# Chapter 3

# ⋆ Boolean Algebras and Circuits

*There are 10 types of people in this world: those who understand binary numbers and those who don't.*

- Anonymous.

At the end of the last chapter we noted a close analogy between Equivalence Laws for Propositional Logic on the one hand, and Set Identities on the other. In this chapter we explore this connection by looking at ***Boolean algebras***, the mathematical structures underlying both propositional logic and sets.

This analogy extends to the world of digital computers and other electronic devices, which are built from circuits which have binary inputs and outputs; that is, they manipulate values from the set $\mathbb{B} = \{0, 1\}$. At the implementation level these binary inputs and outputs are delivered by voltages on wires, with a low voltage being interpreted as 0 and a high voltage being interpreted as 1. The simplest components of digital circuits, ***logic gates***, are based on the connectives of propositional logic, with 0 (low voltage) and 1 (high voltage) being interpreted as **F** (false) and **T** (true), respectively. Composing logic gates together to create ever more complicated electronic components can thus be done in a way which is amenable to analysis via propositional logic. In this chapter we shall examine the fundamental role of Boolean algebra in underlying the building blocks of digital computers.

## 3.1 Boolean Algebras

A ***Boolean algebra*** is a set $B$ which contains (at least) two distinct special elements 0 and 1, referred to as ***zero*** and ***unit***, respectively, along with two binary operators $+$ and $\cdot$, referred to as ***sum*** and ***product***, as well as a unary operator $'$, referred to as ***complementation***. That is, for every pair $(x, y)$ of elements of $B$ there are three further (but not necessarily different) elements of $B$ denoted $x+y$, $x \cdot y$, and $x'$. These operators must all satisfy the ten ***Laws of Boolean Algebra*** given in Figure 3.1.

F. Moller, G. Struth, *Modelling Computing Systems*,
Undergraduate Topics in Computer Science,
DOI 10.1007/978-1-84800-322-4_4, © Springer-Verlag London 2013

| | | | | |
|---|---|---|---|---|
| ***Commutativity:*** | $x + y$ | $=$ | $y + x$ | *(Comm1)* |
| | $x \cdot y$ | $=$ | $y \cdot x$ | *(Comm2)* |
| ***Associativity:*** | $(x + y) + z$ | $=$ | $x + (y + z)$ | *(Assoc1)* |
| | $(x \cdot y) \cdot z$ | $=$ | $x \cdot (y \cdot z)$ | *(Assoc2)* |
| ***Distributivity:*** | $x + (y \cdot z)$ | $=$ | $(x + y) \cdot (x + z)$ | *(Distr1)* |
| | $x \cdot (y + z)$ | $=$ | $(x \cdot y) + (x \cdot z)$ | *(Distr2)* |
| ***Identity:*** | $x + 0$ | $=$ | $x$ | *(Ident1)* |
| | $x \cdot 1$ | $=$ | $x$ | *(Ident2)* |
| ***Complement:*** | $x + x'$ | $=$ | $1$ | *(Compl1)* |
| | $x \cdot x'$ | $=$ | $0$ | *(Compl2)* |

Figure 3.1: The Laws of Boolean Algebra.

Boolean algebras provide an abstract representation of familiar ideas in various areas of study. Indeed we have already met concrete examples of Boolean algebras in the form of sets and propositions.

**Example 3.1** The Boolean Algebra of Sets

The power set $\mathcal{P}(U)$ of a set $U$ gives rise to a Boolean algebra, with the roles of 0, 1, $+$, $\cdot$ and $'$ taken by $\emptyset$, $U$, $\cup$, $\cap$ and $^-$, respectively.

In this case, the laws give rise to the following set identities, which we confirmed in Section 2.7:

| | | | | |
|---|---|---|---|---|
| ***Commutativity:*** | $A \cup B$ | $=$ | $B \cup A$ | *(Comm1)* |
| | $A \cap B$ | $=$ | $B \cap A$ | *(Comm2)* |
| ***Associativity:*** | $(A \cup B) \cup C$ | $=$ | $A \cup (B \cup C)$ | *(Assoc1)* |
| | $(A \cap B) \cap C$ | $=$ | $A \cap (B \cap C)$ | *(Assoc2)* |
| ***Distributivity:*** | $A \cup (B \cap C)$ | $=$ | $(A \cup B) \cap (A \cup C)$ | *(Distr1)* |
| | $A \cap (B \cup C)$ | $=$ | $(A \cap B) \cup (A \cap C)$ | *(Distr2)* |
| ***Identity:*** | $A \cup \emptyset$ | $=$ | $A$ | *(Ident1)* |
| | $A \cap U$ | $=$ | $A$ | *(Ident2)* |
| ***Complement:*** | $A \cup \overline{A}$ | $=$ | $U$ | *(Compl1)* |
| | $A \cap \overline{A}$ | $=$ | $\emptyset$ | *(Compl2)* |

**Example 3.2** The Boolean Algebra of Propositions

The set of propositions gives rise to a Boolean algebra, with the roles of 0, 1, $+$, $\cdot$ and $'$ taken by false, true, $\vee$, $\wedge$ and $\neg$, respectively. (Equality $p = q$ is interpreted by equivalence $p \Leftrightarrow q$.)

In this case, the laws give rise to the following equivalences, which we confirmed in Section 1.7:

| | | |
|---|---|---|
| ***Commutativity:*** | $p \vee q \Leftrightarrow q \vee p$ | *(Comm1)* |
| | $p \wedge q \Leftrightarrow q \wedge p$ | *(Comm2)* |
| ***Associativity:*** | $(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$ | *(Assoc1)* |
| | $(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$ | *(Assoc2)* |
| ***Distributivity:*** | $p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$ | *(Distr1)* |
| | $p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$ | *(Distr2)* |
| ***Identity:*** | $p \vee \mathsf{false} \Leftrightarrow p$ | *(Ident1)* |
| | $p \wedge \mathsf{true} \Leftrightarrow p$ | *(Ident2)* |
| ***Complement:*** | $p \vee \neg p \Leftrightarrow \mathsf{true}$ | *(Compl1)* |
| | $p \wedge \neg p \Leftrightarrow \mathsf{false}$ | *(Compl2)* |

**Example 3.3** The two-valued Boolean Algebra

The two-element set $\mathbb{B} = \{0, 1\}$ itself gives rise to an important Boolean algebra, with the operations defined as follows:

| $x$ | $y$ | $x+y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| $x$ | $y$ | $x \cdot y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| $x$ | $x'$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

As we shall see, this particular algebra is of fundamental importance in the design of digital circuits.

**Exercise 3.3** (Solution on page 421)

Verify that the laws of Boolean algebra hold for the two-valued Boolean algebra $\mathbb{B}$.

From now on we shall typically omit $\cdot$ and write $xy$ rather than $x \cdot y$, and freely omit parentheses by allowing $\cdot$ to bind tighter than $+$ and $'$ to bind tighter than $\cdot$; thus for example, we shall write $x + (y \cdot (z'))$ simply as $x + yz'$.

## 3.2 Deriving Identities in Boolean Algebras

From the Laws of Boolean Algebra, we can derive very many identities which must be true in any Boolean algebra (in particular, as set identities and logical equivalences). In this section we derive some important identities, and leave it as an exercise to consider what these identities say as set identities and logical equivalences, many of which were derived already in previous chapters. We shall state our new identities as *Theorems* (true statements), and justify their truth using *proofs* (step-by-step derivations of their truth); the appearance of the box symbol "□" indicates the end of a proof.

**Theorem 3.3** Further Distributive Laws

$$\begin{aligned} (x+y)z &= xz+yz && \textit{(Distr3)} \\ xy+z &= (x+z)(y+z) && \textit{(Distr4)} \end{aligned}$$

**Proof:** We prove only the first identity, and leave the second as an exercise.

$$\begin{aligned} (x+y)z &= z(x+y) && \textit{(Comm2)} \\ &= zx+zy && \textit{(Distr2)} \\ &= xz+yz && \textit{(Comm2, twice)} \end{aligned}$$

□

**Theorem 3.4** Idempotence Laws

$$\begin{aligned} x+x &= x && \textit{(Idemp1)} \\ xx &= x && \textit{(Idemp2)} \end{aligned}$$

**Proof:** We prove only the first identity, and leave the second as an exercise.

$$\begin{aligned} x+x &= (x+x)1 && \textit{(Ident2)} \\ &= (x+x)(x+x') && \textit{(Compl1)} \\ &= x+xx' && \textit{(Distr1)} \\ &= x+0 && \textit{(Compl2)} \\ &= x && \textit{(Ident1)} \end{aligned}$$

□

**Theorem 3.5** Domination Laws

$$\begin{aligned} x+1 &= 1 && \textit{(Dom1)} \\ x\,0 &= 0 && \textit{(Dom2)} \end{aligned}$$

**Proof:** We prove only the first identity, and leave the second as an exercise.

$$\begin{aligned} x+1 &= x+(x+x') && (Compl1)\\ &= (x+x)+x' && (Assoc1)\\ &= x+x' && (Idemp1)\\ &= 1 && (Compl1) \end{aligned}$$ □

**Theorem 3.6** Absorption Laws

$$\begin{aligned} x+xy &= x && (Absorp1)\\ x(x+y) &= x && (Absorp2) \end{aligned}$$

**Proof:** We prove the first identity here, and present the proof of the second in Example 3.11.

$$\begin{aligned} x+xy &= x1+xy && (Ident2)\\ &= x(1+y) && (Distr2)\\ &= x(y+1) && (Comm1)\\ &= x1 && (Dom1)\\ &= x && (Ident2) \end{aligned}$$ □

Next, we prove a law that we shall find useful in further calculations.

**Theorem 3.7**

*If* $x+y=x+z$ *and* $xy=xz$ *then* $y=z$.

**Proof:**

$$\begin{aligned} y &= y(x+y) && (Comm1,\ Absorp2)\\ &= y(x+z) && (Assunption\ 1)\\ &= yx+yz && (Distr2)\\ &= zx+zy && (Comm2,\ Assumption\ 2)\\ &= z(x+y) && (Distr2)\\ &= z(x+z) && (Assumption\ 1)\\ &= z && (Comm1,\ Absorp2) \end{aligned}$$ □

Next, we consider a few results about complementation. The first of these is the observation that the two Complementation Laws $x + x' = 1$ and $xx' = 0$ uniquely determine the complement: there is no value $y$ different from $x'$ which satisfies these two equations.

**Theorem 3.8** Uniqueness of Complement

*If $x + y = 1$ and $xy = 0$ then $y = x'$. That is to say, $x'$ is the only element which satisfies $x + x' = 1$ and $xx' = 0$.*

**Proof:** Suppose that $x + y = 1$ and $xy = 0$. Then

$$\begin{aligned} x + y &= 1 && \textit{(Assumption 1)} \\ &= x + x' && \textit{(Compl1)} \end{aligned}$$

and

$$\begin{aligned} xy &= 0 && \textit{(Assumption 2)} \\ &= xx' && \textit{(Compl2)} \end{aligned}$$

Thus, by Theorem 3.7, $y = x'$. □

**Theorem 3.9** Involution Law

$(x')' = x$.

**Proof:**

$$\begin{aligned} x' + (x')' &= 1 && \textit{(Compl1)} \\ &= x' + x && \textit{(Comm1, Compl1)} \end{aligned}$$

and

$$\begin{aligned} x'(x')' &= 0 && \textit{(Compl2)} \\ &= x'x && \textit{(Comm2, Compl2)} \end{aligned}$$

Thus, by Theorem 3.7, $(x')' = x$. □

**Exercise 3.9** (Solution on page 422)

Prove that $0' = 1$ and $1' = 0$.

**Theorem 3.10** De Morgan Laws

$$\begin{aligned} (x + y)' &= x'y' && \textit{(DeMorgan1)} \\ (xy)' &= x' + y' && \textit{(DeMorgan2)} \end{aligned}$$

**Proof:** We prove only the first identity, and leave the second as an exercise.

We first note that it suffices to show that

$$(x+y)(x'y') \;=\; 0 \quad \text{and} \quad (x+y)+(x'y') \;=\; 1$$

as then, by the Uniqueness of Complement Theorem 3.8, we would get that $(x+y)' \;=\; x'y'$.

$$\begin{aligned}
(x+y)(x'y') &= x(x'y')+y(x'y') && \textit{(Distr3)}\\
&= 0y'+0x' && \textit{(Assoc2, Comm2, Compl2)}\\
&= 0+0 && \textit{(Dom2)}\\
&= 0 && \textit{(Idemp1)}
\end{aligned}$$

$$\begin{aligned}
(x+y)+(x'y') &= ((x+y)+x')((x+y)+y') && \textit{(Distr1)}\\
&= (y+1)(x+1) && \textit{(Assoc1, Comm1, Compl1)}\\
&= 1\cdot 1 && \textit{(Dom1)}\\
&= 1 && \textit{(Idemp2)}
\end{aligned}$$

□

**Exercise 3.10** (Solution on page 422)

Prove the following theorems.

1. $(xy+x'y')' = xy'+x'y$.
2. If $x+y=x+z$ and $x'+y=x'+z$ then $y=z$.
3. If $x+y=0$ then $x=y=0$.
4. $x=0$ if, and only if, $y=xy'+x'y$ for all $y$.

## 3.3 The Duality Principle

Given any formula in a Boolean algebra, its ***dual*** is formed by interchanging 0 and 1, and $+$ and $\cdot$, throughout. More generally, the dual of a statement involving Boolean algebra is that statement with every formula replaced with its dual. Thus for example, the dual of $x+y'z=1$ is $x(y'+z)=0$.

The following is a fundamental principle of Boolean algebras.

**Theorem 3.11** The Principle of Duality

*The dual of every theorem of Boolean algebra is also a theorem.*

**Proof:** To see that this is a valid principle, we merely need realise that a proof of a theorem becomes a proof of the dual of the theorem simply by

replacing each formula used in the proof by its dual. This is so since the Laws of Boolean Algebra consist of five statements and their duals. □

**Example 3.11**

Consider the following derivation of the second Absorption Law $x(x+y) = x$:

$$\begin{aligned} x(x+y) &= (x+0)(x+y) && \textit{(Ident1)} \\ &= x+0y && \textit{(Distr1)} \\ &= x+y0 && \textit{(Comm2)} \\ &= x+0 && \textit{(Dom2)} \\ &= x && \textit{(Ident1)} \end{aligned}$$

If we compare this derivation with that given in the proof of the first Absorption Law $x + (xy) = x$ in Theorem 3.6, the duality is immediately apparent: the two derivations are identical, but for the fact that each expression is replaced by its dual, and the identity justifying each step in the above derivation is the dual of the identity justifying the same step in the first derivation.

---

This principle allows us to infer the validity of the dual of any theorem that we prove, since a proof of the dual theorem can be constructed automatically from the proof of the theorem, simply by replacing every formula and identity with its dual, as in the above example. Throughout the previous section we provided theorems presenting pairs of identities; and in each case we only proved the first of each identity, leaving the proof of the second as an exercise. In fact, the second identity in each case is the dual of the first; so by using the Duality Principle, proofs of these are unnecessary. The Duality Principle guarantees that they are valid.

**Exercise 3.11** (Solution on page 423)

Write out the dual of each of the following theorems from Exercise 3.10.

1. $(xy + x'y')' = xy' + x'y$.
2. If $x+y = x+z$ and $x'+y = x'+z$ then $y = z$.
3. If $x+y = 0$ then $x = y = 0$.
4. $x = 0$ if, and only if, $y = xy' + x'y$ for all $y$.

---

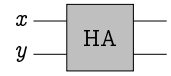
## 3.4 Logic Gates and Digital Circuits

Computers manipulate all forms of information: numbers, names, sounds, pictures, videos; but an electronic computer can only reliably represent data in essentially one way: either a wire has a high voltage, or it has a low voltage. By interpreting a high voltage as the number 1 and a low voltage as the number 0, every piece of data represented and manipulated by an electronic computer is reduced within the electronics of the machine to combinations of the binary digits (the ***bits***) 0 and 1.

At its lowest level, a computer manipulates this binary data using digital circuits which transform voltages on wires feeding into the circuit into voltages on wires leading out from it. How the electronics works (using transistors) to cause the output voltages to reflect the correct values according to the input voltages is not a question that will concern us here; such concerns are left to physicists and electronics engineers.

**Example 3.12**

Consider a circuit HA with two input wires, labelled $x$ and $y$, and two output wires, labelled $s$ and $c$. Such a circuit might be represented as follows:




Note that when we draw a circuit, we will assume that its input lines enter from the left and its output lines exit from the right.

Such a picture may represent the circuit simply as a black box as above, with no indication as to how the output values relate to the input values. However, we can describe the behaviour of the circuit by indicating what output values are produced from each of the possible input values. To do this, we can list all of the possibilities in the form of a truth table. For example, the circuit HA which we have in mind above behaves as follows:

| $x$ | $y$ | $s$ | $c$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Thus, for example, if both input wires $x$ and $y$ hold high voltages, thus both representing the value 1, then the output wire $s$ will be given a low voltage, representing the value 0, and the output wire $c$ will be given a high voltage, representing the value 1.

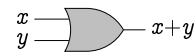
Computer circuits can be extremely complicated – far more complicated than the above example. However, all circuits, including the one above, can

be built up from three very basic building blocks (which can all be easily implemented using transistors): ***OR gates***, ***AND gates*** and ***NOT gates***.

An ***OR gate*** is a simple component circuit which takes two inputs $x$ and $y$ and produces the single output $x+y$ defined by

$$x+y \;=\; \begin{cases} 1 & \textit{if } x=1 \textit{ or } y=1; \\ 0 & \textit{otherwise.} \end{cases}$$

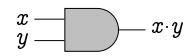
Graphically it is drawn as follows:




An ***AND gate*** is a simple component circuit which takes two inputs $x$ and $y$ and produces the single output $x\cdot y$ defined by

$$x\cdot y \;=\; \begin{cases} 1 & \textit{if } x=1 \textit{ and } y=1; \\ 0 & \textit{otherwise.} \end{cases}$$

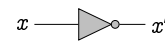
Graphically it is drawn as follows:




A ***NOT gate*** is a simple component circuit which takes one input $x$ and produces the single output $x'$ defined by

$$x' \;=\; \begin{cases} 1 & \textit{if } x=0; \\ 0 & \textit{if } x=1. \end{cases}$$

Graphically it is drawn as follows:




Truth tables defining these three gates are as follows:

| $x$ | $y$ | $x+y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| $x$ | $y$ | $x\cdot y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| $x$ | $x'$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

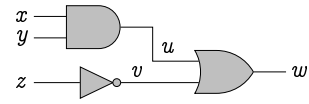
We can observe from the above definitions that the three basic gates compute exactly the functions of the two-valued Boolean algebra $\mathbb{B}$ defined in Example 3.3. (Note that, as before, we shall typically write $xy$ instead of $x\cdot y$.) This section makes clear, then, the fundamental importance of this

particular Boolean algebra. It is absolutely essential in the design of digital computers.

We can build large complicated circuits from these three basic gates by stringing them together – always in a left-to-right fashion. (Allowing feedback wires provides its own uses – and complications – which we shall not explore.)

**Example 3.13**

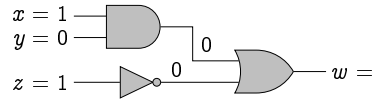
Consider the following circuit:




There are three inputs $x$, $y$ and $z$ to this circuit. The inputs $x$ and $y$ feed into an AND gate which outputs an intermediate value $u = xy$. Meanwhile, the input $z$ feeds into a NOT gate which outputs a second intermediate value $v = z'$. The two intermediate values $u$ and $v$ output by the first two gates then feed as inputs into an OR gate which outputs the final value $w = u + v$. The effect of the whole circuit, therefore, is to output the value $w = xy + z'$. The value that is output is thus given according to the following table:

| $x$ | $y$ | $z$ | $u$ | $v$ | $w$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 |

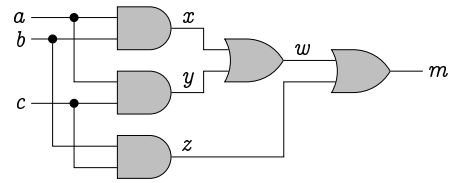
For example, if the inputs have values $x$=1, $y$=0 and $z$=1, then the output of the AND gate will be 0, as will the output of the NOT gate; and since both of the inputs to the OR gate will be 0, the value of the output $w$ will also be 0:




The relationship between the table defining the function $w = xy + z'$ and the truth table for the proposition $(P \wedge Q) \vee \neg R$ is, hopefully, obvious.

**Example 3.14**

Consider the following circuit with three input lines $a$, $b$ and $c$, and one output line $m$:




Note that we have used a dot to split a line, directing the same value (voltage) to two different inputs; and we have allowed lines to cross without interference (as if they were insulated from each other).

We can analyse the behaviour of this circuit as follows:

- the inputs $a$ and $b$ feed into the first AND gate to produce an intermediate value $x = ab$;
- the inputs $a$ and $c$ feed into the second AND gate to produce an intermediate value $y = ac$;
- the inputs $b$ and $c$ feed into the third AND gate to produce an intermediate value $x = bc$;
- the values $x$ and $y$ then feed into the first OR gate to produce a further intermediate value $w = x + y$;
- finally, the values $w$ and $z$ feed into the second OR gate to produce the final value $m = w + z$.

We can tabulate the value that is output by this circuit on any set of inputs as follows:

| $a$ | $b$ | $c$ | $x$ | $y$ | $z$ | $w$ | $m$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Algebraically, the effect of the circuit is to output the value

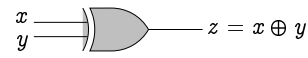
$$m = w + z = x + y + z = ab + ac + bc.$$

In other words, this circuit computes the majority function: the output $m$ will be 1 exactly when at least two of the input values are 1.

**Exercise 3.14** (Solution on page 423)

The ***exclusive-OR gate***, or ***XOR gate***, has the following definition (and gate symbol):

| $x$ | $y$ | $x \oplus y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

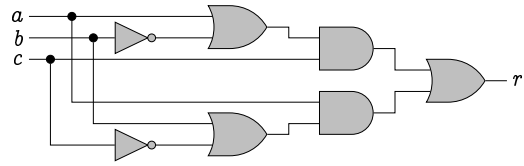



That is, the output $z = x \oplus y$ has the value 1 when exactly one of the inputs $x$ or $y$ has the value 1 (and the other has the value 0).

Build a circuit which realises this gate.

**Exercise 3.15** (Solution on page 423)

Describe the behaviour of the following circuit by providing a Boolean expression and a truth table defining the output value $r$.




**Exercise 3.16** (Solution on page 424)

Consider a car safety system in which a warning bell rings whenever the motor is running while a door is open or a seat belt is unbuckled. This is to be implemented as a Boolean circuit which takes three inputs $M$, $D$ and $B$, respectively representing the states of the motor, doors and seat belts:

- $M$ will be 1 if the motor is running and 0 otherwise;
- $D$ will be 1 if the doors are closed and 0 otherwise;
- $B$ will be 1 if the seat belts are fastened and 0 otherwise.

The circuit is to produce a single output $R$ which should be 1 if the warning bell should ring and 0 otherwise. Build a circuit for this system.

# 3.5 Making Computers Add

In this section we consider the problem of constructing a circuit which will add two integers. To do this, we must first understand how integers are represented and manipulated by a computer using just the binary digits.

## 3.5.1 Binary Numbers

People have ten fingers, and children learn early on to count using the ten digits on their hands. When counting beyond ten on your fingers, the natural thing to do is to keep track of how many times you run through your fingers, which you can ask someone else to do using their ten fingers. Then a third person in turn can use their ten fingers to keep track of how many times the second person runs through all of their fingers, which happens every time you reach 100 (i.e., each time you run through your own ten fingers ten times). When the third person runs out of fingers, you will have counted ten lots of 100, i.e. up to 1000. If you are still counting, a fourth person can use their ten fingers to keep track of how many lots of 1000 you have counted. A fifth person can then keep track of how many lots of 10000 are counted; a sixth how many lots of 100000; etc.

This mechanism for counting is reflected in our use of decimal numbers, which is a positional notation for expressing quantities. For example, when we write the decimal number 6538, we interpret the four digits as follows:

- the 8 in the rightmost position represents 8 ones;
- the 3 in the second position from the right represents 3 lots of tens;
- the 5 in the third position from the right represents 5 lots of hundreds (ie, tens of tens); and
- the 6 in the fourth position from the right represents 6 lots of thousands (ie, tens of hundreds, or tens of tens of tens).

That is,

$$\begin{array}{rrcrcr} 6538 = & 6 \times 10^3 & = & 6 \times 1000 & = & 6000 \\ + & 5 \times 10^2 & = & 5 \times 100 & = & 500 \\ + & 3 \times 10^1 & = & 3 \times 10 & = & 30 \\ + & 8 \times 10^0 & = & 8 \times 1 & = & \underline{\quad 8} \\ & & & & & 6538 \end{array}$$

Digital computers have access to only the two binary digits, 0 and 1, not the ten decimal digits. Therefore, they naturally represent quantities as binary numbers rather than decimal numbers, which are sequences of binary digits (bits) rather than decimal digits. For example, the binary number 11101 is interpreted as follows:

- the 1 in the rightmost position represents 1 one;
- the 0 in the second position from the right represents 0 lots of twos;
- the 1 in the third position from the right represents 1 lot of fours (ie, twos of twos);
- the 1 in the fourth position from the right represents 1 lot of eights (ie, twos of fours, or twos of twos of twos); and
- the 1 in the fifth position from the right represents 1 lot of sixteens (ie, twos of eights, or twos of twos of twos of twos).

That is,

$$\begin{array}{rrrrrrr} 11101 & = & 1 \times 2^4 & = & 1 \times 16 & = & 16 \\ & + & 1 \times 2^3 & = & 1 \times 8 & = & 8 \\ & + & 1 \times 2^2 & = & 1 \times 4 & = & 4 \\ & + & 0 \times 2^1 & = & 0 \times 2 & = & 0 \\ & + & 1 \times 2^0 & = & 1 \times 1 & = & 1 \\ & & & & & & 29 \end{array}$$

Any natural number can be represented as a binary number, just as easily as it can be represented as a decimal number. The method for translating from binary to decimal can be extracted from the above description; and the method for translating from decimal to binary is almost as easy: we merely have to keep subtracting from the number in question the largest power of two that we can.

**Example 3.16**

To translate the decimal value 51 into binary:

- subtract $2^5 = 32$ from 51 to give a remainder of 19;
- subtract $2^4 = 16$ from 19 to give a remainder of 3;
- subtract $2^1 = 2$ from 3 to give a remainder of 1;
- subtract $2^0 = 1$ from 1 to give a remainder of 0.

We can thus express the decimal number $51 = 32 + 16 + 2 + 1$ in binary as:

$$\begin{array}{rrrrrrr} 110011 & = & 1 \times 2^5 & = & 1 \times 32 & = & 32 \\ & + & 1 \times 2^4 & = & 1 \times 16 & = & 16 \\ & + & 0 \times 2^3 & = & 0 \times 8 & = & 0 \\ & + & 0 \times 2^2 & = & 0 \times 4 & = & 0 \\ & + & 1 \times 2^1 & = & 1 \times 2 & = & 2 \\ & + & 1 \times 2^0 & = & 1 \times 1 & = & 1 \\ & & & & & & 51 \end{array}$$

### 3.5.2 Adding Binary Numbers

Consider how we would naturally add two (decimal) numbers by hand. We would first line the numbers up one on top of the other. Then we would add the units, writing down the unit sum digit and moving the carry digit (if there is one) to the top of the tens column; then we would add the three numbers in the tens column, writing down the tens sum digit and moving the carry digit to the top of the hundreds column; then we would add the three numbers in the hundreds column, writing down the hundreds sum digit and moving the carry digit to the top of the thousands column; and continue doing this same calculation with each column from right to left.

This same method works equally well for binary numbers, and is the basis for how digital computers add numbers represented in binary.

**Example 3.17**

To add the two binary numbers 11101 and 10110, write them one over the other and add the bits column-wise from right to left, including carries where necessary, as indicated:

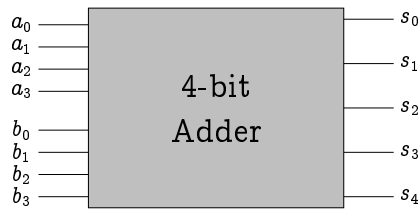
$$\begin{array}{r} {\scriptstyle 1\ \ 1\ \ 1\ \ \ \ \ } \\ 1\,1\,1\,0\,1 \\ 1\,0\,1\,1\,0 \\ \hline 1\,1\,0\,0\,1\,1 \end{array}$$

The first two columns on the right each gives a sum of 1 with no carry; but the third column from the right give a 0 sum with a carry, as then does the fourth column. The fifth column gives a sum of 1 with a carry, which gives a sum of 1 for the new sixth column.

**Exercise 3.17** (Solution on page 424)

What decimal sum is being calculate in the above example?

We are now in a position to design a digital circuit which adds two integers represented as binary numbers. More specifically, we shall build a circuit which will have 8 input lines representing two 4-bit binary numbers $a_3a_2a_1a_0$ and $b_3b_2b_1b_0$, and 5 output lines representing the 5-bit binary number $s_4s_3s_2s_1s_0$ resulting from adding $a_3a_2a_1a_0$ and $b_3b_2b_1b_0$:



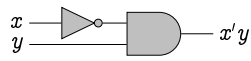
The construction we give can be easily scaled up to add arbitrarily-long bit strings.

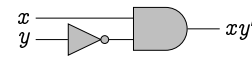
### 3.5.3 Building Half Adders

The basic component from which we shall build our 4-bit adder is the circuit HA from Example 3.12 (page 95), which takes the two inputs $x$ and $y$ and produces the two outputs $s$ and $c$ representing the sum of $x$ and $y$, with $s$ being the sum bit and $c$ being the carry bit. Such a circuit is called a ***half adder***.

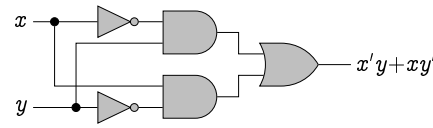
Our first task is to express the outputs in terms of the functions of the basic gates. For a start, computing the carry bit $c$ is obvious: being 1 exactly when both $x$ and $y$ are 1, it is their product $c = xy$. The sum bit $s$ is only slightly more cumbersome. It is 1 when one of the inputs is 1 and the other is 0: $s = x'y+xy'$.

Towards building these functions in a circuit using the three basic gates, we can first note the following two circuits that compute $x'y$ and $xy'$, respectively:

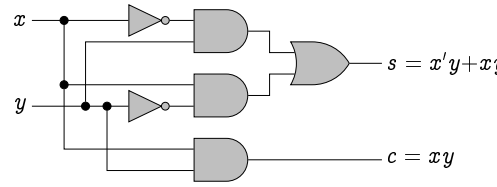



These can then be combined to give a circuit for $x'y+xy'$ as follows:




The above circuit computes the sum bit of the half adder; it only remains to add a further AND gate which computes the carry bit to complete the circuit:




This circuit consists of six gates: three AND gates, two NOT gates and one OR gate. The question then arises: is it possible to build a simpler circuit which performs the computation of a half adder. Such questions are important when contemplating fitting ever-more computing power on a computer chip; you would certainly want to find the smallest possible circuits to compute the functions that are implemented on the chip.

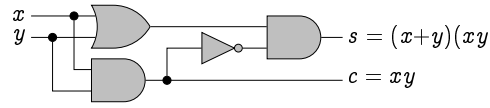
Using the laws of Boolean algebra, we can make the following calculation:

$$\begin{aligned}
x'y + xy' &= xy' + x'y && \textit{(commutativity)} \\
&= (xy + x'y')' && \textit{(Exercise 3.10(1))} \\
&= (xy)'(x'y')' && \textit{(DeMorgan1)} \\
&= (xy)'(x'' + y'') && \textit{(DeMorgan2)} \\
&= (xy)'(x + y) && \textit{(Involution, twice)} \\
&= (x+y)(xy)' && \textit{(commutativity)}
\end{aligned}$$

This complicated calculation, in fact, tells us something natural: that having one input line holding the value 1 and the other holding the value 0: $x'y+xy'$ is the same as having one of the input lines holding the value 1 and not having both input lines holding the value 1: $(x+y)(xy)'$.

Indeed, such intuitive observations are where ideas for optimisations typically arise. The above derivation was a necessary step in the design process, in justifying the intuition which suggested the optimisation.

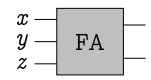
Importantly, the final expression $(x+y)(xy)'$ is simpler to evaluate than $x'y+xy'$, requiring only four basic operations rather than five; moreover, the product $xy$, is calculated in the process, so we need no further operations to complete the half adder circuit. The corresponding circuit is as follows:




We have thus managed to build the half adder using four gates instead of six. Improving designs like this in order to reduce the number of gates – in this case by a third – is of obvious importance when it comes to fitting more power within the limited space on a circuit board. Reasoning with Boolean algebra is a crucial activity in the design of computer processors.

### 3.5.4 Building Full Adders

We are designing a circuit which will add two binary numbers using the usual method of summing bits column-by-column. So far we have constructed a half adder which takes two bits and adds these together, producing a sum bit and a carry bit. However, we will also need a circuit which adds not just two digits, as the half adder does, but rather three digits, to cater for the carry bit. Such a circuit is called a ***full adder*** and has the following form:




The input wires $x$, $y$ and $z$ each have the value 0 or 1, and sum up to either 0, 1, 2 or 3, which is reflected in the output wires $s$ and $c$.

The sum bit $s$ will be 1 if exactly one or all three of the input bits $x$, $y$ and $z$ are 1:

$$s = xy'z' + x'yz' + x'y'z + xyz;$$

and the carry bit will be 1 if at least two of the input bits are 1:

$$c = xyz' + xy'z + x'yz + xyz.$$

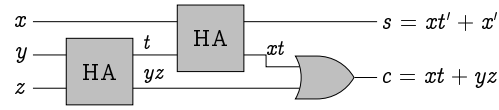
Letting $t = yz'+y'z$ be the value of the sum bit from a half adder with inputs $y$ and $z$, and noting from Exercise 3.10(1) that $t' = yz+y'z'$, we can note that

$$\begin{aligned} s &= xy'z' + x'yz' + x'y'z + xyz \\ &= x'(yz'+y'z) + x(yz+y'z') \qquad \textit{(commutativity/distributivity)} \\ &= x't + xt' \end{aligned}$$

and

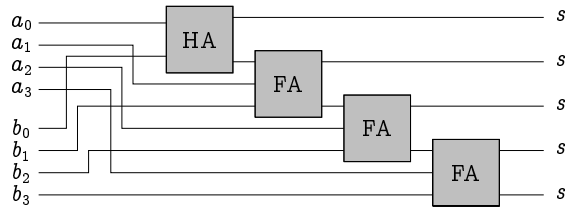
$$\begin{aligned} c &= xyz' + xy'z + x'yz + xyz \\ &= x(yz'+y'z) + yz(x+x') \qquad \textit{(distributivity)} \\ &= xt + yz \qquad \textit{(identity)} \end{aligned}$$

These outputs are generated by combining two half adders and an OR gate as follows:




### 3.5.5 Putting It All Together

Having defined a full adder, adding two $n$-bit numbers is then achieved by stringing $n$ such full adders together. In particular, to build our 4-bit adder, which adds together the two 4-bit binary numbers $a_3a_2a_1a_0$ and $b_3b_2b_1b_0$ to produce the 5-bit binary number $s_4s_3s_2s_1s_0$, we would use the following circuit:

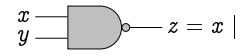



We start with a half adder, as we don't have to worry about a carry bit for the first two bits $a_0$ and $b_0$. Of course, stringing more full adders together would allow larger values to be added, meaning that this circuit can be easily scaled up.

## 3.6 Additional Exercises

1. Prove the second Further Distributive Law, the second Idempotence Law, the second Absorption Law, and the second Domination Law, all using the Laws of Boolean Algebra. (That is, do not rely on the Duality Principle.)
2. Prove that the set $S = \{1, 2, 5, 10\}$ of divisors of 10 is a Boolean algebra with zero 1 and unit 10, with $x + y$ interpreted as the least common multiple of $x$ and $y$, $\mathrm{lcm}(x, y)$; $xy$ interpreted as the greatest common factor of $x$ and $y$, $\gcd(x, y)$; and $x' = 10/x$.
3. Prove that if we take $S = \{1, 2, 3, 6, 12\}$ to be the set of divisors of 12 in Exercise 2 above, then we would *not* get a Boolean algebra.
4. Does the finite powerset $\mathcal{P}_{\mathrm{fin}}(U)$ of a set $U$ give rise to a Boolean algebra (with, as usual, the roles of 0, 1, $+$, $\times$ and $\cdot'$ taken by $\emptyset$, $U$, $\cup$, $\cap$ and $\bar{\cdot}$, respectively)? Justify your answer.
5. (a) Prove that $xy' = 0$ if, and only if, $x' + y = 1$.
   (b) State and prove the dual of the theorem in part (a).
6. (a) Prove that $x = y$ if, and only if, $xy' + x'y = 0$.
   (b) State and prove the dual of the theorem in part (a).
7. The ***NAND gate*** has the following definition (and symbol):

| $x$ | $y$ | $x \mid y$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

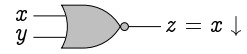



   That is, the output $z = x \mid y$ has the value 0 if both of the inputs $x$ or $y$ have the value 1; otherwise it has the value 1.

   (a) Build a circuit using AND, OR and NOT gates which implements this operator.
   (b) Show how to build circuits for computing $x'$, $x+y$ and $xy$ only using NAND gates.

8. The ***NOR gate*** has the following definition (and symbol):

| $x$ | $y$ | $x \downarrow y$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |



That is, the output $z = x \downarrow y$ has the value 1 if neither of the inputs $x$ nor $y$ has the value 1; otherwise it has the value 0.

(a) Build a circuit using AND, OR and NOT gates which implements this operator.

(b) Show how to build circuits for computing $x'$, $x+y$ and $xy$ only using NOR gates.

9. Build circuits which implement the following Boolean expressions.

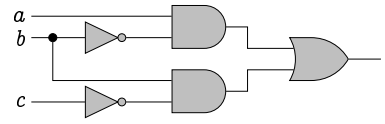
(a) $(a + b)(b + c)$

(b) $a'b + (b + c)'$

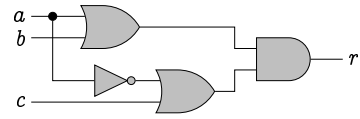
(c) $(ab)' + (bc)'$

10. Describe the behaviour of the following circuits by providing a Boolean expression and a truth table defining the output value $X$.

(a)



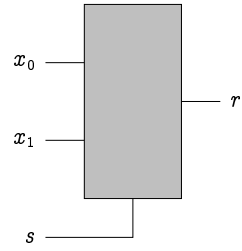

(b)




11. Joel, Felix and Oskar are using a simple voting machine to cast secret ballots to decide which DVD to watch tonight, the choice being between the latest Final Destination film and the new Fockers film. Each of them will vote either "0" for Final Destination or "1" for the Fockers; and they will then watch whichever film receives the majority of the three votes.

Build a circuit which accepts three inputs $J$, $F$ and $O$ representing their respective votes, and produces one output $X$ representing the outcome of the election.

12. A ***multiplexer*** is a circuit with three input lines $x_0$, $x_1$ and $s$, and one output line $r$, defined as follows:




| $s$ | $x_0$ | $x_1$ | $r$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

The $s$ line acts as a ***selector***; the value of the output $r$ will be either that of $x_0$ or that of $x_1$, depending on the value of $s$.

Build a circuit which implements this multiplexer.

(Hint: First argue that $r = s'x_0 + sx_1$.)