Einführung in die Boolesche Algebra

Hans Jürgen Ohlbach

Keywords: Boolesche Algebra, Boolesche Funktionen, Boolesches Axiomensystem, Rechenregeln, Normalformen, SAT-Solving, Mengenalgebra

Inhaltsverzeichnis

Diese Einführung in die Boolesche Algebra[1] richtet sich in erster Linie an Studierende, die Informatik als Nebenfach studieren.

[1] Unter einer *Algebra* versteht man in der Mathematik eine nichtleere Menge zusammen mit einer fest definierten Menge von Funktionen auf dieser Menge, die bestimmten Gesetzen genügen.

1 Einleitung

Unsere heutigen Computer arbeiten intern mit Bitfolgen. Bitfolgen bestehen aus Nullen und Einsen. Um zu verstehen, wie Computer mit Bitfolgen arbeiten, muss man zunächst die Rechenvorschriften (den *Kalkül*) für den Umgang mit 0 und 1, den Bits, verstehen. Diese Rechenvorschriften wurden schon 1847 von dem englischen Mathematiker George Bool in dessen *Logikkalkül* entwickelt. Er interessierte sich natürlich nicht für die Nullen und Einsen im Computer, die gab es damals ja noch nicht, sondern für den Umgang mit den Wahrheitswerten *falsch* und *wahr*. Es zeigte sich aber, dass der Umgang mit den Wahrheitswerten genau denselben Gesetzen folgt wie der Umgang mit 0 und 1 im Computer. Daher kann man heute 0 und *falsch* sowie 1 und *wahr* synonym benutzen. Für den Kalkül benutzen allerdings die Techniker eher den Namen *Boolesche Algebra*, und die Logiker eher den Namen *Aussagenlogik*. Obwohl es im Prinzip das gleiche ist, sind für Techniker andere Aspekte des Kalküls wichtig, als für Logiker. Daher findet man oft ganz unterschiedliche Darstellungen der Theorie. In diesem Text konzentrieren wir uns eher auf die Aspekte, die für Techniker wichtig sind, und sprechen daher von *Boolescher Algebra*, und *Bits* 0,1, statt Wahrheitswerten.

Boolesche Algebra bildet die theoretische Basis für den Entwurf digitaler Schaltungen. Als Aussagenlogik ist sie der Ausgangspunkt von unterschiedlichen Formalismen der Wissensrepräsentation. Auch in konkreten Programmen kommen Boolesche Ausdrücke vor, z.B. als Bedingungen in if-Konstrukten oder while-Schleifen. Selbst die Mengenlehre mit den Operationen Vereinigung, Durchschnitt und Komplement bildet eine Boolesche Algebra. Es gibt daher viele Gründe, sich mit Boolescher Algebra vertraut zu machen.

2 Boolesche Funktionen

Boolesche Funktionen arbeiten zunächst auf einzelnen Bits, nicht auf Bitfolgen. Der Umgang mit Bitfolgen ist nicht Thema dieses Textes. Da es nur die beiden Bits 0 und 1 gibt, ist die Menge von Funktionen auf diesen Bits recht übersichtlich. Jede dieser Funktionen lassen sich durch *Wahrheitstabellen* beschreiben. Eine Wahrheitstabelle gibt für jede Kombination von Argumenten (eben 0 und 1) den Funktionswert an.

Einstellig Boolesche Funktionen: Es gibt nur *vier verschiedene* einstellige Funktionen, die durch folgende Wahrheitstabellen beschrieben werden:

	g_1	g_2	g_3	g_4
0	0	0	1	1
1	0	1	0	1

g_1 ist nichts anderes als die Konstante 0, g_4 ist die Konstante 1. g_2 ist die *Identitätsfunktion*, die einfach das Argument zurück gibt, und g_3 ist die *Negation*, die das Argument umdreht.

Im folgenden benutzen wir für die Identitätsfunktion den Namen id, und für die Negationsfunktion das Zeichen $\neg$: $\neg 0 = 1, \neg 1 = 0$.

Andere Schreibweisen für die Negation eines Wertes x sind $-x$, oder $\overline{x}$.

Zweistellig Boolesche Funktionen:
Für zweistellige Boolesche Funktionen gibt es insgesamt $2^{2^2} = 16$ Möglichkeiten $f_i(x, y)$:

x	y	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Man kann sich jetzt leicht überlegen, dass es dann $2^{2^3} = 256$ dreistellige Boolesche Funktionen gibt, oder generell 2^{2^n} n-stellige Boolesche Funktionen.

Zum Glück braucht man nur ganz wenige davon. Die Restlichen lassen sich mit Hilfe dieser wenigen definieren. Wie man aus der Wahrheitstabelle sieht, ist f_1 wieder die Konstante 0, und f_{16} ist die Konstante 1. Die wichtigeren der 16 Funktionen sind f_2, f_7, f_8, f_{10} und f_{14}. Sie sind unter bestimmten Namen bekannt:

x	y	f_2 = und	f_7 = xor	f_8 = oder	$f_{10} = \Leftrightarrow$	$f_{14} = \Rightarrow$
0	0	0	0	0	1	1
0	1	0	1	1	0	1
1	0	0	1	1	0	0
1	1	1	0	1	1	1

In technischen und Logiktexten und auch in Computerprogrammen benutzt man allerdings noch andere Notationen für diese Funktionen:

Funktion	technisch	logisch	Programme	Bedeutung
und	.	$\wedge$	&	logisches und
oder	+	$\vee$	|	logisches oder
xor	^	xor	^	exklusives oder
$\Leftrightarrow$	=	$\Leftrightarrow$	==	Äquivalenz
$\Rightarrow$	$\Rightarrow$	$\Rightarrow$		Implikation

Im folgenden bleiben wir bei der logischen Notation, d.h. $und = \wedge$, $oder = \vee$, sowie $nicht = \neg$. Die Syntaktische Darstellung dieser Funktionen wird auch oft als *Junktoren* bezeichnet.

Ob man f_{14} tatsächlich als *Implikation* $\Rightarrow$ bezeichnen sollte, kann diskutiert werden. $x \Rightarrow y$ ist nämlich immer dann 1, wenn $x = 0$ ist, oder in logischer Sprechweise $x \Rightarrow y$ ist immer dann wahr, wenn x falsch ist. Eine Aussage „der Mond ist aus grünem Käse $\Rightarrow 1 + 1 = 3$“ ist dann eine wahre Aussage. Allerdings bietet sich auch keine andere der 16 Funktionen als besserer Kandidat für eine Implikation an. Diese Unzufriedenheit hat zur Entwicklung andere Logiken geführt (intuitionistische Logik, Relevanzlogik usw.), die nicht auf Boolescher Algebra basieren.

Es zeigt sich, dass man neben der einstelligen Negation nur die Funktionen und und $oder$ braucht. Alle anderen lassen sich damit definieren.

$$\begin{array}{ll}
f_3(x,y) & = x \wedge \neg y \\
f_4(x,y) & = x \\
f_5(x,y) & = \neg x \wedge y \\
f_6(x,y) & = y \\
f_7(x,y) = x \; xor \; y & = (x \wedge \neg y) \vee (\neg x \wedge y) \\
f_9(x,y) & = \neg(x \vee y) \\
f_{10}(x,y) = x \Leftrightarrow y & = (x \wedge y) \vee (\neg x \wedge \neg y) \\
f_{11}(x,y) & = \neg y \\
f_{12}(x,y) & = x \vee \neg y \\
f_{13}(x,y) & = \neg x \\
f_{14}(x,y) = x \Rightarrow y & = \neg x \vee y \\
f_{15}(x,y) & = \neg(x \wedge y)
\end{array}$$

Alle drei- und mehrstelligen Booleschen Funktionen lassen sich auf ähnliche Weise ausschließlich mit Hilfe $\neg, \wedge, \vee$ bilden. Diese drei Booleschen Funktionen nennt man daher *funktional vollständig*.

Die Menge der Booleschen Funktionen, mit denen man alle anderen definieren kann, lässt sich sogar noch weiter reduzieren. Es zeigt sich, dass als einzige Funktion entweder $f_9(x,y) = \neg(x \vee y)$ (negiertes $oder$ oder nor), oder $f_{15}(x,y) = \neg(x \wedge y)$ (negiertes und oder $nand$) ausreichen, um alle anderen Funktionen zu definieren. Um das zu zeigen, genügt es $\neg, \wedge, \vee$ entweder mit Hilfe von $nand$ oder mit Hilfe von nor zu definieren.

Wir schreiben dabei $x \; nand \; y$ für $\neg(x \wedge y)$ und $x \; nor \; y$ für $\neg(x \vee y)$.
Indem man die Gleichheiten
$(x \wedge y) = \neg\neg(x \wedge y) = \neg(x \; nand \; y)$ und
$(x \vee y) = \neg\neg(x \vee y) = \neg(\neg x \wedge \neg y) = \neg x \; nand \; \neg y$
ausnutzt, ergibt sich:

$nand$-Definitionen:

$$\begin{array}{lcl}
\neg x & = x \; nand \; x & \\
x \wedge y = \neg(x \; nand \; y) & = & (x \; nand \; y) \; nand \; (x \; nand \; y) \\
x \vee y = \neg x \; nand \; \neg y & = & (x \; nand \; x) \; nand \; (y \; nand \; y)
\end{array}$$

Analog erhält man die nor-Definitionen:

$$\begin{array}{lcl}
\neg x & = x \; nor \; x & \\
x \vee y = \neg(x \; nor \; y) & = & (x \; nor \; y) \; nor \; (x \; nor \; y) \\
x \wedge y = \neg x \; nor \; \neg y & = & (x \; nor \; x) \; nor \; (y \; nor \; y)
\end{array}$$

$nand$ nennt man daher *minimal funktional vollständig*. nor ist ebenfalls *minimal funktional vollständig*.

Diese Erkenntnisse haben sehr praktischen Nutzen. Sie bedeuten, dass man alle Booleschen Funktionen technisch als Schaltkreise realisieren kann, die entweder ausschließlich mit Bausteinen für $\neg, \wedge, \vee$ auskommen, oder sogar noch weniger, ausschließlich mit $nand$-Bausteinen, oder ausschließlich mit nor-Bausteinen.

3 Axiome der Booleschen Algebra

Die Funktionen $\neg, \wedge, \vee$ genügen ganz bestimmten Gesetzen, die der Mathematiker Peano als folgendes *Axiomensystem* zusammengefasst hat:

Name	Gesetz	Duales Gesetz
Kommutativgesetze	$x \wedge y = y \wedge x$	$x \vee y = y \vee x$
Assoziativgesetze	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$	$x \vee (y \vee z) = (x \vee y) \vee z$
Idempotenzgesetze	$x \wedge x = x$	$x \vee x = x$
Distributivgesetze	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
Neutralitätsgesetze	$x \wedge 1 = x$	$x \vee 0 = x$
Extremalgesetze	$x \wedge 0 = 0$	$x \vee 1 = 1$
Doppelnegation (Involution)	$\neg\neg x = x$	
De Morgansche Gesetze	$\neg(x \wedge y) = \neg x \vee \neg y$	$\neg(x \vee y) = \neg x \wedge \neg y$
Komplementärgesetze	$x \wedge \neg x = 0$	$x \vee \neg x = 1$
Dualitätsgesetze	$\neg 0 = 1$	$\neg 1 = 0$
Absorptionsgesetze	$x \vee (x \wedge y) = x$	$x \wedge (x \vee y) = x$

Abbildung 1: Axiomensystem der Booleschen Algebra

Definition 1 (Dualität) *Es fällt auf, dass die Gesetze in der rechten Spalte dadurch entstehen, dass man im entsprechenden Gesetz in der linken Spalte $\wedge$ mit $\vee$ vertauscht, und gleichzeitig 0 mit 1 vertauscht. Das nennt man die* **Dualität** *in der Booleschen Algebra.*

Die Dualität bedeutet, dass man auch in allen abgeleiteten Gesetzen die Vertauschungen machen kann, und man erhält wieder ein ableitbares Gesetz.

Die Gültigkeit dieser Gesetze kann man leicht mit Hilfe der Wahrheitstabellen nachrechnen. Für die Gesetze mit 2 Variablen muss man 4 Fälle überprüfen, und für die Gesetze mit 3 Variablen muss man 8 Fälle überprüfen. Wir machen das beispielhaft für $\neg(x \wedge y) = \neg x \vee \neg y$:

x	y	$x \wedge y$	$\neg(x \wedge y)$	$\neg x$	$\neg y$	$\neg x \vee \neg y$
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

Die Spalten für $\neg(x \wedge y)$ und $\neg x \vee \neg y$ stimmen offensichtlich überein.

Einige dieser Gesetze kennt man auch aus der Arithmetik: Kommutativität und Assoziativität gelten für Addition und Multiplikation. Das Distributivgesetz gilt in der Form $x \cdot (y + z) = x \cdot y + x \cdot z$ (Ausmultiplizieren). Es gilt auch $x \cdot 0 = 0$ und $x \cdot 1 = x$. Doppelnegation gilt auch: $--x = x$. Komplementarität gilt in der Form: $x + -x = 0$ und $x \cdot 1/x = 1$.

Die Gesetze sind der Ausgangspunkt für *Rechenregeln* bzw. bestimmte Datenstrukturen:

Kommutativität und Assoziativität: Die Reihenfolge und Klammerung der Argumente von $\wedge$ und $\vee$ ist egal. Man kann die Argumente als beliebig lange, beliebig geordnete Liste speichern.

Idempotenz: Doppelvorkommnisse von Argumenten von $\wedge$ und $\vee$ kann man löschen. Zusammen mit Kommutativität und Assoziativität bedeutet das, dass die Argumente als *Menge* behandelt werden können.

Neutralität und Extremalität: Die Konsequenz dieser Gesetze ist, dass man aus gemischten Termen mit Variablen, sowie 0 und 1, alle Vorkommnisse von 0 und 1 löschen kann, bis eventuell auf das letzte 0 oder 1.

Doppelnegation, Komplementarität und Absorptionsgesetze: Diese lassen sich für Vereinfachungsschritte nutzen, um redundante Teile in Termen zu eliminieren.

Doppelnegation und De Morgansche Gesetze: Diese lassen sich nutzen, um das Negationszeichen zu eliminieren oder wenigstens ganz nach innen, direkt vor die Variablen zu schieben (Negationsnormalform).

Distributivität: Nachdem man die Negationsnormalform hergestellt hat, kann man mit den Distributivitätsgesetzen entweder $\wedge$ ganz nach innen schieben, und $\vee$ ganz nach oben, oder alternativ $\vee$ ganz nach innen schieben, und $\wedge$ ganz nach oben. Im ersten Fall erhält man die *Disjunktive Normalform* (DNF), im zweiten Fall die *Konjuktive Normalform* (KNF). In beiden Fällen erhält man eine Menge von *Klauseln*, bestehend aus einer Menge von *Literalen*, die sog. *Klauselnormalform*:

$$\begin{array}{l} l_{11}, \ldots, l_{1n_1} \\ \ldots \\ l_{k1}, \ldots, l_{kn_k} \end{array}$$

Die *Literale* l_{ij} sind entweder Variablen oder negierte Variablen. Bei der Disjunktive Normalform steht das Komma für $\wedge$, und der Zeilenvorschub für $\vee$, und bei der Konjunktiven Normalform ist es umgekehrt. Die Dualität der Booleschen Algebra bedeutet, dass es für die Bearbeitung solcher Klauselmengen egal ist, ob man die DNF oder KNF hat. Für die Implementierung bedeutet es daher, dass man Klauselmengen einfach als Listen von Listen repräsentieren kann. Dies macht die Bearbeitung sehr leicht.

3.1 Normalformalgorithmus

Einen beliebigen verschachtelten Booleschen Term mit 0,1, den Variablen $x_1, \ldots, x_n$, und den Junktoren $\neg, \wedge, \vee, \Leftrightarrow, \Rightarrow, \hat{}$ kann man in folgenden Schritten in konjunktive oder disjunktive Normalform umwandeln:

Definierte Junktoren ersetzen:

- Ersetze alle Vorkommnisse von $x \Leftrightarrow y$ durch $(x \wedge y) \vee (\neg x \wedge \neg y)$. (alternative Ersetzung: $(\neg x \vee y) \wedge (x \vee \neg y)$. Welche davon am Ende eine kürzere Normalform ergibt, hängt von der Umgebung des Terms $x \Leftrightarrow y$ ab. Mit geeigneten Strategien kann man das entscheiden.)

- Ersetze alle Vorkommnisse von $x \Rightarrow y$ durch $\neg x \vee y$.
- Ersetze alle Vorkommnisse von $x \barwedge y$ durch $(x \wedge \neg y) \vee (\neg x \wedge y)$ oder alternativ durch $(x \vee y) \wedge (\neg x \vee \neg y)$.

Jetzt gibt es nur noch die Junktoren $\neg, \wedge$ und $\vee$.

Elimination von 0 und 1: Wende die Extremalitätsgesetze $x \wedge 0 = 0$, $x \vee 1 = 1$, die Neutralitätsgesetze $x \wedge 1 = x$ und $x \vee 0 = x$, sowie die Dualitätsgesetze $\neg 0 = 1$ und $\neg 1 = 0$ als Ersetzungsregeln an, um 0 und 1 zu eliminieren. Jetzt gibt es nur noch Variablen in dem Term.

Negationen nach innen ziehen:
Wende die de Morganschen Gesetze als Ersetzungsregeln an: $\neg(x \wedge y) \mapsto (\neg x \vee \neg y)$ und $\neg(x \vee y) \mapsto (\neg x \wedge \neg y)$, sowie die das Doppelnegationsgesetz $\neg\neg x \mapsto x$ an, um die Negationsnormalform herzustellen. Jetzt ist das Negationszeichen nur noch vor den Variablen.

DNF/CNF herstellen:
Wenn man jetzt das Distributivgesetz $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ als Ersetzungsregel anwendet, bekommt man die Disjunktive Normalform.

Wendet man stattdessen das Gesetz $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ als Ersetzungsregel anwendet, bekommt man die Konjunktive Normalform.

Gibt es tief verschachtelte $\Leftrightarrow$ oder $\barwedge$-Terme, dann kann deren Normalform exponentiell größer sein, als der Ausgangsterm. In den meisten Fällen ist die Normalform aber nicht sehr viel größer als der Ausgangsterm.

3.2 Vereinfachung der Klauselnormalform

Die Klauselnormalform

$$\begin{array}{l} l_{11}, \ldots, l_{1n_1} \\ \ldots \\ l_{k1}, \ldots, l_{kn_k} \end{array}$$

enthält i.A. noch viele Redundanzen, die man durch folgende Schritte reduzieren kann:

Tautologien eliminieren: Klauseln, die eine Variable x und deren Negation $\neg x$ enthalten, können gelöscht werden
(Anwendung der Komplementaritätsgesetze,
in CNF: $x \vee \neg x = 1$ zusammen mit $x \wedge 1 = x$ bzw.
in DNF $x \wedge \neg x = 0$ zusammen mit $x \vee 0 = x$).

Doppelvorkommnisse eliminieren: In Klauseln, die Literale doppelt enthalten, kann eines davon gelöscht werden (Anwendung der Idempotenzgesetze).

Subsumption: Ist eine Klausel K Teilmenge einer Klausel L, dann kann L gelöscht werden (Anwendung der Absorptionsgesetze).

Unit Resolution: Besteht eine Klausel nur aus einem Literal x oder $\neg x$, dann kann das Komplement $\neg x$ bzw. x aus allen anderen Klauseln gelöscht werden. (Anwendung des abgeleiteten Gesetzes: $x \wedge (\neg x \vee y) = x \wedge y$ und dessen duales Gesetz.)

Ersetzungsresolution 1: Gibt es eine Klausel l, y, wobei l ein Literal ist und y beliebig viele Literale enthalten kann, und eine zweite Klausel l', y, wobei l' das Komplement von l ist, dann kann die zweite Klausel komplett gelöscht werden und aus der ersten Klausel kann das l gelöscht werden. (Anwendung des abgeleiteten Gesetzes: $(x \vee y) \wedge (\neg x \vee y) = y$ und dessen duales Gesetz.)

Ersetzungsresolution 2: Gibt es eine Klausel l, y, wobei l ein Literal ist, und y beliebig viele Literale enthalten kann, und eine zweite Klausel l', y, z, wobei l' das Komplement von l ist, und das z auch beliebig viele Literale enthalten kann, dann kann das l' aus der zweiten Klausel gelöscht werden.

(Anwendung des abgeleiteten Gesetzes: $(x \vee y) \wedge (\neg x \vee y \vee z) = (x \vee y) \wedge (y \vee z)$[2] und dessen duales Gesetz.)

Wegen der Dualität der Booleschen Algebra sind diese Schritte unabhängig davon, ob die Klauseln die Disjunktive oder Konjunktive Normalform darstellen.

Aufgrund der Kommutativität und Assoziativität von $\wedge$ und $\vee$, ist die Reihenfolge der Literale in den Klauseln nicht festgelegt. Daher sind die Regeln, die sich auf bestimmte Literale in einer Klausel beziehen so zu interpretieren, dass das fragliche Literal irgendwo in der Klausel stehen kann, und nicht notwendigerweise am Anfang.

Zum Beispiel ist die Ersetzungsresolution 2 auch in folgender Situation anwendbar:
Klausel K1: a, b, c
Klausel K2: $b, \neg c, d, a$

Diese Klauseln können nämlich umsortiert werden zu
Klausel K1: c, a, b
Klausel K2: $\neg c, a, b, d$
worauf das obige Muster anwendbar ist, so dass $\neg c$ aus Klausel K2 eliminiert werden kann.

Die Elimination von Tautologien und Doppelvorkommnissen sind Schritte, die man einmal für alle Klauseln macht. Die anderen Schritte sind aber so, dass die Anwendung einer Vereinfachung weitere Vereinfachung erst ermöglicht, so dass die Schritte immer wieder gemacht werden müssen, bis keine Veränderung mehr möglich ist.

Die Löschung von Literalen in einer Klausel kann dazu führen, dass die Klausel leer wird. Was eine leere Klausel bedeutet, hängt aber jetzt doch davon ab, ob es die Konjunktive oder Disjunktive Normalform war. In Konjunktiver Normalform bedeutet die leere Klausel die 0 (oder falsch), und in Disjunktiver Normalform bedeutet die leere Klausel die 1 (oder wahr).

[2] Durch Fallunterscheidung kann man die Äquivalenz von
1: $(x \vee y) \wedge (\neg x \vee y \vee z)$ und 2: $(x \vee y) \wedge (y \vee z)$ am leichtesten nachweisen.
Fall $x = 0$. Dann ist 1: $(0 \vee y) \wedge (1 \vee y \vee z) = y$ und 2: $(0 \vee y) \wedge (y \vee z) = y \wedge (y \vee z) = y$.
Fall $x = 1$. Dann ist 1: $(1 \vee y) \wedge (0 \vee y \vee z) = 1 \wedge (y \vee z) = (y \vee z)$ und 2: $(1 \vee y) \wedge (y \vee z) = y \vee z$.

4 Kategorisierung von Booleschen Termen

Boolesche Terme wie z.B. $x \wedge \neg(x \vee y)$ sind Darstellungen von verschachtelten Booleschen Funktionen wie $f(x, y) = x \wedge \neg(x \vee y)$. Solche Funktionen kann man aufrufen mit den Werten 0 oder 1 für x und y, und erhält dann entweder 0 oder 1 als Funktionswert. Man sagt dann jedoch meistens nicht „ich rufe die Funktion $f(x, y)$ mit den Argumenten, z.B. $x = 0, y = 1$, auf", sondern man sagt „der Boolesche Term $x \wedge \neg(x \vee y)$ wird unter der Belegung, oder mit der Bindung $[x/0, y/1]$[3] ausgewertet". Gemeint ist aber genau das gleiche!

Die Komplementaritätsgesetze $x \wedge \neg x = 0$ und $x \vee \neg x = 1$ bedeuten, dass es offensichtlich Boolesche Terme gibt, die egal, ob das x mit 0 oder mit 1 belegt wird, immer 0 bzw. 1 ergeben. Das müssen dabei nicht so einfache Terme sein, wie $x \wedge \neg x$, sondern es können sehr tief verschachtelte Terme sein, denen man nicht so einfach ansieht, dass es nicht darauf ankommt, ob das x an 0 oder 1 gebunden wird.

Daher hat man Boolesche Terme $f(x_1, \ldots, x_n)$ folgendermaßen kategorisiert:

Inkonsistente Terme (unerfüllbare Terme, Widersprüche): diese ergeben immer 0, egal mit welchen Werten die Variablen x_i belegt werden.

Allgemeingültige Terme (Tautologien): diese ergeben immer 1, egal mit welchen Werten die Variablen x_i belegt werden.

Konsistente Terme (erfüllbare Terme): für diese gibt es Belegungen für die Variablen x_i, so dass der Term zu 1 wird. Eine solche Belegung bezeichnet man auch als *Modell* für den Term. Das schließt die allgemeingültigen Terme mit ein.

Falsifizierbare Terme: für diese gibt es Belegungen mit 0 oder 1 für die Variablen x_i, so dass der Term zu 0 wird. Das schließt die inkonsistenten Terme mit ein.

Für jeden Booleschen Term t gilt dann:

t ist allgemeingültig gdw. $\neg t$ ist widersprüchlich.

Eine der wichtigsten Tätigkeiten von Mathematikern ist, Theoreme zu beweisen. Das bedeutet konkret, für Formeln der Art „Annahmen $\Rightarrow$ Behauptung" nachzuweisen, dass sie allgemeingültig sind. Sind die Aussagen einfach genug, dann kann man die Annahmen und die Behauptung in Aussagenlogik bzw. Boolescher Algebra formulieren. Daraus ergibt sich die Aufgabe, „Annahmen $\Rightarrow$ Behauptung" als allgemeingültig nachzuweisen, bzw. „$\neg$(Annahmen $\Rightarrow$ Behauptung)" als widersprüchlich. „$\neg$(Annahmen $\Rightarrow$ Behauptung)" ist aber äquivalent zu „Annahmen $\wedge \neg$ Behauptung". Dies als widersprüchlich nachzuweisen ist eine der Standardvorgehensweisen der Mathematiker, bekannt als *Beweis durch Widerspruch*. Man zeigt, dass die Annahmen zusammen mit der negierten Behauptung einen Widerspruch ergeben.

[3] Manchmal wird das auch andersherum geschrieben: $[0/x, 1/y]$.

4.1 SAT-Solving

Inzwischen wurden Algorithmen entwickelt, mit denen man Boolesche Terme, die Hunderte oder gar Tausende von Variablen haben, daraufhin testen kann, in welcher Kategorie sie liegen. Ein gängiges Verfahren ist, den Booleschen Term zunächst in Konjunktive Normalform zu transformieren (siehe Abschnitt 3.1), diese zu vereinfachen, so weit es geht (siehe Abschnitt 3.2), und dann einen SAT-Solver Algorithmus anzuwenden. SAT-Solving ist ein eigenes Forschungsgebiet, in dem es sogar internationale Wettbewerbe gibt, wo die besten SAT-Solver gegeneinander antreten. SAT-Solver untersuchen eine Klauselmenge danach, ob sie inkonsistent oder konsistent ist. Für einen Test auf Allgemeingültigkeit müsste man den Ausgangsterm negieren, und dann auf Inkonsistenz testen.

Derzeit gibt es im Prinzip zwei unterschiedliche Algorithmenansätze für SAT-Solving

Systematische Fallunterscheidung: Dieser auf Davis und Putnam zurückgehender Ansatz startet, indem man eine der Variablen x auswählt, und zunächst die Belegung $x = 1$ (oder alternativ auch $x = 0$) untersucht. Wenn $x = 1$, kann man alle Klauseln mit x komplett löschen, und in allen anderen Klauseln $\neg x$ löschen. Damit erhält man eine kleinere Klauselmenge, für die man auf die gleiche Weise mit einer weiteren Variable y verfährt, und das geht rekursiv dann weiter. Ist die Klauselmenge leer geworden, dann ist sie erfüllbar und der Algorithmus ist fertig. Ist dagegen eine leere Klausel erzeugt worden, dann war die letzte Wahl, z.B. $y = 1$ falsch, und man fährt mit dem zweiten Fall $y = 0$ fort (Backtracking). Auf diese Weise probiert man systematisch alle möglichen Belegungen für die Variablen durch, bis man entweder eine leere Klauselmenge gefunden hat, oder alle Belegungen eine leere Klausel generiert haben. Im zweiten Fall ist die Klauselmenge unerfüllbar.

Die verschiedenen Implementierungen unterscheiden sich einmal in den Heuristiken, mit denen man die nächste zu testende Variable auswählt, und zum anderen, wie man die vereinfachten Klauselmengen noch weiter vereinfacht, bevor man die nächste Variable ausprobiert (Constraint Propagation).

Random Walk: Bei diesem Ansatz startet man mit einer zufällig gewählten Belegung *für alle Variablen*. Wenn jede Klausel ein Literal enthält, welches unter der Belegung zu 1 wird, dann ist die Klauselmenge erfüllbar, und man hat ein Modell gefunden. Wenn es Klauseln gibt, die unter der Belegung zu 0 werden, dann wählt man möglichst geschickt für einzelne Variablen eine alternative Belegung, und testet dann die Klauseln wieder neu.

Bei unerfüllbaren Klauselmengen kann dieses Verfahren leider nicht terminieren. Bei erfüllbaren Klauselmengen kann eine geschickte Implementierung davon jedoch extrem schnell sein.

5 Erzeugung Boolescher Funktionen aus Wertetabellen

Die digitalen Schaltkreise in unseren heutigen Computer sind zum großen Teil Hardwarerealisierungen von Booleschen Funktionen. Bei der Konstruktion eines digitalen Schaltkreises startet man jedoch nicht mit einer Booleschen Funktion, sondern zunächst mit einer informellen Beschreibung, die man dann in eine Wertetabelle übersetzt, aus der man automatisch die Boolesche Funktion generieren kann (und aus der man dann den Schaltkreis ableitet).

Wie das konkret geschieht, kann man an ein paar kleinen Beispielen sehr gut illustrieren.

Beispiel: Halbaddierer
Ein Halbaddierer addiert zwei Bits $x + y$ und erzeugt ein Ergebnisbit E und ein Übertragsbit U. Die Wertetabelle dazu ist:

x	y	E	U
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Diese Wertetabelle spezifiziert die zwei Booleschen Funktionen $E(x, y)$ und $U(x, y)$. Um die Booleschen Terme dafür zu finden, überprüft man in den entsprechenden Spalten, wann die Funktion 1 wird.

E wird genau dann 1 wenn $x = 0$ und $y = 1$ oder wenn $x = 1$ und $y = 0$ ist. Daraus lässt sich der Boolesche Term ableiten:

$$E(x, y) = (\neg x \wedge y) \vee (x \wedge \neg y)$$

(das ist genau die xor-Funktion).

U wird genau dann 1 wenn $x = 1$ und $y = 1$ ist. Daraus lässt sich der Boolesche Term ableiten:

$$U(x, y) = (x \wedge y).$$

Die beiden Funktionen lassen sich mit sog. *Gattern*, die die Funktionen $\wedge$, $\vee$ und $\neg$ technisch realisieren, zu einem Schaltkreis zusammenschalten. Dabei sind die Booleschen Formeln genau die Vorschrift, wie die Gatter zu verschalten sind.

Beispiel: Volladdierer
Ein Halbaddierer addiert zwei Bits und erzeugt ein Ergebnisbit E und ein Übertragsbit U. Um diesen Übertrag weiter zu verarbeiten, braucht man einen Volladdierer, der zwei Bits x und y und ein Übertragsbit z addiert. Die Wertetabelle dafür ist

x	y	z	E	Ü
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Um den Booleschen Term für E zu finden, überprüfen wir alle Einträge in der E-Spalte, wo 1 steht, und erzeugen aus jedem zugehörigen x, y, z-Tripel einen Booleschen Term. Wenn an der x-Position eine 1 steht wird daraus x, wenn dort 0 steht, wird daraus $\neg x$. E wird in insgesamt 4 Fällen zu 1. Daraus ergibt sich:

$$E(x, y, z) = (\neg x \wedge \neg y \wedge z) \vee (\neg x \wedge y \wedge \neg z) \vee (x \wedge \neg y \wedge \neg z) \vee (x \wedge y \wedge z)$$

U wird auch in insgesamt 4 Fällen zu 1. Daraus ergibt sich:

$$U(x, y, z) = (\neg x \wedge y \wedge z) \vee (x \wedge \neg y \wedge z) \vee (x \wedge y \wedge \neg z) \vee (x \wedge y \wedge z)$$

Auch aus diesen Funktionen lässt sich ablesen, wie die Gatter zu verschalten sind, um einen Volladdierer zu realisieren.

Die Vorgehensweise sollte jetzt klar sein. Die Wertetabelle ergibt sich aus dem, was der Schaltkreis / die Boolesche Funktion leisten soll. Man bekommt dabei eine bestimmte Anzahl von Eingabewerten $(x, y, z, \ldots)$ und eine bestimmte Menge von gewünschten Ausgabewerten. Den Booleschen Term für jeden der Ausgabewerte erhält man, indem man in der entsprechenden Spalte die Einträge mit 1 betrachtet, und für jeden solchen 1er Eintrag aus den Bedingungen, wie die 1 entsteht den Booleschen Term generiert. Eine 1 erzeugt den zugehörigen Variablennamen direkt, eine 0 erzeugt die Negation davon. Diese Literale werden mit $\wedge$ verbunden. Die aus den 1er-Zeilen erzeugten Terme werden mit $\vee$ verbunden. Das ergibt übrigens sofort eine Disjunktive Normalform.

Die so erzeugten Terme werden oft sehr redundant, und müssen daher mit den in den früheren Abschnitten beschriebenen Verfahren vereinfacht werden.

Falls die Spalte für den Ausgabewert mehr 0en als 1en enthält, kann man statt den Zeilen mit 1 auch die Zeilen mit 0 zu einem Booleschen Term machen. Dieser beschreibt dann die negierte Boolesche Funktion. Negiert man sie nochmal, erhält man die richtige Boolesche Funktion.

6 Mengenalgebra

Die Boolesche Algebra mit den Werten 0 und 1 ist nicht die einzige mathematische Struktur, die die Axiome der Booleschen Algebra 3 erfüllen. Betrachten wir eine nichtleere Grundmenge G, mit all ihren Teilmengen, und machen folgende Ersetzungen:

$$\begin{array}{rcl} \wedge & \rightarrow & \cap \\ \vee & \rightarrow & \cup \\ \neg & \rightarrow & \text{Komplement } ' \\ 0 & \rightarrow & \text{die leere Menge } \emptyset \\ 1 & \rightarrow & G \end{array}$$

dann gelten, wenn X, Y, Z Teilmengen von G sind, die gleichen Axiome der Booleschen Algebra:

Name	Gesetz	Duales Gesetz
Kommutativgesetze	$X \cap Y = Y \cap X$	$X \cup Y = Y \cup X$
Assoziativgesetze	$X \cap (Y \cap Z) = (X \cap Y) \cap Z$	$X \cup (Y \cup Z) = (X \cup Y) \cup Z$
Idempotenzgesetze	$X \cap X = X$	$X \cup X = X$
Distributivgesetze	$X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$	$X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$
Neutralitätsgesetze	$X \cap G = X$	$X \cup \emptyset = X$
Extremalgesetze	$X \cap \emptyset = \emptyset$	$X \cup G = G$
Doppelnegation (Involution)	$X'' = X$	
De Morgansche Gesetze	$(X \cap Y)' = X' \cup Y'$	$(X \cup Y)' = X' \cap Y'$
Komplementärgesetze	$X \cap X' = \emptyset$	$X \cup X' = G$
Dualitätsgesetze	$\emptyset' = G$	$G' = \emptyset$
Absorptionsgesetze	$X \cup (X \cap Y) = X$	$X \cap (X \cup Y) = X$

Wenn man also nicht sicher ist, ob eine Berechnung für die Boolesche Algebra mit 0 und 1 stimmt oder nicht, dann kann man sie in die Mengenalgebra übertragen, und dort überprüfen. Die Intuition ist für die Mengenalgebra oft ausgeprägter als für die Boolesche Algebra mit 0 und 1.

In der Tat ist nicht die Boolesche Algebra mit 0 und 1, sondern die Mengenalgebra eine Art *prototypische* Boolesche Algebra. Es gilt nämlich das *Stonesche Repräsentationstheorem*: Jede Boolesche Algebra ist isomorph zu einer Mengenalgebra.

7 Aussagenlogik

Viele Logikbücher starten mit der Einführung der *Aussagenlogik*. Aus rein mathematischer Sicht ist die Aussagenlogik genau die Boolesche Algebra mit 1 und 0 bzw. wahr und falsch. Allerdings ergeben sich aus den Anwendungen der Booleschen Algebra, z.B. Schaltungsentwurf, sowie der Aussagenlogik, z.B. Wissensrepräsentation, unterschiedliche Gesichtspunkte und Problematiken. Die Booleschen Terme im Schaltungsentwurf sind i.A. sowieso erfüllbar, und es geht eher um ihre Vereinfachung. Bei Anwendungen der Aussagenlogik geht es eher um den Beweis von Implikationen:
Annahmen $\Rightarrow$ Behauptung. Mittels Beweis durch Widerspruch lässt sich das Problem auf das SAT-Problem reduzieren: Ist Annahmen $\wedge \neg$ Behauptung inkonsistent oder nicht?

Stichwortverzeichnis