, den er bis zu seiner Emeritierung im Jahr 1998 inne hatte.

Bibliografische Information Der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

© 2006 Oldenbourg Wissenschaftsverlag GmbH
Rosenheimer Straße 145, D-81671 München
Telefon: (089) 45051-0
www.oldenbourg-wissenschaftsverlag.de

Das Werk einschließlich aller Abbildungen ist urheberrechtlich geschützt. Jede Verwertung außerhalb der Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlages unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Bearbeitung in elektronischen Systemen.

Lektorat: Margit Roth
Herstellung: Anna Grosser
Umschlagkonzeption: Kraxenberger Kommunikationshaus, München
Gedruckt auf säure- und chlorfreiem Papier
Druck: Oldenbourg Druckerei Vertriebs GmbH & Co. KG
Bindung: R. Oldenbourg Graphische Betriebe Binderei GmbH

ISBN 3-486-57849-9
ISBN 978-3-486-57849-2

In diesem ersten Teil werden wir uns mit der Frage nach den Bausteinen, aus denen ein Rechner besteht, beschäftigen. Wir werden theoretische Hilfsmittel vorstellen, mit welchen sich Komponenten entwerfen lassen, die vorgegebene Probleme lösen können. Der Entwurf wird dabei aus rein *logischer* Sicht betrieben: Ein Rechner erscheint uns zunächst als eine „Black Box“, die durch ein bestimmtes Verhalten charakterisiert ist. Gesucht sind nun Bausteine, mit denen sich diese Black Box so ausfüllen lässt, dass das nach außen sichtbare Verhalten erklärbar wird.

In Kapitel 1 behandeln wir zweiwertige Schaltfunktionen, insbesondere Boolesche Funktionen. Für diese entwickeln wir Schaltnetze als Realisierungsmöglichkeit, und wir zeigen die Universalität dieses Konzeptes auf.

In Kapitel 2 behandeln wir Standardbausteine zur Realisierung Boolescher Schaltungen. Am Addierer zeigen wir exemplarisch die Entwicklung von Schaltnetzen: Durch erhöhten Hardware-Aufwand lassen sich Schaltnetze beschleunigen.

Mit Methoden zur Optimierung und zum Test von Schaltnetzen werden klassische Techniken des Rechnerentwurfs in Kapitel 3 behandelt. Vereinfachung und Fehlerdiagnose von Schaltnetzen müssen möglichst optimal bewerkstelligt werden, wobei auch der Einfluß physikalisch bedingter Fehlerquellen (Hasards) nicht verschwiegen werden darf.

In Kapitel 4 werden OBDDs eingeführt als alternative Darstellung Boolescher Funktionen und es wird deren Komplexität betrachtet. Dabei wird der heutige Kenntnisstand über „schwierig“ zu lösende Probleme skizziert — auch am Beispiel von Überdeckungsproblemen.

Zentrales Thema in Kapitel 5 ist die Einführung von Speicherbausteinen, welche dann auf (getaktete) sequentielle Maschinen führt. Als wichtige Anwendung derartiger Schaltwerke skizzieren wir u. a. zwei für jeden Rechner fundamentale Problemkreise: das Rechnen selbst und die Sicherung von Daten gegen technische Defekte.

In Kapitel 6 behandeln wir die Grundlagen der Darstellung von Daten (positive und negative Integer- bzw. Real-Zahlen oder allgemein Zeichenketten) in einem Rechner für die Zwecke der Rechnerarithmetik.

Kapitel 7 beschreibt zunächst PLAs und PALs als universelle, einheitlich formatierte Bausteine, die für eine automatisierte Herstellung sehr gut geeignet sind und die daher vielfache Verwendung in Rechnern finden, z. B. im Zusammenhang mit Mikroprogrammierung. Sodann werden komplexe Logik-Bausteine, insbesondere solche mit schneller Rekonfigurierbarkeit (FPGAs) beschrieben. Schließlich werden Möglichkeiten und Probleme der VLSI (Very Large Scale Integration) behandelt.

Kapitel 1

Schaltfunktionen und ihre Darstellung

1.1 Zahlendarstellungen

Maschinenmodelle wie der (hier als bekannt vorausgesetzte) endliche Automat verarbeiten Worte über einem fest gewählten (Input-) Alphabet Σ , d. h. Aneinanderreihungen von Symbolen aus Σ , für deren Länge a priori keine Begrenzung festgelegt wird, die also *variable Länge* haben dürfen. Für eine Beschäftigung mit den *realen* Rechnern zugrunde liegenden Konzepten ist diese idealisierte Sicht nicht sinnvoll: wir machen daher hier die generelle Voraussetzung, dass wir zu vorgegebenem Alphabet Σ nur Worte *fester Länge* über Σ betrachten. (Genauer bedeutet dies, dass wir zusätzlich zu einem Σ eine Wortlänge $n \in \mathbf{N}$ festlegen und nur Elemente von Σ^n betrachten.)

Wir wollen uns zunächst mit Zahlensystemen und den Alphabeten beschäftigen, auf denen sie basieren. Sei dazu $b > 1$ eine beliebige natürliche Zahl; dann heißt $\Sigma_b := \{0, \dots, b-1\}$ Alphabet des b -adischen Zahlensystems.

Beispiel 1.1 (a) Dem *Dezimalsystem* liegt das Alphabet $\Sigma_{10} = \{0, 1, 2, \dots, 9\}$ zugrunde. Dieses „klassische“ Alphabet der indo-arabischen Kultur wird jedem Leser bestens vertraut sein. Worte über diesem Alphabet sind z. B. 123, 489, 2046. Feste Wortlänge, etwa $n = 4$, erreicht man offensichtlich durch „führende Nullen“: 0123, 0489, 2046.

(b) $\Sigma_2 = \{0, 1\}$: Dual- oder Binäralphabet

$\Sigma_8 = \{0, 1, 2, 3, 4, 5, 6, 7\}$: Oktalalphabet

$\Sigma_{16} = \{0, \dots, 9, A, \dots, F\}$: Hexadezimalalphabet

Man beachte, dass Σ_{16} streng genommen das Alphabet $\{0, \dots, 15\}$ bezeichnet; anstelle der „Ziffern“ 10, 11 usw. werden jedoch generell, d. h. in allen Alphabeten Σ_b mit $b > 9$, „neue“ Symbole A, B usw. (hier also A, \dots, F) verwendet.

Diese Basen $b = 2, 8$ bzw. 16 spielen in der Informatik eine besondere Rolle, wie sich bald zeigen wird. Das gleiche gilt für $b = 256$, auf welcher der ASCII-Code (American Standard Code for Information Interchange) basiert (vgl. Kapitel 6).

(c) Von geringerer Bedeutung sind heute die Basen $b = 12$ („Dutzend“, „Gros“), $b = 20$ (franz. „vingt“) und das babylonische $b = 60$ (Zeit- und Winkelrechnung). \square

Die Bedeutung solcher Basen und der ihnen entsprechenden Alphabete erhellt der folgende Satz:

Satz 1.1 (*b-adische Darstellung natürlicher Zahlen*) Sei $b \in \mathbf{N}$ mit $b > 1$. Dann ist jede natürliche Zahl z mit $0 \leq z \leq b^n - 1$ (und $n \in \mathbf{N}$) eindeutig als Wort der Länge n über Σ_b darstellbar durch

$$z = \sum_{i=0}^{n-1} z_i b^i$$

mit $z_i \in \Sigma_b$ für $i = 0, \dots, n-1$. Als vereinfachende Schreibweise ist dabei die folgende Zifferschreibweise üblich (wobei streng genommen Wert und Schreibweise einer Zahl nicht identifiziert werden dürften):

$$z = (z_{n-1} z_{n-2} \dots z_1 z_0)_b.$$

Auf den Beweis dieses Satzes wollen wir an dieser Stelle verzichten (vgl. Aufgabe 1.1); als einfache Folgerung hieraus, die uns im Folgenden noch beschäftigen wird, notieren wir:

Korollar 1.2 (*Dualdarstellung natürlicher Zahlen*) Sei $n \in \mathbf{N}$. Dann ist jede natürliche Zahl z mit $0 \leq z \leq 2^n - 1$ eindeutig darstellbar in der Form

$$z = \sum_{i=0}^{n-1} z_i 2^i$$

mit $z_i \in \Sigma_2 = \{0, 1\}$ ($i = 0, \dots, n-1$).

Zu $b > 1$ und festem $n \in \mathbf{N}$ gibt es b^n Worte der Länge n über Σ_b , wobei man feste Wortlänge (d. h. alle b^n Worte sind gleich lang) durch „führende Nullen“ erreicht (vgl. Beispiel 1.1).

Beispiel 1.2 (a) $b = 10$ („Dezimalsystem“) Die Darstellung von $z = 2046$ lautet dann gemäß Satz 1.1

$$z = 2 \cdot 10^3 + 0 \cdot 10^2 + 4 \cdot 10^1 + 6 \cdot 10^0$$

und in Zifferschreibweise $z = (2046)_{10}$.

(b) $b = 2$ („Dualsystem“) Die Darstellung von $z = 87$ lautet dann gemäß Satz 1.1 (bzw. Korollar 1.2)

$$z = 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

und in Zifferschreibweise $z = (1010111)_2$. In diesem Beispiel ist $n = 7$. Mit Dualzahlen der Länge 7 sind somit die Zahlen von 0 bis $2^7 - 1 = 127$ darstellbar. \square

Man beachte, dass bei Verwendung der Zifferschreibweise die Klammerung sowie die explizite Angabe der Basis b im Folgenden häufig entfallen werden, wenn aus dem Zusammenhang hervorgeht, welches b gemeint ist.

Aus der in Satz 1.1 angegebenen Summendarstellung lässt sich die Zifferschreibweise offensichtlich leicht gewinnen: da der Stellung jeder Ziffer dabei jeweils eine

bestimmte Potenz der Basis entspricht, spricht man von solchen b -adischen Zahlendarstellungen auch als von *Stellenwertsystemen*.

Es sei darauf hingewiesen, dass natürliche Zahlen auch völlig anders dargestellt werden können, etwa wie folgt:

Satz 1.3 (*Polyadische Darstellung natürlicher Zahlen*) Es sei $(b_n)_{n \in \mathbf{N}}$ eine Folge natürlicher Zahlen mit $b_n > 1$ für alle $n \in \mathbf{N}$. Dann gibt es für jede natürliche Zahl z genau eine Darstellung der Form

$$z = \sum_{i=0}^N z_i \prod_{j=0}^{i-1} b_j = z_0 + z_1 b_0 + z_2 b_1 b_0 + \dots + z_N b_{N-1} \dots b_0$$

mit $0 \leq z_i < b_i$ für $i = 0, \dots, N$. (ohne Beweis)

1.2 Boolesche Algebra

Von den in Beispiel 1.1 angegebenen Basen für Zahlensysteme spielt die Basis $b = 2$ eine besondere Rolle: Die beiden Elemente 0 und 1 von Σ_2 spiegeln einerseits das in der Natur häufig anzutreffende „Prinzip der Zweiwertigkeit“ wieder, welches sich in Gegensätzen wie „ja — nein“, „wahr — falsch“ oder „hoch — tief“ findet. Dieses Prinzip liegt auch der auf Aristoteles zurückgehenden (klassischen) Aussagenlogik zugrunde („tertium non datur“). Andererseits sind 0 und 1 leicht technisch realisierbar, wenn man sie als zwei wohl unterschiedene Zustände versteht wie z. B. „es fließt Strom — es fließt kein Strom“, „es liegt eine Spannung an — es liegt keine Spannung an“ usw. Daher wollen wir uns zunächst mit diesem Alphabet $\Sigma_2 = \{0, 1\}$ näher beschäftigen, für welches wir von nun an die Bezeichnung B verwenden (zur Erinnerung an den englischen Mathematiker George Boole, der sich Mitte des vorigen Jahrhunderts zuerst mit dieser „Struktur“ aus mathematischer Sicht beschäftigte). Dabei wollen wir 1 als Wahrheitswert W (wahr) und 0 als Wahrheitswert F (falsch) interpretieren.

Wir notieren zunächst zwei mathematische Tatsachen über B :

- (1) Erklärt man auf B zwei zweistellige Operationen „ \leftrightarrow “ und „ \cdot “ durch

$$0 \leftrightarrow 0 = 1 \leftrightarrow 1 = 0.$$

$$1 \leftrightarrow 0 = 0 \leftrightarrow 1 = 1.$$

$$0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0.$$

$$1 \cdot 1 = 1.$$

so ist $(B, \leftrightarrow, \cdot)$ ein Körper der Ordnung 2 (häufig auch als Galoisfeld $\text{GF}(2)$ bezeichnet) mit dem Nullelement 0 und dem Einselement 1.

- (2) Erklärt man auf B drei Verknüpfungen wie folgt: Seien $x, y \in B$:

$$x \cup y := \text{Max}(x, y)$$

$$x \cap y := \text{Min}(x, y)$$

$$\bar{x} := 1 - x$$

so ist $(B, \cup, \cap, \bar{})$ eine *Boolesche Algebra*, d. h. ein distributiver, komplementärer Verband, in welchem es ein kleinstes (0) und ein größtes (1) Element gibt.

Die wichtigsten, in einer Booleschen Algebra geltenden Gesetze lauten:

- (a) *Kommutativgesetz*: $x \cup y = y \cup x$, $x \cap y = y \cap x$
- (b) *Assoziativgesetz*: $(x \cup y) \cup z = x \cup (y \cup z)$, $(x \cap y) \cap z = x \cap (y \cap z)$
- (c) *Verschmelzungsgesetze*: $(x \cup y) \cap x = x$, $(x \cap y) \cup x = x$
- (d) *Distributivgesetz*: $x \cap (y \cup z) = (x \cap y) \cup (x \cap z)$, $x \cup (y \cap z) = (x \cup y) \cap (x \cup z)$
- (e) *Komplementgesetz*: $x \cup (y \cap \bar{y}) = x$, $x \cap (y \cup \bar{y}) = x$
- (f) $x \cup 0 = x$, $x \cap 0 = 0$, $x \cap 1 = x$, $x \cup 1 = 1$
- (g) *de Morgansche Regeln*: $\overline{x \cup y} = \bar{x} \cap \bar{y}$, $\overline{x \cap y} = \bar{x} \cup \bar{y}$
- (h) $x = x \cup x = x \cap x = \bar{\bar{x}}$

Es sei bemerkt, dass es sehr viele nicht triviale Beispiele für Boolesche Algebren gibt, insbesondere solche mit unendlicher Grundmenge B . Wir haben es hier lediglich mit dem allereinfachsten Beispiel zu tun:

Satz 1.4 Für $B = \{0, 1\}$ liegt eine Boolesche Algebra vor.

Beweis: Wir wollen den Beweis nicht vollständig führen, sondern lediglich die Beweismethode exemplarisch für (c) erläutern und den Rest dem Leser überlassen. Da B nur zwei Elemente besitzt, reicht es, eine Liste mit allen möglichen Belegungen der in der Gleichung vorkommenden Variablen mit diesen beiden Werten anzulegen; sodann rechnen wir mit Hilfe der Definitionen der vorkommenden Operationen die linke bzw. rechte Seite aus und vergleichen, ob jeweils der gleiche Wert für entsprechende Belegungen herauskommt. Für $(x \cup y) \cap x = x$ erhalten wir die in Tabelle 1.1 zusammengefaßten Resultate.

Tabelle 1.1: Zum Beweis von Satz 1.4 (c).

Argumente			linke Seite	rechte Seite
x	y	$x \cup y$	$(x \cup y) \cap x$	x
0	0	0	0	0
0	1	1	0	0
1	0	1	1	1
1	1	1	1	1

Die Gleichheit der beiden rechten Spalten beweist die Behauptung. ∇

Die in Satz 1.4 angegebenen Rechenregeln werden wir noch häufig benutzen; wir wollen jedoch die Tatsache, dass wir dabei in der zweielementigen Booleschen Algebra

Tabelle 1.2: Boolesche Addition, Multiplikation und Negation.

x	y	$x + y$	$x \cdot y$	\bar{x}
0	0	0	0	1
0	1	1	0	1
1	0	1	0	0
1	1	1	1	0

rechnen, nicht mehr explizit erwähnen und uns stattdessen das Rechnen dadurch etwas erleichtern, dass wir für \cup und \cap die vertrauteren Symbole $+$ und \cdot verwenden. Wir bezeichnen diese auch wieder als Addition bzw. Multiplikation in B (und $\bar{}$ als Komplement) und geben ihre Funktionstabeln in Tabelle 1.2 noch einmal gesondert an.

Digitale Rechenanlagen arbeiten nun im Gegensatz zu Analogrechnern mit endlich vielen, sogar nur zwei verschiedenen Spannungswerten. Daher ist das Dualsystem zur Darstellung von Ein- bzw. Ausgabedaten für einen solchen Rechner naheliegend, aber auch für die rechnerinterne „Codierung“ von Daten und Befehlen. Nun wird man aus Gründen der besseren Lesbarkeit Eingabedaten lieber als Dezimalzahlen angeben und ebenso Ausgabedaten dezimal erhalten. Dazu muss der Rechner also in der Lage sein, eingegebene Dezimalzahlen in Dualzahlen zu konvertieren, sodann intern damit zu rechnen und schließlich rekonvertierte Dualzahlen auszugeben. Hierbei spielen somit Umwandlungen von Zahlen zu einer Basis b in Zahlen zu einer Basis b' eine Rolle, auf welche wir kurz eingehen wollen:

Für im Dualsystem arbeitende Rechner sind Multiplikationen mit 2 bzw. Divisionen durch 2 besonders einfach — durch Stellenverschiebung („shiften“) — realisierbare Operationen; daher ist für die Umwandlung von $b = 10$ nach $b' = 2$ das Divisionsrestverfahren („analytische“ Konvertierung), für die Umwandlung von $b = 2$ nach $b' = 10$ das Verfahren der fortgesetzten Multiplikation und Addition („synthetische“ Konvertierung) günstig. Wir wollen dies exemplarisch erläutern:

Beispiel 1.3 (a) Konvertierung dezimal \rightarrow dual: Man dividiert die gegebene Dezimalzahl mit Rest durch die Basis und wendet die gleiche Operation solange auf den jeweiligen Quotienten an, bis man das Divisionsergebnis 0 erhält. Die gesuchte Dualdarstellung ergibt sich dann durch „rückwärtiges“ Lesen der Reste:

$$\begin{aligned} 49 : 2 &= 24 \text{ Rest } 1 \\ 24 : 2 &= 12 \text{ Rest } 0 \\ 12 : 2 &= 6 \text{ Rest } 0 \\ 6 : 2 &= 3 \text{ Rest } 0 \\ 3 : 2 &= 1 \text{ Rest } 1 \\ 1 : 2 &= 0 \text{ Rest } 1 \end{aligned}$$

Hieraus folgt: $(49)_{10} = (110001)_2$.

Ein richtiges Resultat ist offensichtlich auch durch das im folgenden Beispiel demonstrierte Vorgehen erzielbar:

$$\begin{aligned}
 115 &= 1 \cdot 2^6 + 51 \\
 51 &= 1 \cdot 2^5 + 19 \\
 19 &= 1 \cdot 2^4 + 3 \\
 3 &= 0 \cdot 2^3 + 3 \\
 3 &= 0 \cdot 2^2 + 3 \\
 3 &= 1 \cdot 2^1 + 1 \\
 1 &= 1 \cdot 2^0 + 0
 \end{aligned}$$

Hieraus folgt: $(115)_{10} = (1110011)_2$

(b) Konvertierung dual \rightarrow dezimal:

$$\begin{aligned}
 (1010110)_2 &= 0 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 + 0 \cdot 2^5 + 1 \cdot 2^6 \\
 &= 0 + 2 + 4 + 0 + 16 + 0 + 64 \\
 &= (86)_{10}
 \end{aligned}$$

Die Konvertierung kann für große Zahlen aufwändig werden. □

Für spezielle Paare von Basen sind auch einfachere („lokale“) Verfahren möglich, zum Beispiel wenn es sich bei b und b' um Zweierpotenzen handelt. Ist etwa $b = 2$ und $b' = 8 = 2^3$ (bzw. $b' = 16 = 2^4$), so kann man je drei (bzw. vier) Dualziffern lokal in eine Oktalziffer (bzw. Hexadezimalziffer) umwandeln und umgekehrt.

Beispiel 1.4

$$\begin{aligned}
 (110010111)_2 &= (627)_8 \\
 &= (197)_{16}
 \end{aligned}$$

$$\begin{aligned}
 (110001011001)_2 &= (6131)_8 \\
 &= (C59)_{16}
 \end{aligned}$$

$$\begin{aligned}
 (A9F)_{16} &= (101010011111)_2 \\
 &= (5237)_8
 \end{aligned}$$

$$\begin{aligned}
 (716)_8 &= (111001110)_2 \\
 &= (1CE)_{16}
 \end{aligned}$$

□

Angemerkt sei an dieser Stelle, dass Digitalrechner zwar intern im Dualsystem arbeiten, das Oktal- bzw. Hexadezimalsystem aufgrund der engen „Verwandtschaft“ zum Dualsystem jedoch häufig dazu benutzt werden, die Inhalte von Registern eines Rechners darzustellen. Solche Zellen enthalten im Allgemeinen 0-1-Folgen (im Folgenden auch häufig *Bit¹-Folgen* genannt) einer festen Länge n (der so genannten *Wortlänge* des betreffenden Rechners). Da n recht groß sein kann (z. B. $n = 64$), ist eine Zusammenfassung von drei bzw. vier Dualstellen zu einer Oktal- bzw. Hexadezimalstelle etwa bei der Angabe von Registerinhalten häufig sinnvoll, da sie die Lesbarkeit vereinfacht.

1.3 Schaltfunktionen und Boolesche Funktionen

Wir wollen nun auf Rechner selbst zu sprechen kommen, und zwar aus einer „logischen“ Sicht, d. h. wir wollen die Frage diskutieren, wie sich ein Rechner oder genauer die Elemente eines Rechners verhalten. Aus der Sicht eines Benutzers erscheint ein Rechner in starker Idealisierung und Vereinfachung als eine „Black Box“, die zu einem bestimmten Input (I) einen eindeutig bestimmten Output (O) liefert:



Dabei hängt die Art, *wie* der Output vom Input bestimmt wird, offensichtlich vom Aufbau des Rechners ab; ferner arbeitet der Rechner *deterministisch*, d. h. er reagiert in eindeutiger Weise auf einen bestimmten Input. Unter der bereits erwähnten Annahme, dass Input und Output aus Daten und genauer (mittels oben erläutelter Konvertierungen) aus Dualfolgen bestehen, lässt sich diese Benutzersicht, welche die Black Box durch ihr *Verhalten* zu beschreiben versucht, wie folgt präzisieren:

Definition 1.1 Seien $n, m \in \mathbf{N}$, $n, m \geq 1$. Dann heißt eine Funktion $\mathcal{F} : B^n \rightarrow B^m$ *Schaltfunktion*.

Input für den Rechner ist also ein Bit- n -Tupel, Output ein Bit- m -Tupel. An einigen Beispielen wollen wir die Universalität dieses Konzeptes demonstrieren:

Beispiel 1.5 Addition von zwei 16-stelligen Dualzahlen: Input ist hier ein Bitvektor der Länge 32

$$b_1 \dots b_{16} b_{17} \dots b_{32},$$

dessen erste 16 Bits als erster, die zweiten 16 Bits als zweiter Summand aufgefasst werden. Output ist ein Bitvektor der Länge 17 (wegen der Möglichkeit eines Übertrags), welcher die Summe der beiden Dualzahlen darstellt. Die entsprechende Schaltfunktion lautet somit $\mathcal{A} : B^{32} \rightarrow B^{17}$. \square

¹Bit ist eine Kurzform für binary digit = Binär- oder Dualzahl.

Beispiel 1.6 Multiplikation von zwei 16-stelligen Dualzahlen: In Analogie zu Beispiel 1.5 lautet die entsprechende Schaltfunktion:

$$\mathcal{M} : B^{32} \rightarrow B^{32}$$

mit $\underbrace{b_1 \dots b_{16}}_{\text{1. Faktor}} \underbrace{b_{17} \dots b_{32}}_{\text{2. Faktor}} \mapsto \underbrace{c_1 \dots c_{32}}_{\text{Ergebnis}}$

□

Beispiel 1.7 Sortieren von 30 16-stelligen Dualzahlen: Input ist ein Vektor aus $30 \cdot 16 = 480$ Bits, welcher 30 (unsortierte) Dualzahlen der Länge jeweils 16 darstellt. Output ist ein anderer Bitvektor der Länge 480, welcher die gleichen Dualzahlen, nun jedoch (aufsteigend oder absteigend) sortiert, darstellt. Die Schaltfunktion lautet:

$$\mathcal{S} : B^{480} \rightarrow B^{480}.$$

□

Beispiel 1.8 Primzahltest: Unsere „Black Box“ soll nach Eingabe einer 480-stelligen Dualzahl x eine 1 ausgeben, falls die dieser Dualzahl entsprechende Zahl x eine Primzahl ist, und 0 sonst, d. h. die Schaltfunktion lautet:

$$p : B^{480} \rightarrow B$$

mit $p(x) = \begin{cases} 1 & \text{falls } x \text{ Primzahl} \\ 0 & \text{sonst} \end{cases}$

p führt also den Primzahltest durch für jede natürliche Zahl x mit

$$x \leq 2^{480} - 1 \approx 3,12175 \cdot 10^{144}.$$

Mit heutiger Technologie ist p schwierig zu realisieren, hätte aber andererseits hohe praktische Bedeutung, da Primzahltests (gerade wegen ihrer häufig schwierigen Durchführbarkeit) eine wichtige Rolle in der Kryptologie spielen. Ein entsprechendes Gerät, welches als Wert den größten Primfaktor von x ausgibt, gilt langfristig als nicht realisierbar. □

Für die nächsten beiden Beispiele benötigen wir einige Begriffe aus der Graphentheorie: Es sei P eine endliche Punktmenge, o.B.d.A. $P \subseteq \mathbf{N}$. Ist dann $K \subseteq P \times P$ eine symmetrische, nicht-reflexive Relation über P , so heißt das Paar $G := (P, K)$ ein (gewöhnlicher) *Graph* mit der Punktmenge P und der Kantenmenge K . Üblicherweise faßt man zwei zueinander inverse Kanten $(p_i, p_j), (p_j, p_i)$ zu einer „ungerichteten“ Kante $\{p_i, p_j\}$ zusammen. Ein n -Tupel $w = (p_1, p_2, \dots, p_n)$ von Punkten aus P heißt ein *Weg* in G , falls für alle $i = 1 \dots n-1$ die ungerichtete Kante $\{p_i, p_{i+1}\}$ zu G gehört. Ein solcher Weg w heißt weiter ein *Euler-Kreis*, falls $p_1 = p_n$ ist und alle Kanten von G auf dem Weg genau einmal vorkommen; w heißt *Hamilton-Kreis* („Traveling-Salesman-Tour“), falls $p_1 = p_n$ ist und alle Punkte von G auf dem Weg genau einmal vorkommen.

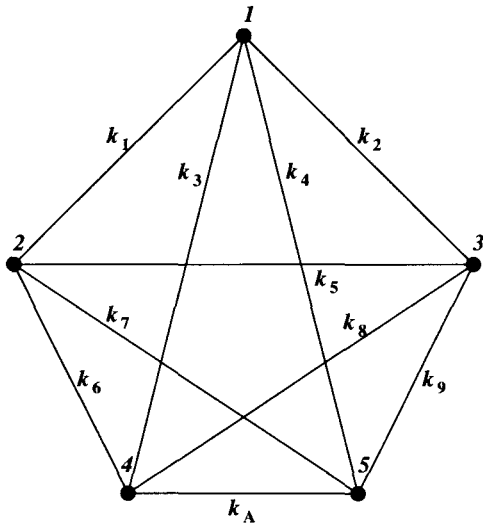


Abbildung 1.1: Ungerichteter Graph mit 5 Punkten.

Beispiel 1.9 Existenz eines Euler-Kreises in einem Graphen mit 5 Punkten: Sei

$$G = (\{1, \dots, 5\}, K).$$

Ein solcher Graph kann bis zu $\binom{5}{2} = 10$ Kanten haben, d. h. es gibt 2^{10} verschiedene ungerichtete Graphen mit 5 Punkten. Wir wählen nun die in Abbildung 1.1 gezeigte Nummerierung für Punkte bzw. Kanten. Damit codieren wir einen beliebigen solchen Graphen durch einen Bitvektor (x_1, x_2, \dots, x_A) wie folgt:

$$x_i = \begin{cases} 1 & \text{falls } k_i \in K \\ 0 & \text{sonst} \end{cases}$$

(Man beachte die Verwendung des Hexadezimalalphabets in diesem Beispiel: Der Buchstabe *A* bedeutet 10.) Will man nun mit Hilfe eines Rechners entscheiden, ob ein gegebener Graph mit 5 Punkten einen Euler-Kreis besitzt oder nicht, so lautet die entsprechende Schaltfunktion:

$$e : B^{10} \rightarrow B$$

$$\text{mit } e(x_1, \dots, x_A) := \begin{cases} 1 & \text{falls der durch } (x_1, \dots, x_A) \text{ codierte} \\ & \text{Graph einen Euler-Kreis besitzt} \\ 0 & \text{sonst} \end{cases}$$

□

Wir werden auf dieses Beispiel in Abschnitt 2.1 noch einmal zurück kommen.

Beispiel 1.10 Existenz eines Hamilton-Kreises in einem Graphen mit 250 Punkten:

Sei nun $|P| = 250$, so gilt $|K| \leq \binom{250}{2} = 31125$, d. h. ein Graph mit 250 Punkten

kann bis zu 31125 Kanten besitzen. Wie im letzten Beispiel lässt sich ein solcher Graph unter der Annahme einer festen Kantenummerierung durch einen Bitvektor der Länge 31125 codieren, und die Frage, ob in einem solchen Graphen ein Hamilton-Kreis existiert, ist formal beschreibbar durch die Schaltfunktion

$$h : B^{31125} \rightarrow B$$

$$\text{mit } h(x) := \begin{cases} 1 & \text{falls der durch } x \text{ codierte Graph einen} \\ & \text{Hamilton-Kreis besitzt} \\ 0 & \text{sonst} \end{cases}$$

Prinzipiell ist diese Aufgabe algorithmisch „leicht“ zu lösen: Man durchlaufe zu gegebenem $G = (P, K)$ alle möglichen Reihenfolgen der 250 Punkte und schaue nach, ob in G die dadurch geforderten Kanten tatsächlich vorkommen. Nun gibt es allerdings $250!$ solcher Reihenfolgen, d. h. nach der Stirlingschen Formel

$$n! \approx \left(\frac{n}{e}\right)^n \cdot \sqrt{2\pi n}$$

gibt es etwa

$$250! \approx \left(\frac{250}{e}\right)^{250} \cdot \sqrt{500\pi} \approx 3,232 \cdot 10^{492}$$

Reihenfolgen. Dass unsere Funktion h damit schwierig zu realisieren ist, sieht man wie folgt: Nehmen wir an, wir besäßen eine Schaltung (einen „Rechner“), deren Bauteile innerhalb von 10 Picosekunden, d. h. 10^{-11} sec, eine Operation ausführen können, und nehmen wir optimistisch an, dass die gesamte Schaltung eine Hierarchietiefe von 100 hat, d. h. sie besteht aus 100 „Ebenen“ von Bauteilen, die jeder Input sequentiell zu durchlaufen hat (vgl. Abbildung 1.2).

Dann kann *ein* Input, ein Bitvektor der Länge 31125, in $100 \cdot 10^{-11}$ sec = 10^{-9} sec bearbeitet werden. *Ein* Graph, welcher Anlaß zu etwa $3,232 \cdot 10^{492}$ Inputs gibt, ist dann in folgender Zeit mit unserem Algorithmus zu bearbeiten:

$$\begin{aligned} & 3,232 \cdot 10^{492} \cdot 10^{-9} \text{ sec} \\ & \approx 3,232 \cdot 10^{483} \text{ sec} \\ & \approx 1,023 \cdot 10^{476} \text{ Jahre} \end{aligned}$$

Selbst mit einer Technologie, welche Schaltungszeiten der oben erwähnten Art ermöglicht, ist h somit nicht zu realisieren. Es sei bereits an dieser Stelle angemerkt, dass auch für die Zukunft hier keine wesentlichen Verbesserungen mehr erwartet werden dürfen, wenngleich sich Bauteile mit Schaltzeiten im Picosekunden-Bereich schon im Laborstadium befinden. Denn einerseits würden wesentlich „schnellere“ Bauteile, die etwa in 10^{-20} sec schalten, keine Verbesserung bringen; die Anzahl der Jahre an Rechenzeit würde von 10^{476} auf 10^{465} fallen. \square

Dem Leser wird aufgefallen sein, dass wir in den letzten Beispielen große Buchstaben zur Bezeichnung von Schaltfunktionen der Form $\mathcal{F} : B^n \rightarrow B^m$ mit $m > 1$ (Beispiele 1.5 - 1.7) und kleine Buchstaben im Fall $m = 1$ (Beispiele 1.8 - 1.10) verwendet haben. Dieser Unterscheidung liegt ein wichtiger Spezialfall zugrunde, den wir nun näher betrachten wollen:

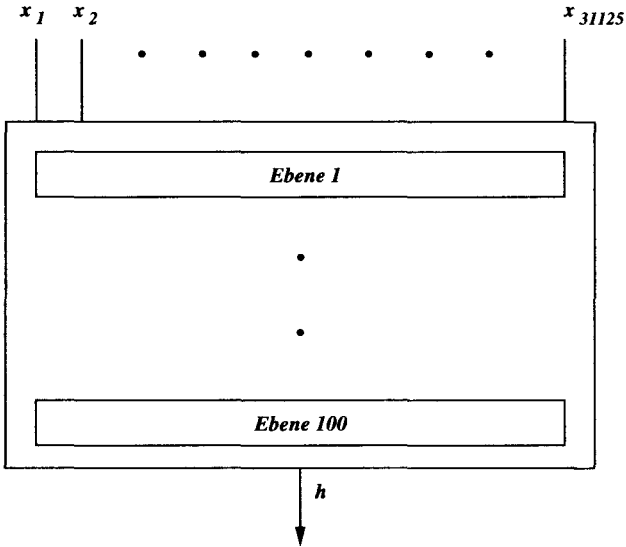


Abbildung 1.2: Schaltung der Hierarchietiefe 100.

Definition 1.2 Eine Schaltfunktion $f : B^n \rightarrow B$ heit (n -stellige) *Boolesche Funktion*.

Zwischen Schaltfunktionen und Booleschen Funktionen besteht folgender Zusammenhang: Sei $\mathcal{F} : B^n \rightarrow B^m$ mit $\mathcal{F}(x_1, \dots, x_n) = (y_1, \dots, y_m)$. Setzt man dann fr jedes $i \in \{1, \dots, m\}$

$$f_i : B^n \rightarrow B,$$

definiert durch

$$f_i(x_1, \dots, x_n) = y_i$$

so ist \mathcal{F} offensichtlich wie folgt darstellbar:

$$\mathcal{F}(x_1, \dots, x_n) = (f_1(x_1 \dots x_n), f_2(x_1 \dots x_n), \dots, f_m(x_1 \dots x_n))$$

fr alle $x_1, \dots, x_n \in B$.

Jede Schaltfunktion ist also durch eine Folge von Booleschen Funktionen beschreibbar, so dass es keine Einschrnkung bedeutet, wenn wir uns zunchst mit Booleschen Funktionen nher beschftigen. Dabei wollen wir zuerst die Frage klren, wie viele n -stellige Boolesche Funktionen es zu vorgegebenem $n > 0$ gibt. Fr $n = 1$ und $n = 2$ geben wir alle Mglichkeiten an:

Beispiel 1.11 Einstellige Boolesche Funktionen der Form $f : B \rightarrow B$: Das einzige Argument kann nur die Werte 0 oder 1 annehmen: fr die Funktionswerte stehen ebenfalls nur diese Werte zur Verfgung. Alle einstelligen Booleschen Funktionen erhlt man durch Bildung aller mglichen Kombinationen dieser Werte, was wir in Tabelle 1.3 zusammenfassen. Offensichtlich gilt $f_0(x) \equiv 0$, $f_1(x) = x$, $f_2(x) = \bar{x}$, $f_3(x) \equiv 1$. f_0 und f_3 sind konstante Funktionen, welche nicht von x abhngen; f_1 ist die Identitt und f_2 die Negation. □

Tabelle 1.3: Alle einstelligen Booleschen Funktionen.

x	$f_0(x)$	$f_1(x)$	$f_2(x)$	$f_3(x)$
0	0	0	1	1
1	0	1	0	1

Tabelle 1.4: Die 16 zweistelligen Booleschen Funktionen.

(1)	$x \cdot \bar{x}$	$x \cdot y$	$x \cdot \bar{y}$	x	$\bar{x} \cdot y$	y	\leftrightarrow	$x + y$	
(2)	$\equiv 0$	Min	$>$	x	$<$	y	\neq	Max	
(3)		\wedge	\nrightarrow	x	\nleftarrow	y	\nleftrightarrow	\vee	
x	y	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

(1)	$\overline{x + y}$		\bar{y}	$x + \bar{y}$	\bar{x}	$\bar{x} + y$	$\bar{x} \cdot y$	$x + \bar{x}$	
(2)	1-Max	=	1-y	\geq	1-x	\leq	1-Min	$\equiv 1$	
(3)	\downarrow	\leftrightarrow	$\neg y$	\leftarrow	$\neg x$	\rightarrow	\uparrow		
x	y	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

Beispiel 1.12 Zweistellige Boolesche Funktionen der Form $f : B^2 \rightarrow B$: Die beiden Argumente können nun auf $2^2 = 4$ verschiedene Arten mit 0 oder 1 belegt werden, und für jedes dieser vier Argumente sind wieder nur die Funktionswerte 0 oder 1 möglich, so dass wir $2^4 = 16$ zweistellige Boolesche Funktionen erhalten. Wir geben diese in Tabelle 1.4 an, wobei wir auch einige alternativ gebräuchliche Funktionssymbole notieren: Die mit (1) markierten Zeilen dieser Tabelle enthalten jeweils die in Zusammenhang mit der Booleschen Algebra ($+$, \cdot , \neg) bzw. dem Booleschen Körper (\leftrightarrow , \cdot) verwendeten Schreibweisen. Die in den mit (2) markierten Zeilen angegebenen Schreibweisen werden in der Arithmetik verwendet. Die mit (3) markierten Zeilen zeigen die in der Logik gebräuchlichen Schreibweisen.

Für einige dieser Funktionen sind auch Namen in Gebrauch:

- f_1 Konjunktion (AND)
- f_7 Disjunktion (OR)
- f_6 Antivalenz (Exclusive Or, XOR, \leftrightarrow , manchmal auch \oplus)
- f_9 Äquivalenz
- f_8 Peircescher Pfeil (Not Or, NOR, \downarrow)

- f_{13} Implikation
- f_{14} Shefferscher Strich (Not And, NAND, \uparrow)

□

Die in diesen beiden Beispielen angestellten Anzahlbetrachtungen lassen sich nun ohne weiteres verallgemeinern:

Satz 1.5 Für jedes $n \in \mathbb{N}$ mit $n \geq 1$ gibt es 2^{2^n} n -stellige Boolesche Funktionen.

Satz 1.5 über die Anzahl Boolescher Funktionen zu gegebener Stellenzahl n kann als Korollar aus folgendem Resultat abgeleitet werden: Für jedes $n, m \in \mathbb{N}$ mit $n, m \geq 1$ gibt es $2^{m \cdot 2^n}$ Schaltfunktionen der Form $\mathcal{F} : B^n \rightarrow B^m$.

Wie wir oben sahen, gibt es für $n = 1$ genau $2^{2^1} = 4$, für $n = 2$ genau $2^{2^2} = 16$ Boolesche Funktionen. Da die Anzahl Boolescher Funktionen vorgegebener Stellenzahl also sehr stark wächst (für $n = 3$ gibt es bereits 256, für $n = 4$ schon 65.536 solcher Funktionen), ist es unmöglich, das oben praktizierte Vorgehen, alle Möglichkeiten anzugeben, weiter fortzusetzen. Wir wollen als nächstes zeigen, dass dies aber auch keineswegs nötig ist, denn jede beliebige n -stellige Boolesche Funktion lässt sich aus wenigen Grundfunktionen „zusammenbauen“, und wir werden sogar sehen, dass ein- und zweistellige Funktionen dazu ausreichen.

Wir stellen dazu einige Vorüberlegungen an: Sei $n \geq 1$ und $f : B^n \rightarrow B$ eine beliebige n -stellige Boolesche Funktion. Dann kann f dargestellt werden durch eine Funktionstafel mit 2^n Zeilen, wobei die Argumente so angeordnet seien, dass in der i -ten Zeile ($0 \leq i \leq 2^n - 1$) gerade die Dualdarstellung von i steht; i heißt ein *Index* zu f .

Beispiel 1.13 $f : B^3 \rightarrow B$ sei die in Tabelle 1.5 gezeigte Boolesche Funktion. □

Tabelle 1.5: Boolesche Funktion zu Beispiel 1.13.

i	x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

Sei nun i eine solche Zeilennummer, und sei $i_1 \dots i_n$ die Ziffernfolge der Dualdarstellung von i .

Definition 1.3 i heißt *einschlägiger* Index zu f , falls $f(i_1, \dots, i_n) = 1$ ist.

In Beispiel 1.13 sind also 3, 5 und 7 die einschlägigen Indizes zu f .

Definition 1.4 Sei i ein Index von $f : B^n \rightarrow B$ und $(i_1 \dots i_n)_2$ die Dualdarstellung von i . Dann heißt die Funktion

$$m_i : B^n \rightarrow B$$

definiert durch

$$m_i(x_1, \dots, x_n) := x_1^{i_1} \cdot x_2^{i_2} \cdot \dots \cdot x_n^{i_n}$$

i -ter *Minterm* von f . Dabei sei

$$x_j^{i_j} := \begin{cases} x_j & \text{falls } i_j = 1 \\ \bar{x}_j & \text{falls } i_j = 0 \end{cases}$$

In Beispiel 1.13 sind also z. B. $m_3(x_1, x_2, x_3) = \bar{x}_1 \cdot x_2 \cdot x_3$, $m_4(x_1, x_2, x_3) = x_1 \cdot \bar{x}_2 \cdot \bar{x}_3$. Der Einfachheit halber lassen wir von nun an bei Mintermen die Argumente weg und schreiben kurz m_i für $m_i(x_1, \dots, x_n)$. Bei Konjunktionen lassen wir ferner (wie üblich) in Zukunft den Punkt weg und schreiben kurz xy für $x \cdot y$.

Bemerkung: Wesentlich für das Folgende ist die Beobachtung, dass ein Minterm m_i genau dann den Wert 1 annimmt, wenn das Argument (x_1, \dots, x_n) die Dualdarstellung von i liefert. In Beispiel 1.13 sieht man sofort:

$$\begin{aligned} m_3 = 1 & \Leftrightarrow \bar{x}_1 = 1 \wedge x_2 = 1 \wedge x_3 = 1 \\ & \Leftrightarrow x_1 = 0 \wedge x_2 = 1 \wedge x_3 = 1, \end{aligned}$$

und 011 ist die Dualdarstellung von 3.

Damit können wir Minterme wie folgt zur Beschreibung Boolescher Funktionen verwenden:

Satz 1.6 (*Darstellungssatz für Boolesche Funktionen*) Jede Boolesche Funktion $f : B^n \rightarrow B$ ist eindeutig darstellbar als Summe (im Sinne der Funktion f_7) der Minterme ihrer einschlägigen Indizes, d. h. ist $I \subseteq \{0, \dots, 2^n - 1\}$ die Menge der einschlägigen Indizes von f , so gilt

$$f = \sum_{i \in I} m_i,$$

und keine andere Minterm-Summe stellt f dar.

Beweis: Zu zeigen ist zweierlei: (1) Es existiert eine solche Darstellung, und (2) diese ist eindeutig.

Zu (1) (Existenz): Wir zeigen, dass die Funktionen f und $\sum_{i \in I} m_i$ für jedes Argument den gleichen Wert liefern: Sei dazu $j \in \{0, \dots, 2^n - 1\}$ und $j_1 \dots j_n$ die Dualdarstellung von j . Zwei Fälle sind zu unterscheiden:

(a) $f(j_1 \dots j_n) = 1$: Hieraus folgt $j \in I$, d. h. j ist ein einschlägiger Index von f . Also kommt m_j in der Summe $\sum_{i \in I} m_i$ vor; somit gilt (nach obiger Bemerkung) $\sum_{i \in I} m_i = 1$.

(b) $f(j_1 \dots j_n) = 0$: Hieraus folgt $j \notin I$, d. h. j ist kein einschlägiger Index von f . Dann kommt aber laut Voraussetzung m_j in $\sum_{i \in I} m_i$ nicht vor. Da nach obiger Bemerkung aber nur dieser Minterm den Wert 1 hätte beitragen können, folgt $\sum_{i \in I} m_i = 0$.

Zu (2) (Eindeutigkeit): Angenommen, es gibt zwei verschiedene Darstellungen von f durch Summen von Mintermen, d. h. es existieren $I, J \subseteq \{0, \dots, 2^n - 1\}$ mit $I \neq J$ und

$$f = \sum_{i \in I} m_i = \sum_{j \in J} m_j \quad (*)$$

Wegen $I \neq J$ gibt es dann einen Index, etwa k , der in der einen, aber nicht in der anderen Menge liegt. Sei etwa $k \in I$ und $k \notin J$. Sei dann $k_1 \dots k_n$ die Dualdarstellung von k , so gilt:

$$\sum_{i \in I} m_i(k_1, \dots, k_n) = 1 \text{ da } k \in I$$

$$\sum_{j \in J} m_j(k_1, \dots, k_n) = 0 \text{ da } k \notin J$$

Die beiden Summen stellen also verschiedene Funktionen dar, ein Widerspruch zu unserer Annahme (*). ∇

Zu diesem Satz sind einige Bemerkungen angebracht:

1. In der in Satz 1.6 angegebenen Darstellung einer Booleschen Funktion ist zu jedem Argument höchstens ein Summand gleich 1 (nämlich der dem Argument entsprechende Minterm, falls das Argument einen einschlägigen Index darstellt). Dies werden wir später noch benutzen, um eine alternative Darstellung Boolescher Funktionen anzugeben (vgl. Satz 1.12).
2. Ist die darzustellende Boolesche Funktion f identisch 0, so ist die leere Summe von Mintermen die entsprechende DNF-Darstellung. (Allerdings lässt sich $f \equiv 0$ auch wie folgt schreiben: $f(x_1, \dots, x_n) = x_1 \bar{x}_1$, was jedoch keine Mintermdarstellung ist.)

Die in Satz 1.6 angegebene Darstellung heißt auch *disjunktive Normalform* (DNF) einer Booleschen Funktion. In der englischsprachigen Literatur wird die DNF auch als Sonderfall einer *Sum of Products*-Darstellung, kurz SOP, bezeichnet.

Für Beispiel 1.13 liefert Satz 1.6 Folgendes:

$$\begin{aligned} f(x_1, x_2, x_3) &= m_3 + m_5 + m_7 \\ &= \bar{x}_1 x_2 x_3 + x_1 \bar{x}_2 x_3 + x_1 x_2 x_3 \end{aligned}$$

Eine wichtige Folgerung aus diesem Satz, welche uns die weiter oben gestellte Frage nach möglichen „Grundbausteinen“ für Boolesche Funktionen positiv beantwortet, ist:

Korollar 1.7 Jede n -stellige Boolesche Funktion ist mittels der zweistelligen Booleschen Funktionen $+$ und \cdot sowie der einstelligen Funktion $\bar{}$ darstellbar.

Die oben angegebenen ein- und zweistelligen Booleschen Funktionen reichen für weitere Betrachtungen also nicht nur völlig aus, sondern liefern bereits wesentlich mehr, als man tatsächlich benötigt. Daher wollen wir für solche „Funktionensysteme“ eine eigene Bezeichnung einführen:

Definition 1.5 Ein System $\mathcal{B} = \{f_1, \dots, f_n\}$ Boolescher Funktionen heißt (funktional) *vollständig*, wenn sich jede Boolesche Funktion allein durch Einsetzungen bzw. Kompositionen von Funktionen aus \mathcal{B} darstellen lässt.

Damit lautet Korollar 1.7 kurz:

Korollar 1.8 $\{+, \cdot, \neg\}$ ist funktional vollständig.

Mit Hilfe der in Satz 1.4 (g) angegebenen de Morganschen Regeln erhält man daraus sofort:

Korollar 1.9 $\{+, \neg\}$ und $\{\cdot, \neg\}$ sind vollständig.

Beweis: $x \cdot y = \overline{\overline{x} + \overline{y}}$, $x + y = \overline{\overline{x} \cdot \overline{y}}$ ∇

Ebenso sieht man leicht ein, dass man in keinem dieser drei vollständigen Systeme auf \neg verzichten kann, da \neg nicht mittels $+$ und \cdot „simulierbar“ ist. Andererseits ist auch \neg nicht vollständig, jedoch werden wir in Abschnitt 1.6 andere einelementige vollständige Systeme kennenlernen (vgl. Satz 1.17).

Bevor wir diese Überlegungen zum Anlaß nehmen wollen, für $+$, \cdot und \neg spezielle Schaltelemente einzuführen, wollen wir noch kurz auf eine zur DNF „duale“ Darstellung Boolescher Funktionen eingehen.

Definition 1.6 Sei i ein Index von $f : B^n \rightarrow B$, und sei m_i der i -te Minterm von f . Dann heißt die Funktion

$$M_i : B^n \rightarrow B,$$

definiert durch

$$M_i(x_1, \dots, x_n) := \overline{m_i(x_1, \dots, x_n)}$$

i -ter *Maxterm* von f .

Wie bei Mintermen lassen wir auch bei Maxtermen die Argumente weg, wenn es der Zusammenhang erlaubt, so dass wir Definition 1.6 auch kurz wie folgt schreiben können: $M_i := \overline{m_i}$. In Beispiel 1.13 sind also z. B.

$$M_3 = \overline{\overline{x_1} \cdot x_2 \cdot x_3} = x_1 + \overline{x_2} + \overline{x_3}$$

$$M_4 = \overline{x_1 \cdot \overline{x_2} \cdot \overline{x_3}} = \overline{x_1} + x_2 + x_3$$

In Analogie zu Mintermen gilt dann: Ein Maxterm M_i nimmt genau dann den Wert 0 an, wenn das Argument $(x_1 \dots x_n)$ die Dualstellung von i ist. Damit beweist man leicht den folgenden Satz:

Satz 1.10 Jede Boolesche Funktion $f : B^n \rightarrow B$ ist eindeutig darstellbar als Produkt der Maxterme ihrer nicht einschlägigen Indizes.

Diese Darstellung heißt auch *konjunktive Normalform* (KNF) von f . In der englischsprachigen Literatur wird die KNF als Sonderfall einer *Product of Sum*-Darstellung, kurz POS, bezeichnet. Für Beispiel 1.13 liefert Satz 1.10:

$$\begin{aligned} f(x_1, x_2, x_3) &= M_0 \cdot M_1 \cdot M_2 \cdot M_4 \cdot M_6 \\ &= (x_1 + x_2 + x_3) \cdot (x_1 + x_2 + \overline{x_3}) \cdot (x_1 + \overline{x_2} + x_3) \\ &\quad \cdot (\overline{x_1} + x_2 + x_3) \cdot (\overline{x_1} + \overline{x_2} + x_3) \end{aligned}$$

Zählt man die in dieser Darstellung vorkommenden Operationen (19) und vergleicht diese Anzahl mit der in der weiter oben für die gleiche Funktion angegebenen DNF vorkommenden (10), so erhält man ein einfaches Kriterium für die Verwendung dieser Normalformen: Offensichtlich ist die DNF zu bevorzugen, wenn die Anzahl der einschlägigen Indizes kleiner ist als die Anzahl der nicht einschlägigen (wie in Beispiel 1.13); ansonsten verwende man die KNF.

1.4 Schaltnetze

Die Bedeutung der letzten Beobachtung wird sofort klar, wenn wir nun für die Operationen $+$, \cdot und $-$ Schaltelemente einführen und damit in der Lage sind, „schwarze Kästen“, welche Boolesche Funktionen berechnen sollen, auszufüllen: Jedes Schaltelement verursacht „Kosten“ (z. B. Materialkosten), und man wird natürlich bestrebt sein, solche Kosten niedrig zu halten.

Die Untersuchung der logischen Eigenschaften von Schaltelementen und von deren Funktion innerhalb größerer Schaltungen ist für den in der Informatik arbeitenden Informatiker von größtem Interesse. Die Herstellungstechnologie kann sich nämlich im Laufe der Entwicklung drastisch ändern. Denn die fortschreitende Miniaturisierung der Schaltungstechnik in Richtung auf die Nano-Technologie mag zu völlig neuen Realisierungsparadigmen für solche Bausteine führen. So könnte z. B. in Schaltungen, die mit Molekülgittern arbeiten, eine Negation oder eine NAND-Funktionalität mit neuartiger Kostenstruktur realisierbar werden, was wesentliche Änderungen im Schaltungsentwurf mit sich bringen würde.

Abbildung 1.3 zeigt gebräuchliche Symbole („Gatter“) zur graphischen Darstellung der Booleschen Negation, Addition und Multiplikation im Zusammenhang mit Schaltungen. Wir zeigen in dieser Abbildung sowohl die von uns im Folgenden verwendeten (einfacheren) Symbole als auch die vom Institute of Electrical and Electronics Engineers (IEEE) vorgeschlagenen (und speziell in der englischsprachigen Literatur häufig verwendeten) Symbole.

Werden solche Bausteine zusammen geschaltet, so spricht man von *Schaltnetzen*. Uns interessieren hier nicht die technischen Einzelheiten solcher Geräte, d. h. insbesondere nicht die Frage nach der Realisierung von Gattern bzw. Schaltnetzen durch Transistoren, Widerstände und Dioden, sondern wir unterstellen, dass sie uns als Bausteine zur Verfügung stehen, und wollen uns daher nur mit ihrem *logischen* Aufbau entsprechend der zugehörigen Booleschen Funktionen beschäftigen.

Die Sätze 1.6 bzw. 1.10 besagen also jetzt, dass jedes Gerät, dessen Arbeitsweise durch eine Boolesche Funktion beschrieben werden kann, durch ein Schaltnetz realisierbar ist, welches nur aus Invertern, Oder- und Und-Gattern besteht. Für Beispiel 1.13, d. h. die Funktion

$$f(x_1, x_2, x_3) = \bar{x}_1 x_2 x_3 + x_1 \bar{x}_2 x_3 + x_1 x_2 \bar{x}_3,$$

sieht ein solches „Gerät“ wie in Abbildung 1.4 gezeigt aus.

Damit haben wir erstmalig eine „Black Box“ ausgefüllt, indem wir ein Schaltnetz für die sie beschreibende Boolesche Funktion angegeben haben. Zur Vereinfachung solcher Zeichnungen werden im Allgemeinen Inverter direkt vor den entsprechenden Eingang eines nachfolgenden Und- oder Oder-Gatters gesetzt („integriert“), wie in

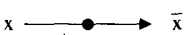
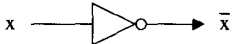
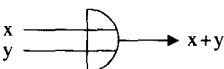

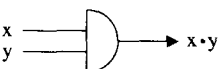

Funktion	Unser Symbol	IEEE-Symbol
Negation (Komplement-Gatter)		
Addition (Oder-Gatter)		
Multiplikation (Und-Gatter)		

Abbildung 1.3: Grundbausteine zur Realisierung Boolescher Funktionen.

Abbildung 1.5 angedeutet. Außerdem verwendet man auch Und- und Oder-Gatter mit mehr als zwei Eingängen. Das in Abbildung 1.4 gezeigte Schaltnetz würde damit die in Abbildung 1.6 gezeigte Form erhalten.

Von diesen Vereinfachungsmöglichkeiten werden wir im Folgenden verschiedentlich Gebrauch machen. Kommen wir jedoch auf das erste, für dieses f angegebene Schaltnetz zurück: Jedem Operator, der in der DNF von f vorkommt, entspricht dort genau ein Gatter; insgesamt besteht das Schaltnetz aus 10 Gattern. Die in Beispiel 1.10 bereits angesprochene „Schaltzeit“ eines solchen „Moduls“ lässt sich nun besser als oben demonstrieren: Unter der Annahme, dass jedes Gatter nach $10 \text{ psec} = 10^{-11} \text{ sec}$ „geschaltet“ hat, d. h. dass 10^{-11} sec nach Anlegen eines Inputs der entsprechende Output eines Gatters vorliegt, und infolge der Tatsache, dass jeder dreistellige Input 5 „Stufen“ zu durchlaufen hat, liegt ein Funktionswert somit nach $5 \cdot 10^{-11} \text{ sec}$ vor. Dies ist insofern idealisiert, als wir dabei die Zeit vernachlässigt haben, die ein Signal benötigt, einen Leitungsweg (von einem Input oder einem Gatter zu einem anderen Gatter) zurückzulegen. Für unsere Betrachtungen kann dieser Fehler in Kauf genommen werden, da er prinzipiell keine Veränderung des Ergebnisses bewirkt. Es sei jedoch an dieser Stelle darauf hingewiesen, dass in der Praxis die Länge von Verbindungsdrähten durchaus eine Rolle spielt. Ein Signal kann eine Leitung höchstens mit Lichtgeschwindigkeit durchlaufen, d. h. mit einer Geschwindigkeit von

$$3 \cdot 10^5 \frac{\text{km}}{\text{sec}} = 0,3 \cdot 10^{12} \frac{\text{mm}}{\text{sec}} = 0,3 \frac{\text{mm}}{\text{psec}}.$$

Deshalb können sich z. B. Signale innerhalb einer Picosekunde grundsätzlich nicht über einen Chip mit einer Längenausdehnung von 1 mm ausbreiten.

Als nächstes wollen wir eine formale Definition eines Schaltnetzes angeben. Dazu kommen wir zurück auf den weiter oben eingeführten Begriff des Graphen, den wir nun durch Auszeichnung von Richtungen wie folgt abändern: Sei P eine endliche

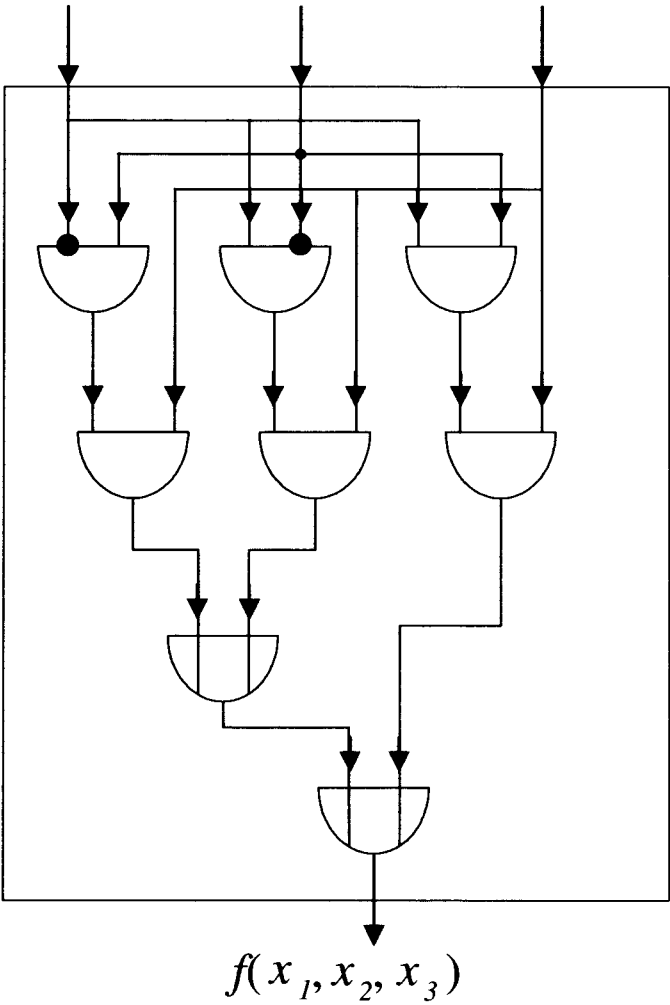


Abbildung 1.4: Schaltung für die Boolesche Funktion aus Beispiel 1.13.



Abbildung 1.5: Alternative Darstellung von Invertieren.

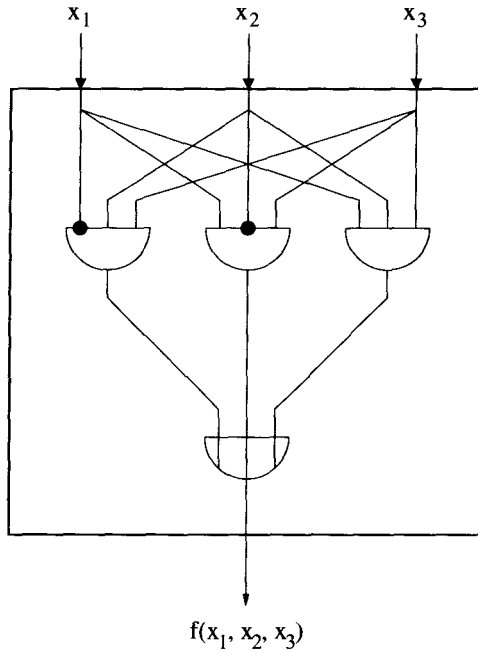


Abbildung 1.6: Alternative Schaltung für die Funktion aus Beispiel 1.13.

Punktmenge und $K \subseteq P \times P$ eine (beliebige) Relation über P . Dann heißt $G := (P, K)$ ein *gerichteter Graph* mit der Punktmenge P und der Menge K von gerichteten Kanten. Die weiter oben gegebene Definition eines Weges in G kann offensichtlich ohne Änderungen auf gerichtete Graphen übertragen werden, und wir bezeichnen einen Weg als *Kreis* oder *Zykel*, falls sein Anfangs- und Endpunkt übereinstimmen.

Definition 1.7 Ein *Schaltnetz* ist ein gerichteter, zyklfreier Graph (engl.: *Directed Acyclic Graph*; kurz: DAG).

Als *Input* (eines Schaltnetzes) bezeichnet man die Punkte in einem DAG, in die keine Kante hineinführt, und entsprechend als *Output* die Punkte, aus denen keine Kante herausführt.

Das erste der oben angegebenen Schaltnetze für die Boolesche Funktion aus Beispiel 1.13 (vgl. Abbildung 1.4) entspricht dem in Abbildung 1.7 gezeigten DAG. Von jedem Input gehen in diesem Beispiel drei Kanten aus, was in den bisher gezeigten Schaltnetzen der Auffächerung eines jeden x_i in drei Signale entspricht. Die hierfür übliche Bezeichnung ist *Fan-Out*: Jeder Input hat (hier) einen Fan-Out von 3, und kein anderer Punkt (bzw. Gatter) hat einen Fan-Out (was jedoch unter Umständen durchaus zugelassen ist).

Markiert man in einem solchen DAG – wie oben geschehen – alle Punkte, welche nicht Input sind, mit den Symbolen der ihnen zugeordneten Booleschen Funktion, so spricht man von einem *Operator-Schaltnetz*, anderenfalls von einem *Verbindungsnetz*.

Warum man in Definition 1.7 Zyklfreiheit fordert, soll das durch Abbildung 1.8 illustrierte Beispiel erläutern: Die dort gezeigte simple Schaltung enthält einen Zykel,

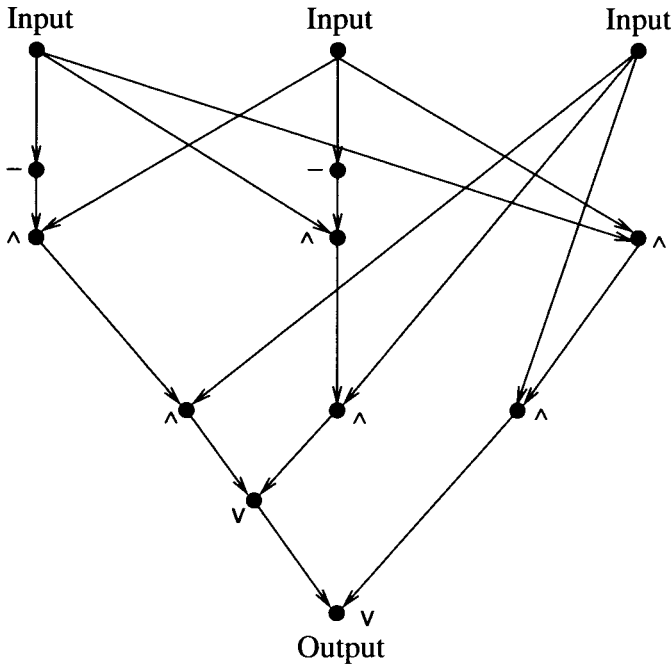


Abbildung 1.7: DAG zum Schaltnetz aus Abbildung 1.4.

d. h. einen Kantenzug, der in sich zurückläuft. Sei $x = 0$. Liegt nun in der rechten Schleife eine 0 an, erhält z den Wert 1; dann liegt aber am rechten Eingang diese 1 an und z erhält den Wert 0 usw., d. h. die Schaltung zeigt ein instabiles Verhalten (man spricht auch von einer „Flimmerschaltung“) und ist damit für die Praxis unbrauchbar.

Satz 1.11 Jeder (nichtleere) DAG (mit endlich vielen Punkten) hat mindestens einen Input und mindestens einen Output.

Beweis: Sei $G = (P, K)$ ein DAG mit $P \neq \emptyset$, $|P| < \infty$. Angenommen, G hat keinen Input. Sei dann p_1 ein beliebiger Punkt von G , so hat dann p_1 (mindestens) einen Vorgänger p_2 . p_2 hat wiederum (mindestens) einen Vorgänger p_3 usw. Da G endlich ist und laut Annahme in jeden Punkt (mindestens) eine Kante hineinführt, gilt irgendwann $p_i = p_j$ für $i < j$, d. h. es liegt die in Abbildung 1.9 gezeigte Situation vor. Damit enthält G also einen Zykel, im Widerspruch zur Voraussetzung. Völlig analog zeigt man durch Konstruktion eines Nachfolgerzykels, dass G (mindestens) einen Output besitzt. ∇

Wir bemerken bereits an dieser Stelle, dass Schaltnetze heute als Chips oder integrierte Schaltungen realisiert werden, bei welchen große Anzahlen von Gattern und Verbindungsleitungen zwischen diesen aus Halbleitern auf einer sehr kleinen Fläche konstruiert werden. Die dabei verwendeten Konstruktionsmethoden und Technologien erlegen dem Entwurf effizienter Schaltungen eine Reihe von Beschränkungen auf, auf welche wir kurz eingehen wollen:

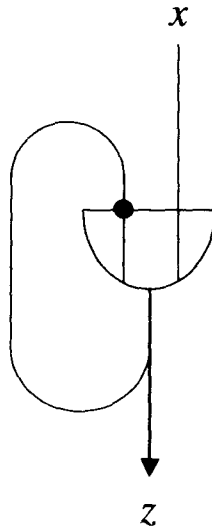


Abbildung 1.8: Flimmerschaltung.

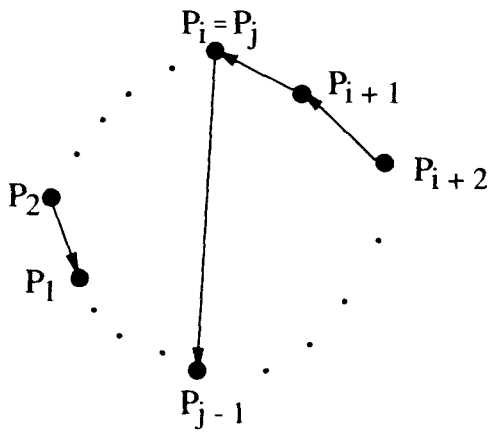


Abbildung 1.9: Zum Beweis von Satz 1.11.

1. Geschwindigkeit:

Jedes Gatter hat, wie bereits erwähnt, eine gewisse *Verzögerung* bzw. Schaltzeit, welche zwischen der Aktivierung der Inputs und dem Bereitstehen des Outputs vergeht. Diese Zeit kann für einzelne Gatter im Bereich weniger Picosekunden liegen; die Verzögerung eines Schaltnetzes hängt jedoch davon ab, wie viele Stufen von Gattern Inputsignale insgesamt zu durchlaufen haben. Eine dreistufige Schaltung ist daher im Allgemeinen langsamer als eine zweistufige. Generell wird man versuchen, Schaltungen mit möglichst wenigen Stufen zu realisieren.

2. Größe:

Die Herstellungskosten eines Schaltnetzes sind im Wesentlichen proportional zur Anzahl der verwendeten Gatter, so dass eine geringe Gatteranzahl erstrebenswert ist. Diese Anzahl wiederum beeinflusst die Chip-Fläche und damit die Schaltgeschwindigkeit: Kleine Schaltungen sind i. A. schneller als große. Je größer ein Chip wird, desto höher ist auch die Wahrscheinlichkeit, dass sich bei seiner Herstellung Produktionsfehler einschleichen; außerdem erfordern große Chips längere Verbindungen zwischen ihren Schaltelementen.

3. Fan-In/Out:

Wie oben erwähnt, heißt die Anzahl der Inputs, mit denen der Output eines Gatters verbunden ist, *Fan-Out*; analog heißt die Anzahl der Inputs eines Gatters *Fan-In*. Gatter mit hohem Fan-In oder hohem Fan-Out sind im Allgemeinen langsamer als solche mit geringem Fan-In/Out. Zu bevorzugen sind also Schaltungen, deren Gatter einen kleinen Fan-In/Out haben. Außerdem muss man eventuell einen sehr hohen Fan-Out durch „Treiberschaltungen“ verstärken.

Die gerade genannten Entwurfsziele großer Schaltungen (möglichst wenig Stufen, möglichst wenig Gatter bzw. kleine Fläche, geringer Fan-In/Out) sind nicht immer gleichzeitig erreichbar.

1.5 Körpersummendarstellung Boolescher Funktionen

Weiter oben haben wir (funktional) vollständige Systeme von Booleschen Funktionen kennen gelernt (Korollare 1.8. 1.9), welche auf die Negation nicht verzichten konnten. Wir werden nun ein System angeben, welches ohne diese auskommt. Dazu kommen wir wieder auf die DNF-Darstellung einer Booleschen Funktion zurück: Sei $f : B^n \rightarrow B$, und sei I die Menge der einschlägigen Indizes von f , so gilt nach Satz 1.6:

$$f = \sum_{i \in I} m_i.$$

Wir haben bereits bemerkt, dass ein Minterm m_i genau dann den Wert 1 annimmt, wenn sein Argument die Dualdarstellung von i liefert. Daraus erhält man:

Satz 1.12 (*Körpersummen-Normalform*) Sei $f : B^n \rightarrow B$ und $I = \{\alpha_1, \dots, \alpha_k\}$ die Menge der einschlägigen Indizes von f . Dann gilt:

$$f = m_{\alpha_1} \uplus m_{\alpha_2} \uplus \dots \uplus m_{\alpha_k}.$$

Dabei ist \oplus die aus Beispiel 1.12 bekannte Funktion f_6 , die Antivalenz oder XOR, welche man früher auch als *Ringsumme* bezeichnet hat.

Der Beweis wird in die Übungen verwiesen (vgl. Aufgabe 1.12). Ein tieferes Verständnis gewinnt man mit der Beobachtung aus Abschnitt 1.2, dass man sich mit der Antivalenz \oplus und der Konjunktion \cdot algebraisch in einem *Körper* mit den einzigen Elementen 0 und 1 befindet und dass wir mit Satz 1.12 nichts anderes als einen Darstellungssatz für beliebige Funktionen über diesem Körper beweisen. Es sind stets Polynome!

Korollar 1.13 $\{\oplus, \cdot, \neg\}$ ist funktional vollständig.

Bevor wir ein vollständiges System ohne Negation angeben, stellen wir einige Eigenschaften der Ringsumme ohne Beweis zusammen:

Satz 1.14 Für alle $x, y, z \in B$ gilt:

- (a) $x \oplus 1 = \bar{x}$, $x \oplus 0 = x$
- (b) $x \oplus x = 0$, $x \oplus \bar{x} = 1$
- (c) $x \oplus y = y \oplus x$ (Kommutativität)
- (d) $x \oplus (y \oplus z) = (x \oplus y) \oplus z$ (Assoziativität)
- (e) $x \cdot (y \oplus z) = x \cdot y \oplus x \cdot z$ (Distributivität bzgl. \cdot)
- (f) $0 \oplus 0 \oplus \dots \oplus 0 = 0$
- (g) $\underbrace{1 \oplus 1 \oplus \dots \oplus 1}_{n\text{-mal}} = \begin{cases} 1 & \text{falls } n \text{ ungerade} \\ 0 & \text{falls } n \text{ gerade} \end{cases}$

Für den Beweis des folgenden Satzes vergleiche man Aufgabe 1.21:

Satz 1.15 (*Komplementfreie Ringsummenentwicklung nach Reed und Muller 1954*) Jede Boolesche Funktion $f : B^n \rightarrow B$ ist eindeutig darstellbar als Polynom (Multiplikation) in den Variablen x_1, \dots, x_n mit Koeffizienten $a_0, \dots, a_{1\dots n} \in B$ wie folgt:

$$\begin{aligned}
 f = & a_0 \\
 & \oplus a_1 x_1 \oplus a_2 x_2 \oplus \dots \oplus a_n x_n \\
 & \oplus a_{12} x_1 x_2 \oplus \dots \oplus a_{n-1,n} x_{n-1} x_n \\
 & \vdots \\
 & \oplus a_{1\dots n} x_1 x_2 \dots x_n.
 \end{aligned}$$

Beispiel 1.13 (Fortsetzung) An diesem mittlerweile bekannten Beispiel vollziehen wir die in Teil (a) des Beweises angegebene Konstruktion nach:

$$\begin{aligned}
 f &= \bar{x}_1 x_2 x_3 + x_1 \bar{x}_2 x_3 + x_1 x_2 x_3 \\
 &= \bar{x}_1 x_2 x_3 \oplus x_1 \bar{x}_2 x_3 \oplus x_1 x_2 x_3 \\
 &= (x_1 \oplus 1) x_2 x_3 \oplus x_1 (x_2 \oplus 1) x_3 \oplus x_1 x_2 x_3 \\
 &= x_1 x_2 x_3 \oplus x_2 x_3 \oplus x_1 x_2 x_3 \oplus x_1 x_3 \oplus x_1 x_2 x_3 \\
 &= x_1 x_3 \oplus x_2 x_3 \oplus x_1 x_2 x_3
 \end{aligned}$$

Also gilt:

$$a_0 = 0 = a_1 = a_2 = a_3, \quad a_{12} = 0, \quad a_{13} = a_{23} = 1, \quad a_{123} = 1.$$

□

Als Korollar aus Satz 1.15 erhalten wir das weiter oben angekündigte funktional vollständige System Boolescher Funktionen ohne Negation:

Korollar 1.16 $\{\uparrow, \cdot, 1\}$ ist funktional vollständig.

1.6 NAND- und NOR-Darstellungen

Wir setzen zunächst unsere Betrachtungen zur funktionalen Vollständigkeit fort und betrachten noch einmal die in Beispiel 1.12 angegebenen Funktionstabellen für die zweistelligen Booleschen Funktionen NAND (f_{14} , Shefferscher Strich, \uparrow) und NOR (f_8 , Peircescher Pfeil, \downarrow). Es gilt:

Satz 1.17 $\{\uparrow\}$ und $\{\downarrow\}$ sind funktional vollständig.

Beweis: Wir führen den Beweis nur für „ \uparrow “; für „ \downarrow “ verläuft er völlig analog (vgl. Aufgabe 1.22). Zu zeigen ist, dass *jede* Boolesche Funktion allein durch NAND dargestellt werden kann. Da nach Korollar 1.9 bereits $\{+, \cdot\}$ als vollständig bekannt ist, reicht es, für diese Funktionen NAND-Darstellungen anzugeben:

$$\begin{aligned} \bar{x} &= \bar{x} + \bar{x} = \overline{x \cdot x} \\ &= x \uparrow x \end{aligned}$$

$$\begin{aligned} x + y &= \overline{\overline{x + y}} = \overline{\overline{x} \cdot \overline{y}} \\ &= \overline{\overline{x} \cdot \overline{x} \cdot \overline{y} \cdot \overline{y}} \\ &= (x \uparrow x) \uparrow (y \uparrow y) \quad \nabla \end{aligned}$$

Korollar 1.18 Als neue „Grundbausteine“ erhalten wir die in Abbildung 1.10 gezeigten NAND- bzw. NOR-Gatter.

Einer dieser beiden Bausteine reicht also prinzipiell aus, um *jede* Boolesche Funktion durch eine Schaltung zu realisieren. Wir wollen diese Tatsache als nächstes ausnutzen, um Schaltungssynthese allein mit NAND- oder NOR-Gattern zu betreiben. Dies geschieht vor dem Hintergrund, dass diese beiden Bausteine elektrisch einfach zu realisieren sind und damit für die Praxis eine hohe Bedeutung haben.

Wir beschreiben exemplarisch einen einfachen Weg der Herstellung von Schaltungen, die nur aus NAND- oder nur aus NOR-Gattern bestehen. Dazu gehen wir von einer Schaltung aus, die einer SOP-Darstellung entspricht, und konvertieren diese schrittweise in eine äquivalente Schaltung mit lediglich NAND-Gattern.

Beispiel 1.14 $f : B^3 \rightarrow B$ sei die in Tabelle 1.6 gezeigte Boolesche Funktion. Für diese Funktion gilt:

$$f(x_1, x_2, x_3) = \bar{x}_1 \bar{x}_2 x_3 + x_1 \bar{x}_2 \bar{x}_3 + x_1 \bar{x}_2 x_3 + x_1 x_2 \bar{x}_3$$

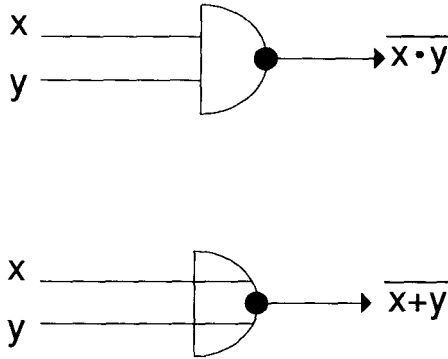


Abbildung 1.10: NAND- (oben) und NOR-Gatter (unten).

Tabelle 1.6: Boolesche Funktion zu Beispiel 1.14.

i	x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Diese DNF-Darstellung lässt sich mit Techniken, die wir in Kapitel 3 im Einzelnen behandeln werden, vereinfachen. Wir beschreiben hier eine unmittelbare Anwendung der Gesetze der Booleschen Algebra:

$$\begin{aligned}
 f(x_1, x_2, x_3) &= (\bar{x}_1 + x_1)\bar{x}_2x_3 + x_1\bar{x}_3(\bar{x}_2 + x_2) \\
 &= \bar{x}_2x_3 + x_1\bar{x}_3
 \end{aligned}$$

Für diese Darstellung ist eine entsprechende Schaltung in Abbildung 1.11 gezeigt.

Diese Darstellung bildet auf der ersten Ebene Und-Verknüpfungen, auf der zweiten Ebene Oder-Verknüpfungen. Wir bringen nun an den Ausgängen der Und-Ebene und an den entsprechenden Eingängen der Oder-Ebene Inverter an wie in Abbildung 1.12 (links) gezeigt. Offensichtlich wird das von der Schaltung berechnete Ergebnis dadurch nicht verändert, denn zwei aufeinander folgende Invertierungen heben sich auf. Die Und-Ebene besteht jetzt bereits aus NAND-Gattern, das bzw. die Gatter der Oder-Ebene wird bzw. werden durch NAND-Gatter ersetzt (was auf Grund der Rechenregeln erlaubt ist). Schließlich werden die noch vor der Und-Ebene liegenden Inverter gemäß der Konstruktion im Beweis von Satz 1.17 ersetzt. Die dadurch resultierende Schaltung, die nur noch aus NAND-Gattern besteht, ist in Abbildung 1.12 (rechts) gezeigt. \square

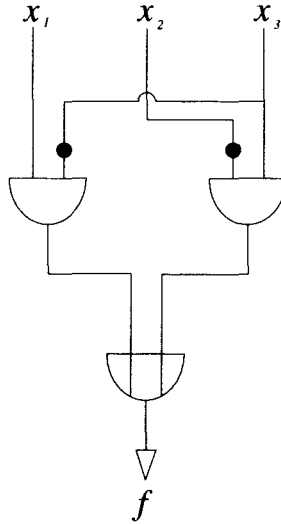


Abbildung 1.11: SOP-Darstellung zu Beispiel 1.14.

Das Beispiel zeigt insbesondere, wie sich aus einer SOP-Darstellung, die allein mit *zweistelligen* Gattern realisiert wurde, eine äquivalente NAND-Darstellung gewinnen lassen kann. Wir zeigen als nächstes eine völlig analoge Konstruktion zur Umwandlung einer POS-Darstellung in eine äquivalente Darstellung allein aus NOR-Gattern.

Beispiel 1.15 $f : B^3 \rightarrow B$ sei die Funktion aus dem letzten Beispiel, deren Funktionswerte in Tabelle 1.6 gegeben sind. Wir betrachten jetzt die KNF-Darstellung dieser Funktion, die sich wiederum unter Anwendung der Rechenregeln der Booleschen Algebra vereinfachen und sodann in eine NOR-Darstellung umrechnen lässt:

$$\begin{aligned}
 f(x_1, x_2, x_3) &= M_0 \cdot M_2 \cdot M_3 \cdot M_7 \\
 &= (x_1 + x_3)((\bar{x}_2 + \bar{x}_3)) \\
 &= \overline{\overline{(x_1 + x_3)}(\overline{\bar{x}_2 + \bar{x}_3})} \\
 &= \overline{\overline{(x_1 + x_3)} + \overline{(\bar{x}_2 + \bar{x}_3)}} \\
 &= \overline{(x_1 + x_3) \downarrow \overline{(\bar{x}_2 + \bar{x}_3)}} \\
 &= (x_1 \downarrow x_3) \downarrow (\bar{x}_2 \downarrow \bar{x}_3) \\
 &= (x_1 \downarrow x_3) \downarrow ((x_2 \downarrow x_2) \downarrow (x_3 \downarrow x_3))
 \end{aligned}$$

Für die oben gezeigte POS-Darstellung ist eine entsprechende Schaltung in Abbildung 1.13 (oben) gezeigt.

Diese Darstellung bildet jetzt auf der ersten Ebene Oder-Verknüpfungen, auf der zweiten Ebene Und-Verknüpfungen. Wir bringen wieder an den Ausgängen der ersten und an den entsprechenden Eingängen der zweiten-Ebene Inverter an wie in

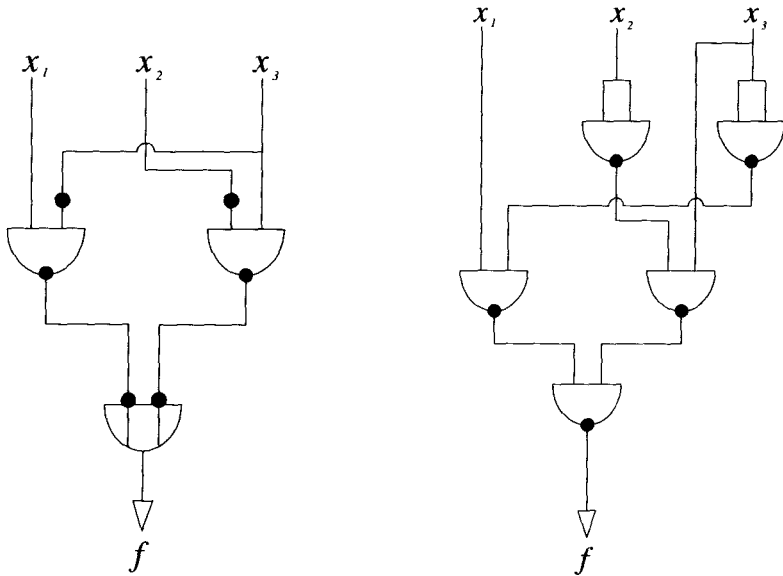


Abbildung 1.12: SOP-Darstellung aus Abbildung 1.11 mit zusätzlichen Invertern (links); äquivalente NAND-Darstellung der Funktion (rechts).

Abbildung 1.13 (unten links) gezeigt. Die Oder-Ebene besteht jetzt bereits aus NOR-Gattern, das bzw. die Gatter der Und-Ebene wird bzw. werden durch NOR-Gatter ersetzt. Schließlich werden wieder die noch vor der Oder-Ebene liegenden Inverter durch NOR-Gatter ersetzt. Die dadurch resultierende Schaltung, die nur noch aus NOR-Gattern besteht, ist in Abbildung 1.13 (unten rechts) gezeigt. \square

Wir bemerken abschließend, dass NAND- und NOR-Gatter auch mit mehr als zwei Inputs versehen werden können, denn offensichtlich gilt:

$$\begin{aligned} x_1 \uparrow x_2 \uparrow \dots \uparrow x_n &= \overline{\overline{x_1} \overline{x_2} \dots \overline{x_n}} = \overline{x_1} + \overline{x_2} + \dots + \overline{x_n} \\ x_1 \downarrow x_2 \downarrow \dots \downarrow x_n &= \overline{x_1 + x_2 + \dots + x_n} = \overline{x_1} \overline{x_2} \dots \overline{x_n} \end{aligned}$$

Allerdings ist zu beachten, dass für NAND und NOR im Unterschied zu AND und OR *kein* Assoziativgesetz gilt (vgl. Aufgabe 1.24, so dass man strenggenommen anstelle von $x_1 \uparrow x_2 \uparrow \dots \uparrow x_n$ z. B. $\uparrow (x_1, \dots, x_n)$ schreiben sollte). Man kann also z. B. nicht ein dreistelliges NAND-Gatter aus zwei zweistelligen NAND-Gattern aufbauen. Die Realisierung von NAND- bzw. NOR-Gattern mit mehr als zwei Inputs ist daher technisch aufwendiger als bei AND- bzw. OR-Gattern, weshalb wir uns in den beiden letzten Beispielen auf zweistellige Gatter beschränkt haben.

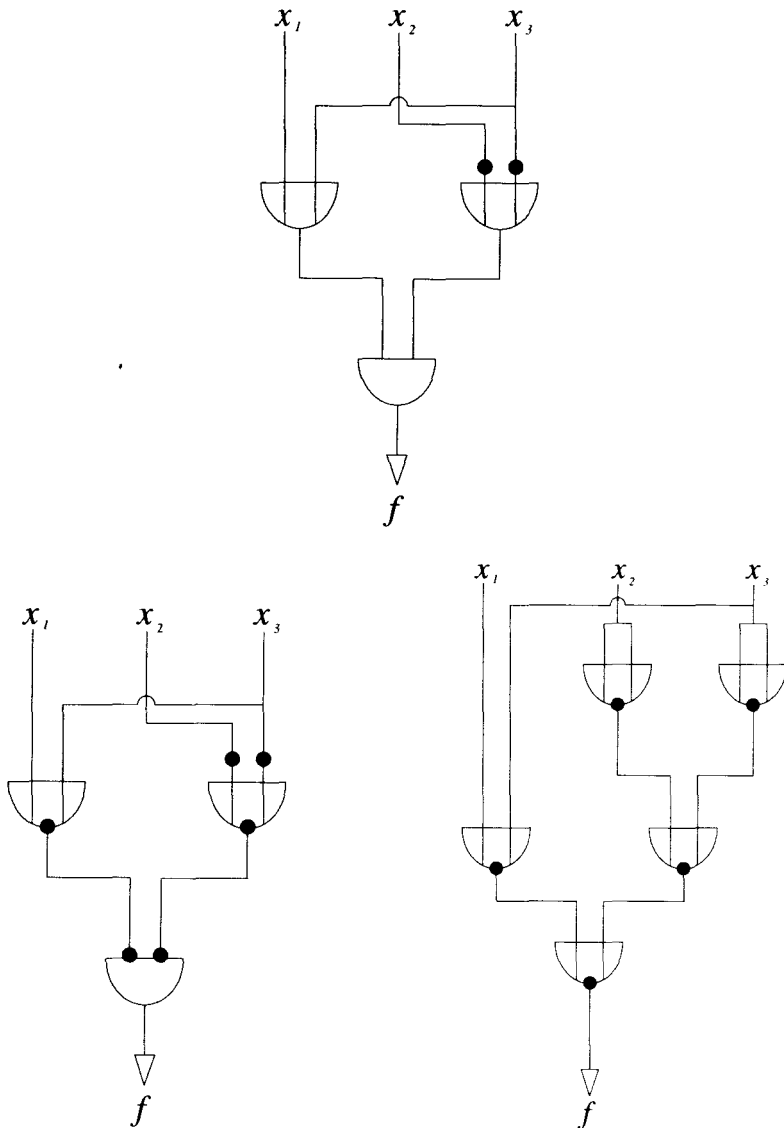


Abbildung 1.13: POS-Darstellung zu Beispiel 1.15 (oben); POS-Darstellung mit zusätzlichen Invertiern (unten links); äquivalente NOR-Darstellung der Funktion (unten rechts).