3. BOOLEAN ALGEBRAS

Contents

1. Definition of Boolean algebras

I shall begin by listing some logical equivalences in propositional logic which were proved in the exercises. I continue notation introduced there by using **1** to mean any tautology and **0** to mean any contradiction.

(1) $(p \vee q) \vee r \equiv p \vee (q \vee r)$.
(2) $p \vee q \equiv q \vee p$.
(3) $p \vee \mathbf{0} \equiv p$.
(4) $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$.
(5) $p \wedge q \equiv q \wedge p$.
(6) $p \wedge \mathbf{1} = p$.
(7) $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$.
(8) $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$.
(9) $p \vee \neg p \equiv \mathbf{1}$.
(10) $p \wedge \neg p \equiv \mathbf{0}$.

If we replace $\equiv$ by $=$ we have exactly the definition of a Boolean algebra.

A *Boolean algebra* is defined by the following data

$$(B, +, \cdot, \bar{\ }, 0, 1)$$

where B is a set that carries the structure, $+$ and $\cdot$ are binary operations, meaning that they have two, ordered inputs and one output, $a \mapsto \bar{a}$ is a unary operation, meaning that it has one input and one output, and two special elements of B, namely 0 and 1. In addition, the following ten axioms are required to hold.

(B1): $(x + y) + z = x + (y + z)$.
(B2): $x + y = y + x$.
(B3): $x + 0 = x$.

(B4): $(x \cdot y) \cdot z = x \cdot (y \cdot z)$.
(B5): $x \cdot y = y \cdot x$.
(B6): $x \cdot 1 = x$.
(B7): $x \cdot (y + z) = x \cdot y + x \cdot z$.
(B8): $x + (y \cdot z) = (x + y) \cdot (x + z)$.
(B9): $x + \bar{x} = 1$.
(B10): $x \cdot \bar{x} = 0$.

These axioms are organized as follows. The first group of three (B1), (B2), (B3) deals with the properties of + on its own: brackets, order, special element. The second group of three (B4), (B5), (B6) deals with the properties of $\cdot$ on its own: brackets, order, special element. The third group (B7), (B8) deals with how + and $\cdot$ interact, with axiom (B8) being odd looking. The final group (B9), (B10) deal with the properties of $a \mapsto \bar{a}$, called *complementation*. On a point of notation, we shall usually write xy rather than $x \cdot y$.

Do I have to remember these axioms? No — in the exam paper I will list these axioms if they are needed. You do, however, have to familiarize yourself with them and learn how to use them.

The Boolean algebra associated with PL is called the *Lindenbaum algebra*. The only issue is how to convert $\equiv$ into $=$. Denote by **Prop** the set of all wff in propositional logic. There are two answers.

Answer 1. Not correct but not far off. The set **Prop** is *essentially* a Boolean algebra. You need to insert the word 'essentially' in order to cover yourself to avoid legal action by outraged Boolean algebraists. If you give this answer you need to convey the impression that you could say more if you chose — only you don't choose to.

Answer 2. Correct. Logical equivalence is an equivalence relation on the set **Prop** and is a congruence with respect to the operations $\vee$, $\wedge$ and $\neg$. The quotient of **Prop** by $\equiv$ is then the Boolean algebra. This is the correct answer but needs a lot of unpacking. You need to know what an equivalence relation is as well as a congruence. If you want to know more I can tell you.

2. Set theory

Sets are another example of a data type. I need them for two reasons here. First, to enable me to describe an important class of examples of Boolean algebras. Second, I shall need sets when I come to discuss first-order logic.

Set theory, invented by Georg Cantor (1845–1918) in the last quarter of the nineteenth century, provides a precise language for doing mathematics. The starting point of set theory are the following two deceptively simple definitions:

- A *set* is a collection of objects which we wish to regard as a whole. The members of a set are called its *elements*.
- Two sets are *equal* precisely when they have the same elements.

We often use capital letters to name sets: such as A, B, or C or fancy capital letters such as $\mathbb{N}$ and $\mathbb{Z}$. The elements of a set are usually denoted by lower case letters. If x *is an element of* the set A then we write

$$x \in A$$

and if x *is not an element of* the set A then we write

$$x \notin A.$$

This means of course

$$\neg(x \in A).$$

A set should be regarded as a bag of elements, and so the order of the elements within the set is not important. In addition, repetition of elements is ignored.[1]

Examples 2.1.

(1) The following sets are all equal:

$$\{a,b\}, \quad \{b,a\}, \quad \{a,a,b\}, \quad \{a,a,a,a,b,b,b,a\}$$

because the order of the elements within a set is not important and any repetitions are ignored. Despite this it is usual to write sets without repetitions to avoid confusion. We have that $a \in \{a,b\}$ and $b \in \{a,b\}$ but $\alpha \notin \{a,b\}$.

(2) The set $\{\}$ is empty and is called the *empty set*. It is given a special symbol $\emptyset$, which is taken from Danish and is the first letter of the Danish word meaning 'empty'. Remember that $\emptyset$ means the same thing as $\{\}$. Take careful note that $\emptyset \neq \{\emptyset\}$. The reason is that the empty set contains no elements whereas the set $\{\emptyset\}$ contains one element. By the way, the symbol for the emptyset is different from the Greek letter phi: ϕ or Φ.

[1] If you want to take account of repetitions you have to use *multisets*.

The number of elements in a set is called its *cardinality*. If X is a set then $|X|$ denotes its cardinality. A set is *finite* if it only has a finite number of elements, otherwise it is *infinite*. If a set has only finitely many elements then we might be able to list them if there aren't too many: this is done by putting them in 'curly brackets' { and }. We can sometimes define infinite sets by using curly brackets but then, because we can't list all elements in an infinite set, we use '...' to mean 'and so on in the obvious way'. This can also be used to define finite sets where there is an obvious pattern. Often, we describe a set by saying what properties an element must have to belong to the set. Thus

$$\{x\colon\ P(x)\}$$

means 'the set of all things x which satisfy the condition P'. Here are some examples of sets defined in various ways.

Examples 2.2.

(1) $D =$ { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday }, the set of the days of the week. This is a small finite set and so we can conveniently list its elements.

(2) $M =$ { January, February, March, ..., November, December }, the set of the months of the year. This is a finite set but I didn't want to write down all the elements so I wrote '...' to indicate that there were other elements of the set which I was too lazy to write down explicitly but which are, nevertheless, there.

(3) $A = \{x\colon\ x \text{ is a prime number}\}$. I define a set by describing the properties that the elements of the set must have.

Sets can be complicated. In particular, a set can contain elements which are themselves sets. For example, $A = \{\{a\}, \{a, b\}\}$ is a set whose elements are $\{a\}$ and $\{a, b\}$ which both happen to be sets. Thus $\{a\} \in \{\{a\}, \{a, b\}\}$.

The following notation will be useful to us later.

Examples 2.3.

(1) The set $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ of all *natural numbers*.

(2) The set $\mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$ of all *integers*. The reason $\mathbb{Z}$ is used to designate this set is because 'Z' is the first letter of the word 'Zahl', the German for number.

(3) The set $\mathbb{Q}$ of all *rational numbers* i.e. those numbers that can be written as fractions whether positive or negative.

(4) The set $\mathbb{R}$ of all *real numbers* i.e. all numbers which can be represented by decimals with potentially infinitely many digits after the decimal point.

Given a set A, a new set B can be formed by *choosing* elements from A to put in B. We say that B is a *subset* of A, which is written $B \subseteq A$. If $A \subseteq B$ and $A \neq B$ then we say that A is a *proper* subset of B. Observe that two sets are equal precisely when the following two conditions hold

(1) $A \subseteq B$.
(2) $B \subseteq A$.

This is often the best way of showing that two sets are equal.

Examples 2.4.

(1) $\emptyset \subseteq A$ for every set A, where we choose no elements from A. It is a very common mistake to forget the empty set when listing subsets of a set.
(2) $A \subseteq A$ for every set A, where we choose all the elements from A.
(3) $\mathbb{N} \subseteq \mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R}$.
(4) $\mathbb{E}$, the set of even natural numbers, is a subset of $\mathbb{N}$.
(5) $\mathbb{O}$, the set of odd natural numbers, is a subset of $\mathbb{N}$.
(6) $\mathbb{P} = \{2, 3, 5, 7, 11, 13, 17, 19, 23, \ldots\}$, the set of *primes*, is a subset of $\mathbb{N}$.
(7) $A = \{x \colon x \in \mathbb{R} \text{ and } x^2 = 4\}$ which is just the set $\{-2, 2\}$.

Operations on sets

There are three operations defined on sets using the words 'and', 'or' and 'not' where these are defined as in PL. Let A and B be sets.

We can construct a new set, called the *intersection* of A and B and denoted by $A \cap B$, whose elements consist of all those elements that belong to both A and to B.

We can construct a new set, called the *union* of A and B and denoted by $A \cup B$, whose elements consist of all the elements of A together with all the elements of B. In constructing unions of sets, we remember that repetitions don't count.

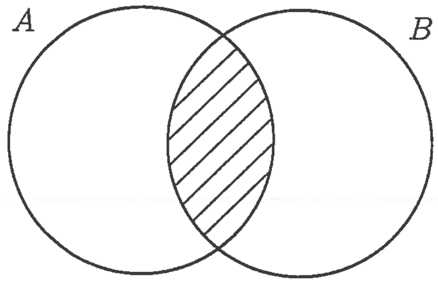
We can construct a new set, called the *difference* or *relative complement* of A and B and denoted by $A \setminus B$,[2] whose elements consist of all those elements that belong to A but not to B.

When illustrating definitions involving sets or trying to gain an idea of what's true about sets in general, we often use *Venn diagrams.* In a Venn diagram, a set is represented by a region in the plane. The intersection of two sets can then be represented by the overlap of the two regions representing each of the sets, and the union is represented by

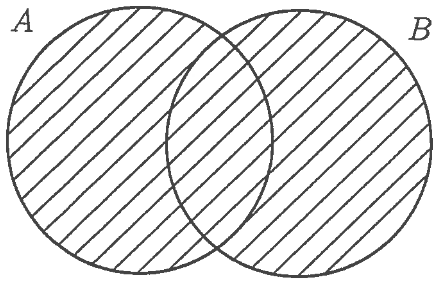
[2] Sometimes denoted by $A - B$.

the region enclosed by both regions. Although Venn diagrams cannot be used to prove results about sets, they are a nice way of visualising sets and their properties.[3]

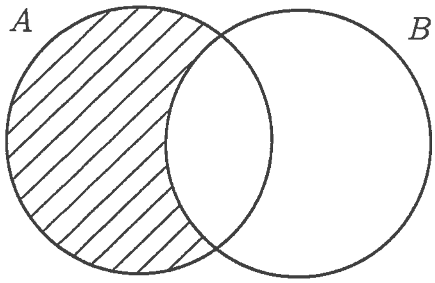
[3] With thanks to Simone Rea for the Venn diagrams



$A \dot{\cap} B$



$A \cup B$



$A \setminus B$

Example 2.5. Let $A = \{1, 2, 3, 4\}$ and $B = \{3, 4, 5, 6\}$. Find $A \cap B$, $A \cup B$, $A \setminus B$ and $B \setminus A$.

Let's begin with $A \cap B$. We have to find the elements that belong to both A and B. We start with the elements in A and work left-to-right: 1 doesn't belong to B; 2 doesn't belong to B; 3 does belong to B as does 4. Thus $A \cap B = \{3, 4\}$.

To find $A \cup B$ we join the two sets together $\{1, 2, 3, 4, 3, 4, 5, 6\}$ and then read from left-to-right weeding out repetitions to get $A \cup B = \{1, 2, 3, 4, 5, 6\}$.

To calculate $A \setminus B$ we have to find the elements of A that don't belong to B. So read the elements of A from left-to-write comparing them with the elements of B: 1 doesn't belong to B; 2 doesn't belong to B: but 3 and 4 do belong to B. It follows that $A \setminus B = \{1, 2\}$.

To calculate $B \setminus A$ we have to find the set of elements of B that don't belong to A: this set is just $\{5, 6\}$.

Examples 2.6.

(1) $\mathbb{E} \cap \mathbb{O} = \emptyset$. This says that there is no number which is both odd and even.
(2) $\mathbb{P} \cap \mathbb{E} = \{2\}$. This says that the only even prime number is 2.
(3) $\mathbb{E} \cup \mathbb{O} = \mathbb{N}$. This says that every natural number is either odd or even.

The power set of a set

The set whose elements are all the subsets of X is called the *power set* of X and is denoted by $\mathsf{P}(X)$.

Warning! The power set of a set X contains both $\emptyset$ and X as elements.

Example 2.7. Let's find all the subsets of the set $X = \{a, b, c\}$. First we have the subset with no elements, the emptyset. Then we have the subsets that contain exactly one element: $\{a\}, \{b\}, \{c\}$. Then the subsets containing exactly two elements: $\{a, b\}, \{a, c\}, \{b, c\}$. Finally, we have the whole set X. It follows that X has 8 subsets and

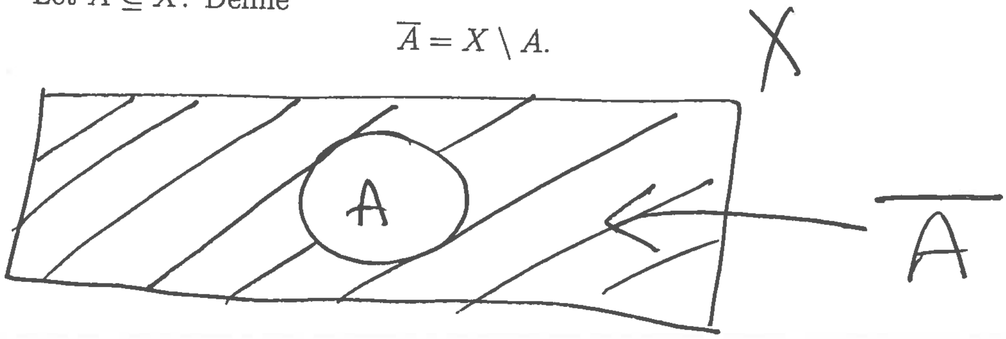
$$\mathsf{P}(X) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, X\}.$$

Proposition 2.8. *Let X be a finite set with n elements. Then*

$$|\mathsf{P}(X)| = 2^n.$$

Let $A \subseteq X$. Define

$$\overline{A} = X \setminus A.$$



Proposition 2.9. *Let X be any set. Then*

$$(\mathsf{P}(X), \cup, \cap, A \mapsto \bar{A}, \emptyset, X)$$

is a Boolean algebra.

I shall now sketch out a proof of this result.

We have to prove that this structure really does satisfy the axioms for a Boolean algebra. To do this, we shall use our logical equivalences for PL. Here is a sample proof. We shall prove that (B7) holds. That is, we shall prove

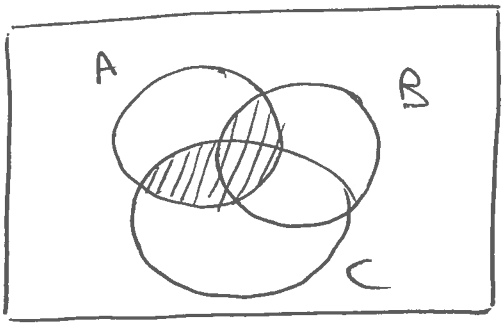
that

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

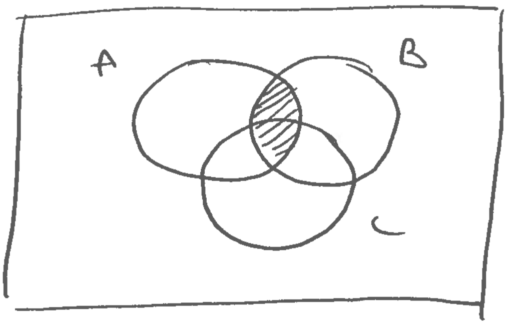
for all $A, B, C \subseteq X$

I shall use Venn diagrams first to illustrate the result.

$A \cap (B \cup C)$

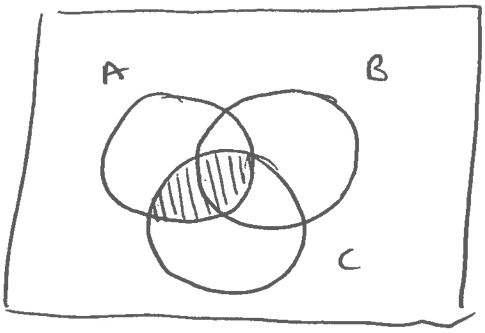


$A \cap B$

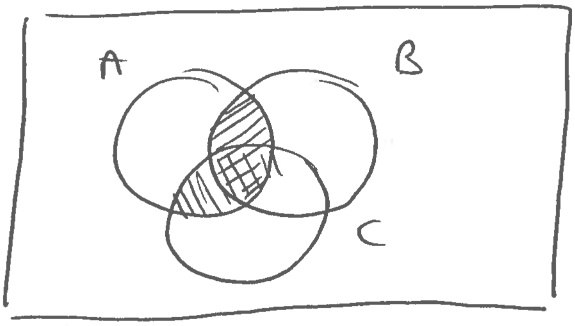


=

$A \cap C$



$(A \cap B) \cup (A \cap C)$



Now we prove the result using PL.

$x \in A \cap (B \cup C)$

iff $(x \in A) \wedge (x \in B \cup C)$

iff $(x \in A) \wedge ((x \in B) \vee (x \in C))$

We now use the fact that

$$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r).$$

It follows that

$$(x \in A) \wedge ((x \in B) \vee (x \in C))$$

$$\equiv$$

$$[(x \in A) \wedge (x \in B)] \vee [(x \in A) \wedge (x \in C)]$$

But $(x \in A) \wedge (x \in B)$ iff $x \in A \cap B$

and

$(x \in A) \wedge (x \in C)$ iff $x \in A \cap C$

It follows that

$x \in (A \cap B) \cup (A \cap C)$ as required.

I shall not prove the following theorem. I state it only to give a little context.

Theorem 2.10.

(1) *Any finite Boolean algebra has 2^n elements for some natural number n.*

(2) *Any two finite Boolean algebras with the same number of elements are 'essentially the same' or isomorphic.*

The 2-element Boolean algebra $\mathbb{B}$ is particularly important. It is defined as follows. Put $\mathbb{B} = \{0, 1\}$. We define operations $a \mapsto \bar{a}$, $\cdot$, and $+$ by means of the following tables; they are the same as the truthtable for $\neg$, $\wedge$ and $\vee$ except that we replace T by 1 and F by 0.

x	$\bar{x}$
1	0
0	1

x	y	$x \cdot y$
1	1	1
1	0	0
0	1	0
0	0	0

x	y	$x + y$
1	1	1
1	0	1
0	1	1
0	0	0

Proposition 2.11. *With the definitions above*

$$(\mathbb{B}, +, \cdot, a \mapsto \bar{a}, 0, 1)$$

is a Boolean algebra.

Proving results about Boolean algebras

Boolean algebras are defined by the ten axioms we gave above. In fact, they satisfy many more properties, but these can all be proved from these axioms. I shall give an example of such a property and its accompanying proof here and let you prove some more in the exercise sheet. In the result below, I write b^2 to mean bb.

Lemma 2.12. *In any Boolean algebra, we have that $b^2 = b$.*

Proof.

$$\begin{aligned} b &= b1 \text{ by (B6)} \\ &= b(b + \bar{b}) \text{ by (B9)} \\ &= bb + b\bar{b} \text{ by (B7)} \\ &= bb + 0 \text{ by (B10)} \\ &= bb \text{ by (B3)}\ . \end{aligned}$$

□

Another process that we can carry out that is proof-based is *simplification.* For example, the complicated Boolean expression

$$x + yz + \bar{x}y + \bar{y}xz$$

is in fact equal to the much simpler expression

$$x + y.$$

This, and other similar examples, will be discussed in the exercises.

3. Binary arithmetic

A *string* is simply an ordered sequence of symbols. Strings are used to describe all kinds of data structures. In this section, we shall focus on strings used to represent numbers, specifically natural numbers. In the lecture, I shall restrict myself to the binary representation of numbers but here I give the general case.

From tallies to the Hindu-Arabic number system

The simplest way of writing a number down is to use a mark like |, called a *tally*, for each thing being counted. So

||||||||||

means 10 things. This system has advantages and disadvantages. The advantage is that you don't have to go on a training course to learn it. The disadvantage is that even quite small numbers need a lot of space like

||||||||||||||||||||||||||||||||||||||

It's also hard to tell whether

|||||||||||||||||||||||||||||||||||||||

is the same number or not. (It's not.)

It's inevitable that people will introduce abbreviations to make the system easier to use. Perhaps it was in this way that the next development occurred. Both the ancient Egyptians and Romans used similar systems but I'll describe the Roman system because it involves letters rather than pictures. First, you have a list of basic symbols:

number	1	5	10	50	100	500	1000
symbol	I	V	X	L	C	D	M

There are more symbols for bigger numbers but we won't worry about them. Numbers are then written according to the *additive principle.* Thus MMVIIII is 2009. Incidently, I understand that the custom of also using a *subtractive principle* so that, for example, IX means 9 rather than using VIIII, is a more modern innovation.

This system is clearly a great improvement on the tally-system. Even quite big numbers are written compactly and it is easy to compare numbers. On the other hand, there is a bit more to learn. The other disadvantage is that we need separate symbols for different powers of 10 and their multiples by 5. This was probably not too inconvenient in the ancient world where it is likely that the numbers needed on a day-to-day basis were never going to be that big. A common criticism of this system is that it is hard to do multiplication in. However, that turns out to be a non-problem because, like us, the Romans used pocket calculators or, more accurately, a toga-friendly device called an *abacus*. The real evidence for the usefulness of this system of writing numbers is that it survived for hundreds and hundreds of years.

The system used throughout the world today, called the *Hindu-Arabic number system*, seems to have been in place by the ninth century in India but it was hundreds of years in development and the result of ideas from many different cultures. The invention of zero on its own is one of the great steps in human intellectual development. The genius of the system is that it requires only 10 symbols

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9$$

and every natural number can be written using a sequence of these symbols. The trick to making the system work is that we use the *position* on the page of a symbol to tell us what number it means. Thus 2009 means

10^3	10^2	10^1	10^0
2	0	0	9

In other words

$$2 \times 10^3 + 0 \times 10^2 + 0 \times 10^1 + 9 \times 10^0.$$

Notice the important role played by the symbol 0 which makes it clear which column a symbol belongs in otherwise we couldn't tell 29 from 209 from 2009. The disadvantage of this system is that you *do* have to go on a course to learn it because it is a highly sophisticated way of writing numbers. On the other hand, it has the enormous advantage that any number can be written down in a compact way.

Once the basic system had been accepted it could be adapted to deal not only with positive whole numbers but also negative whole numbers, using the symbol $-$, and also fractions with the introduction of the decimal point. By the end of the sixteenth century the full decimal system was in place.

Number bases

only 10 ⟺ 2 needed

We shall now look in more detail at the way in which numbers can be written down using a positional notation. In order not to be biased, we shall not just work in base 10 but show how any base can be used. Base 10 probably arose for biological reasons since we have ten fingers.

Given natural numbers a and b, there are unique integers q and r such that

$$a = bq + r$$

where $0 \leq r < b$. The number q is called the *quotient* and the number r is called the *remainder*. For example, if we consider the pair of natural numbers 14 and 3 then

$$14 = 3 \cdot 4 + 2$$

where 4 is the quotient and 2 is the remainder. There's nothing new here except possibly the terminology.

Let a and b be integers. We say that a *divides* b or that b *is divisible by* a if there is a q such that $b = aq$. In other words, there is no remainder. We also say that a is a *divisor or factor* of b. We write $a \mid b$ to mean the same thing as 'a divides b'.[4]

Warning! $a \mid b$ does not mean the same thing as $\frac{a}{b}$. The latter is a number, the former is a statement about two numbers.

Let's see how to represent numbers in *base* b where $b \geq 2$. If $d \leq 10$ then we represent numbers by sequences of symbols taken from the set

$$\mathbb{Z}_d = \{0, 1, 2, 3, \ldots d - 1\}$$

but if $d > 10$ then we need new symbols for 10, 11, 12 and so forth. It's convenient to use A,B,C, ... For example, if we want to write numbers in base 12 we use the set of symbols

$$\{0, 1, \ldots, 9, A, B\}$$

whereas if we work in base 16 we use the set of symbols

$$\{0, 1, \ldots, 9, A, B, C, D, E, F\}.$$

If x is a sequence of symbols then we write x_d to make it clear that we are to interpret this sequence as a number in base d. Thus BAD_{16} is a number in base 16.

[4] Observe that if a is nonzero, then $a \mid a$, if $a \mid b$ and $b \mid a$ then $a = \pm b$, and finally if $a \mid b$ and $b \mid c$ then $a \mid c$.

The symbols in a sequence x_d, reading from right to left, tell us the contribution each power of d such as d^0, d^1, d^2, etc makes to the number the sequence represents. Here are some examples.

Examples 3.1. Converting from base d to base 10.

(i): $11A9_{12}$ is a number in base 12. This represents the following number in base 10:

$$\mathbf{1} \times 12^3 + \mathbf{1} \times 12^2 + \mathbf{A} \times 12^1 + \mathbf{9} \times 12^0,$$

which is just the number

$$12^3 + 12^2 + 10 \times 12 + 9 = 2001.$$

(ii): BAD_{16} represents a number in base 16. This represents the following number in base 10:

$$B \times 16^2 + A \times 16^1 + D \times 16^0,$$

which is just the number

$$11 \times 16^2 + 10 \times 16 + 13 = 2989.$$

(iii): 5556_7 represents a number in base 7. This represents the following number in base 10:

$$5 \times 7^3 + 5 \times 7^2 + 5 \times 7^1 + 6 \times 7^0 = 2001.$$

These examples show how easy it is to convert from base d to base 10.

There are two ways to convert from base 10 to base d.

(1) The first runs in outline as follows. Let n be the number in base 10 that we wish to write in base d. Look for the largest power m of d such that $ad^m \leq n$ where $a < d$. Then repeat for $n - ad^m$. Continuing in this way, we write n as a sum of multiples of powers of d and so we can write n in base d.

(2) The second makes use of the remainder theorem. The idea behind this method is as follows. Let

$$n = a_m \dots a_1 a_0$$

in base d. We may think of this as

$$n = (a_m \dots a_1)d + a_0$$

It follows that a_0 is the remainder when n is divided by d, and the quotient is $n' = a_m \dots a_1$. Thus we can generate the digits of n in base d from *right to left* by repeatedly finding the next quotient and next remainder by dividing the current quotient by d; the process starts with our input number as first quotient.

Examples 3.2. Converting from base 10 to base d.

(i): Write 2001 in base 7. I'll solve this question in two different ways: the long but direct route and then the short but more thought-provoking route.

We see that $7^4 > 2001$. Thus we divide 2001 by 7^3. This goes 5 times plus a remainder. Thus $2001 = 5 \times 7^3 + 286$. We now repeat with 286. We divide it by 7^2. It goes 5 times again plus a remainder. Thus $286 = 5 \times 7^2 + 41$. We now repeat with 41. We get that $41 = 5 \times 7 + 6$. We have therefore shown that

$$2001 = 5 \times 7^3 + 5 \times 7^2 + 5 \times 7 + 6.$$

Thus 2001 in base 7 is just 5556.

Now for the short method.

	quotient	remainder
7	2001	
7	285	6
7	40	5
7	5	5
	0	5

Thus 2001 in base 7 is:

$$5556.$$

(ii): Write 2001 in base 12.

	quotient	remainder
12	2001	
12	166	9
12	13	$10 = A$
12	1	1
	0	1

Thus 2001 in base 12 is:

$$11A9.$$

(iii): Write 2001 in base 2.

	quotient	remainder
2	2001	
2	1000	1
2	500	0
2	250	0
2	125	0
2	62	1
2	31	0
2	15	1
2	7	1
2	3	1
2	1	1
	0	1

Thus 2001 in base 2 is (reading from bottom to top):

$$11111010001.$$

Terminology Number bases have some special terminology associated with them which you might encounter:

Base 2: *binary.*
Base 8: *octal.*
Base 10: *decimal.*
Base 12: *duodecimal.*
Base 16: *hexadecimal.*
Base 20: *vigesimal.*
Base 60: *sexagesimal.*

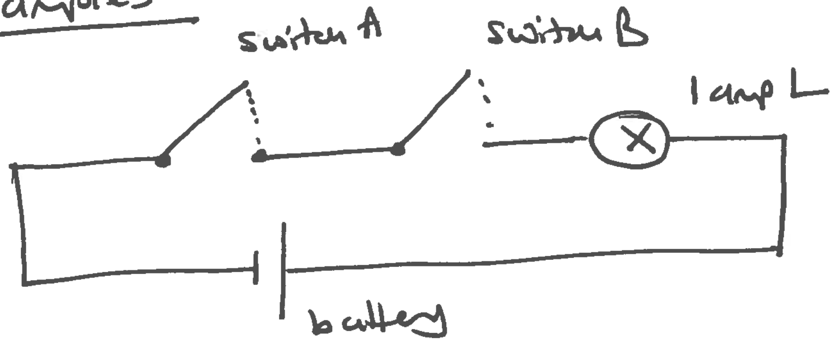
Binary, octal and hexadecimal occur in computer science; there are remnants of a vigesimal system in French and the older Welsh system of counting; base 60 was used by astronomers in ancient Mesopotamia and is still the basis of time measurement (60 seconds = 1 minute, and 60 minutes = 1 hour) and angle measurement.

We shall only work in base 2 and we shall only be interested in binary addition.

3.4 Introduction to circuit design

In 1938, Claude Shannon wrote a master's thesis entitled 'A symbolic analysis of relay and switching circuits'. Shannon noticed something that could have been noticed at any point since Boole introduced Boolean algebra but apparently wasn't.

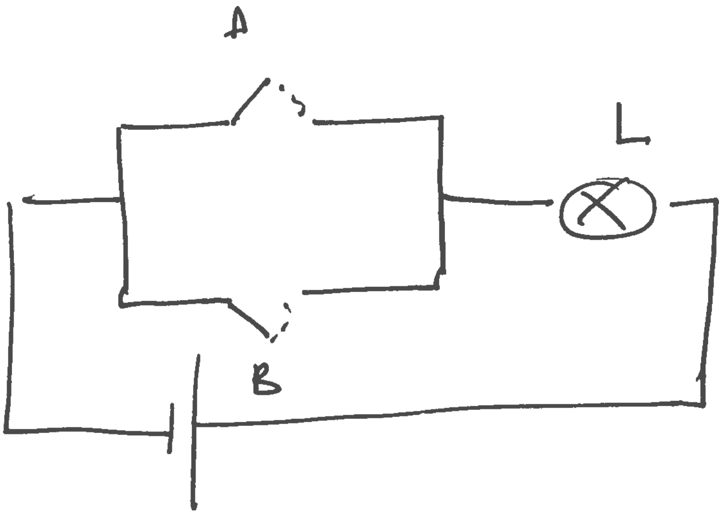
Examples



A	B	L
1	1	1
1	0	0
0	1	0
0	0	0

A = 1 if switch A is closed i.e. current flows through A

L = 1 if current flows through lamp i.e. lamp lights.



A	B	L
1	1	1
1	0	1
0	1	1
0	0	0

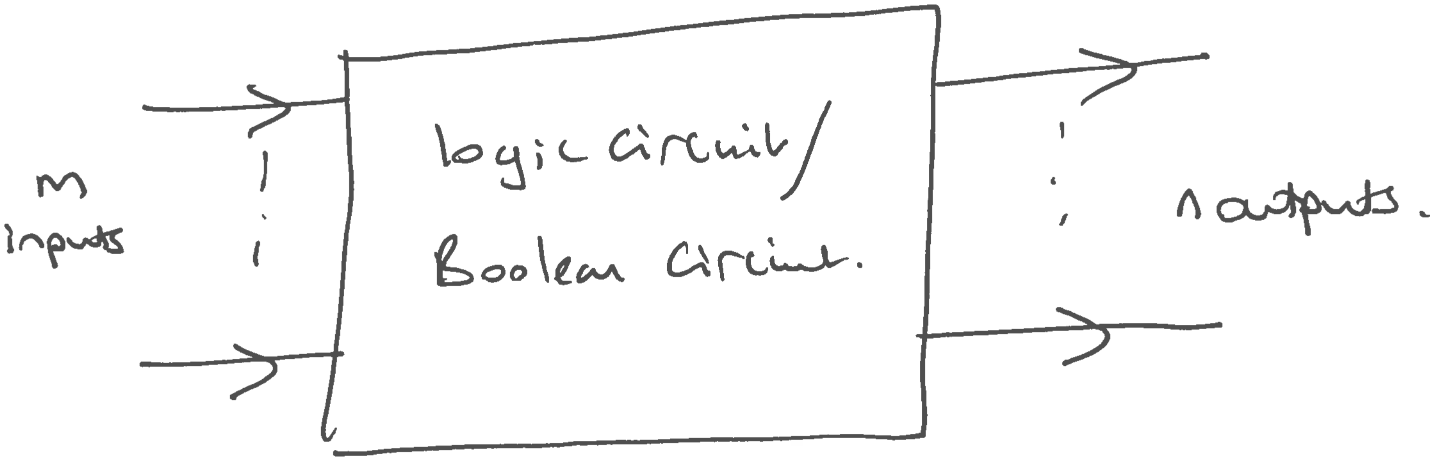
The two examples above suggest that to design switching circuits, we may use Boolean algebra. Shannon showed exactly how to do this. Now switches and later relays have been replaced by transistors. The technology has changed but the maths has stayed the same.

Digital Circuits today are designed with the help of Boolean algebra.

We now explain how in a little more detail, and then work out an extended example.

A Combinatorial circuit is ~~the~~ one that has no internal memory. Those that do are called Sequential and will not be discussed further here.

Boolean algebra can be used directly in the design of such circuits. We may picture a circuit as follows

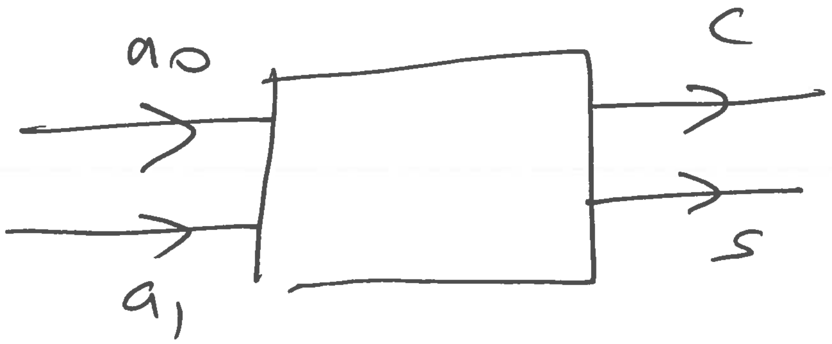


Each input carries either 0 or 1.

Each output carries either 0 or 1.

Here is a concrete example.

Example



input $\qquad\qquad\qquad\qquad$ output

$$(a_1, a_0) \longmapsto (c, s)$$

described by the following input/output table

a_1	a_0	c	s
1	1	1	0
1	0	0	1
0	1	0	1
0	0	0	0

This circuit calculates the sum of any two one-bit binary numbers.

Key idea If we can construct circuits for each output separately, we can then combine the two circuits to obtain a circuit whose input/output behavior is given by the above table. i.e.

a_1	a_0	c
1	1	1
1	0	0
0	1	0
0	0	0

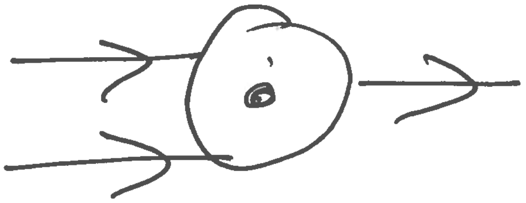
a_1	a_0	s
1	1	0
1	0	1
0	1	1
0	0	0

The general principle is that we design circuits for multiple-input single-output circuits and then combine them.

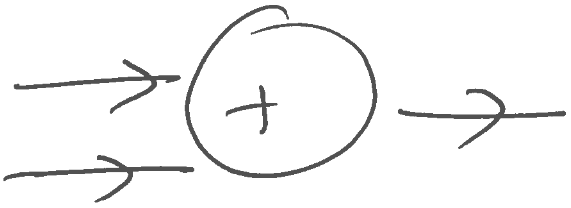
Rather than build a whole circuit 'top-down', we would like to construct one from simpler elements.

A circuit element that implements some Boolean function is called a <u>gate</u>.

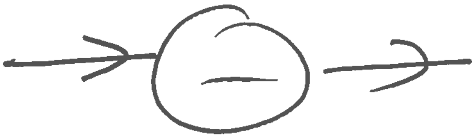
I shall use the following — non-standard — notation.



<u>and-gate</u>



<u>or-gate</u>



<u>inverter</u>

For this to work, it would have to be possible to construct any multiple-input-single-output Boolean function from these gates.

But we have already proved that $\{\neg, \vee, \wedge\}$ is an adequate set of connectives.

Theorem ("Functional Completeness of $\mathbb{B}$").
Every n-input/single output Boolean function can be constructed from inverters, and-gates and or-gates.
[or, of course, nand-gates only or nor-gates only]

We use the same procedure as that used in constructing DNF.

Example

a_1	a_0	s
1	1	0
1	0	1 *
0	1	1 *
0	0	0

DNF

$$\boxed{S = a_1 \cdot \overline{a_0} + \overline{a_1} \cdot a_0}$$

$$\boxed{C = a_1 \cdot a_0}$$

We now convert our Boolean expressions into actual circuits.

(1) Write down parse trees — but for Boolean expressions

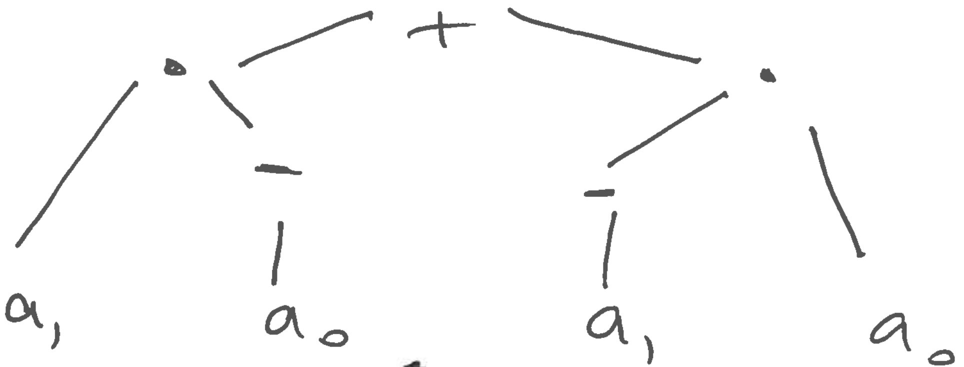
(2) use <u>fanout</u> so that each variable occurs exactly once as input and replace Boolean ops by corresponding gates.

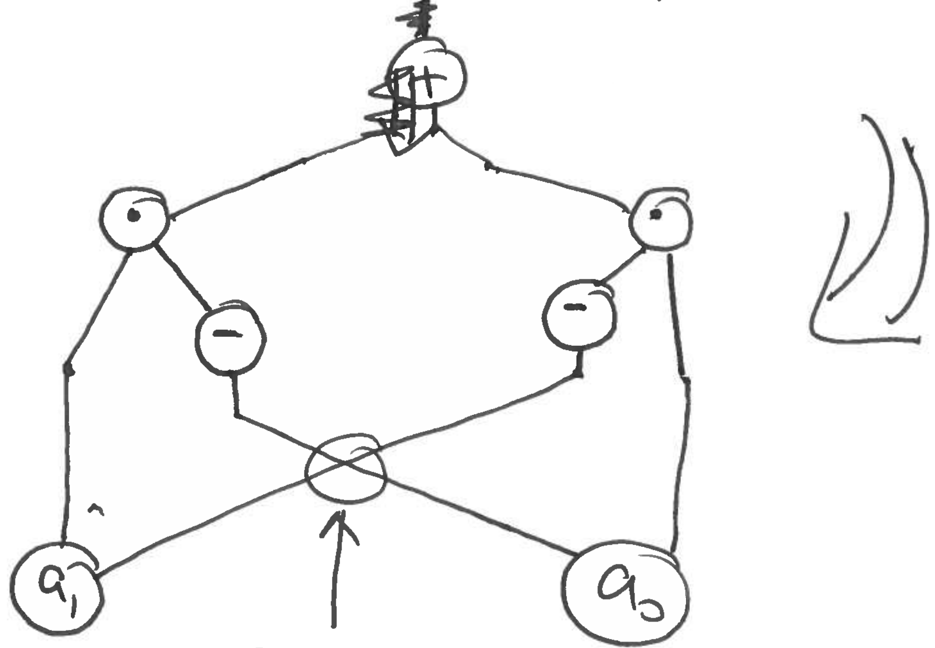
(3) Now match in inputs and outputs?

(4) Simplify and Combine.

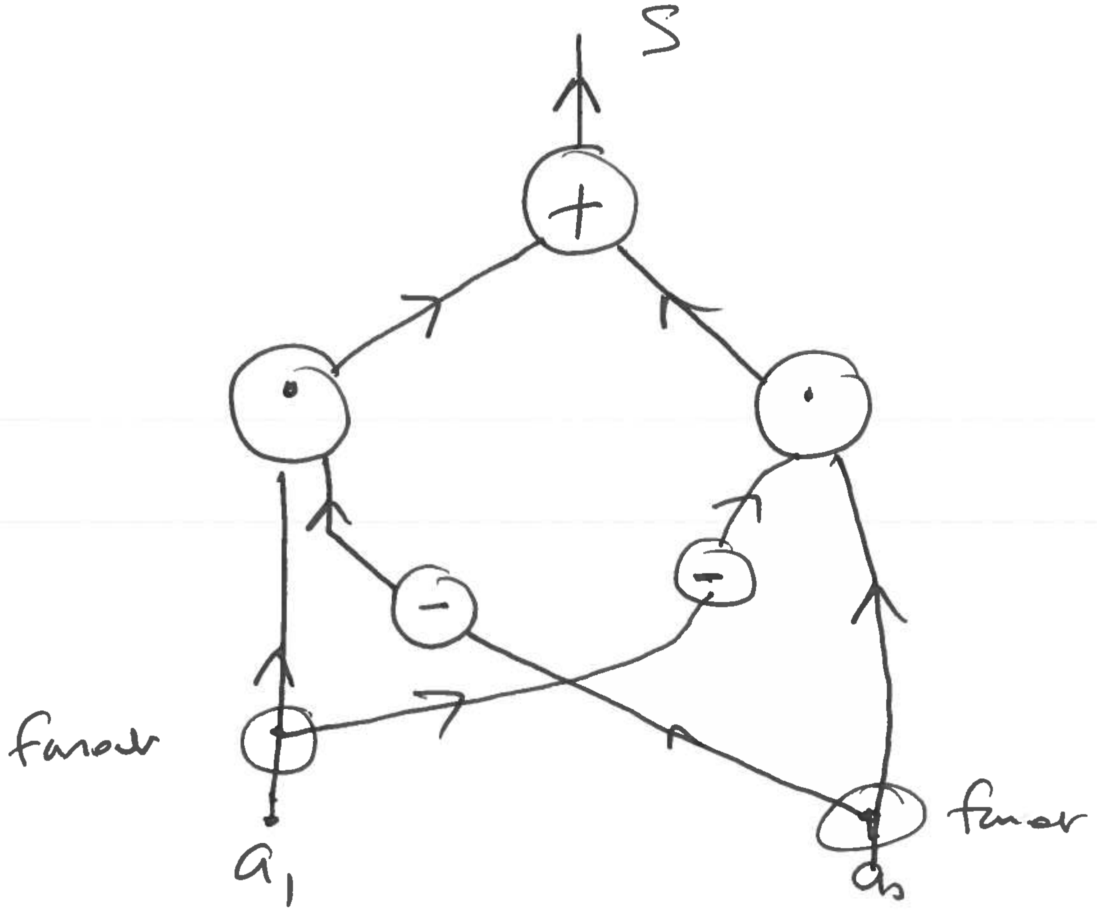
$$S = a_1 \cdot \overline{a_0} + \overline{a_1} \cdot a_0$$

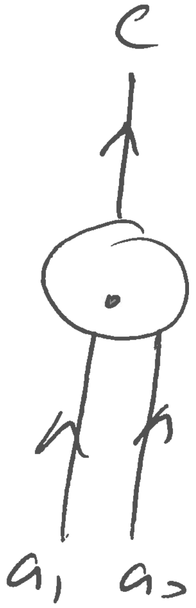
$\Downarrow$ parse tree

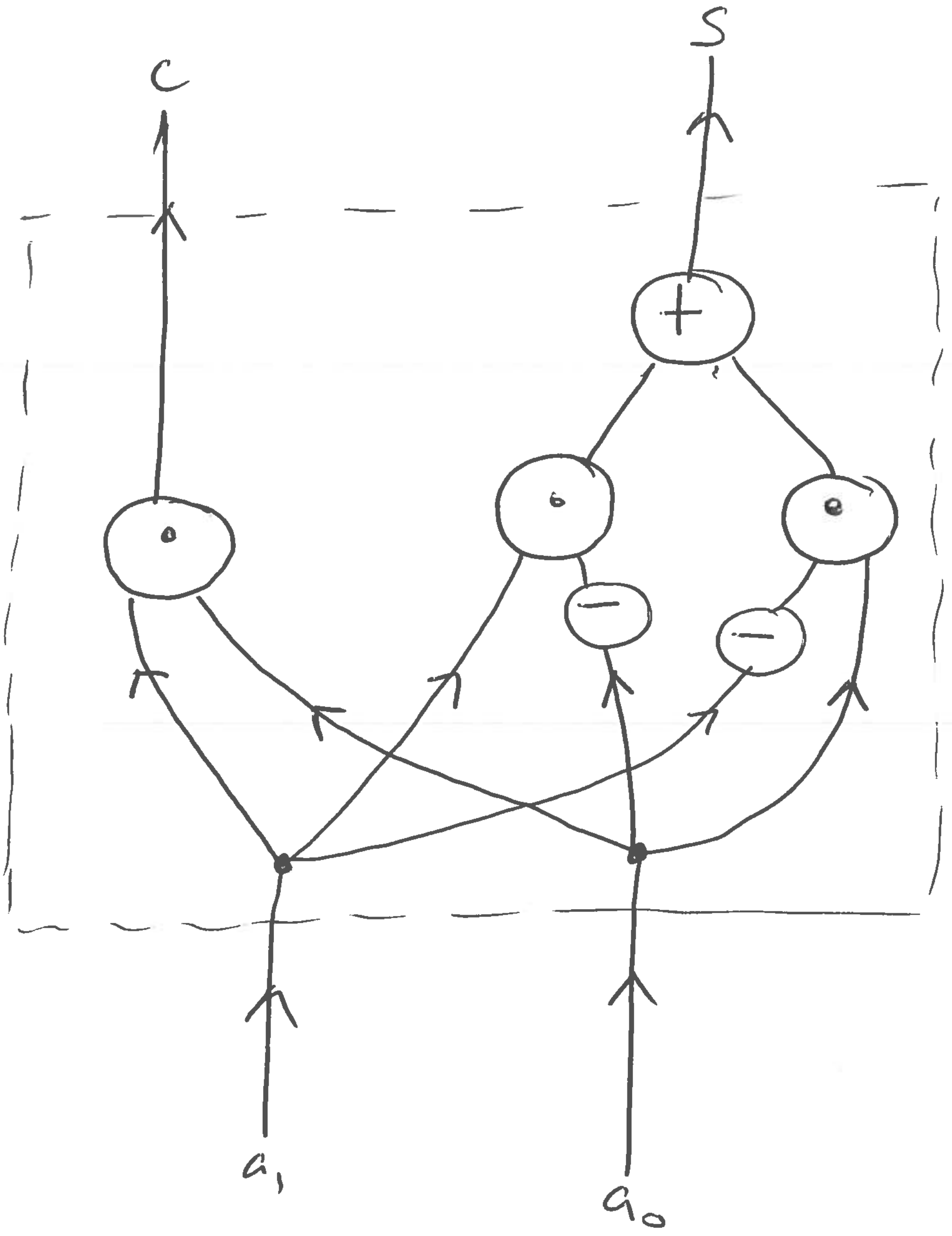




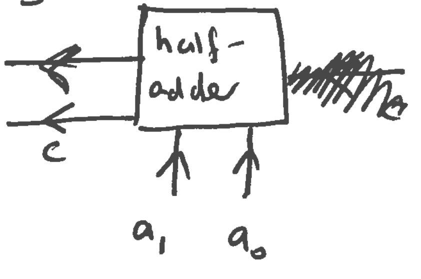
Crossover
not connection


S
+
·
·
−
−
fanout
fanout
a_1
a_2


c
·
a_1
a_2



This is now a circuit that computes our function. This is a <u>half-adder</u>.

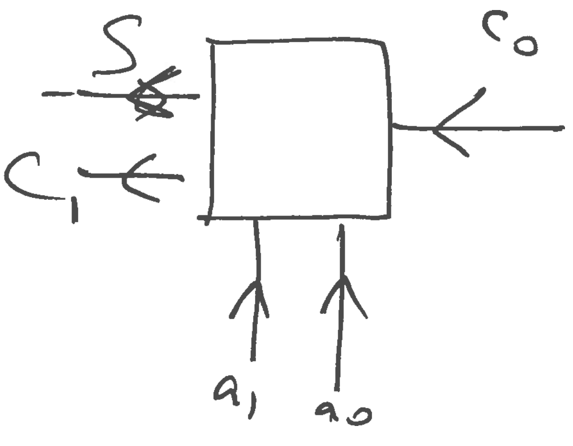


We would like to build a circuit that will add two 4-digit binary numbers together.

The basic building block we shall need is a circuit called a <u>full-adder.</u>

This has three inputs (a_1, a_0, c_0) and two outputs (s, c_1) where

$$s c_1 = a_1 + a_0 + c_0$$ <u>binary arithmetic</u>



A full-adder is a circuit with three inputs (a_1, a_0, c_0) and two outputs (c_1, s) according to the following table

a_1	a_0	c_1	c_1	s
1	1	1	1	1
1	1	0	1	0
1	0	1	1	0
1	0	0	0	1
0	1	1	1	0
0	1	0	0	1
0	0	1	0	1
0	0	0	0	0

Exercise Show how to build a full-adder from two half-adders and an or-gate.

from Mano

but $C = xy$, and therefore we have:

$$S = (C + x'y')'$$

In Fig. 4-2(d) we use the product of sums implementation with C derived as follows:

$$C = xy = (x' + y')'$$

The half-adder can be implemented with an exclusive-OR and an AND gate as shown in Fig. 4-2(e). This form is used later to show that two half-adder circuits are needed to construct a full-adder circuit.

Full-Adder

A full-adder is a combinational circuit that forms the arithmetic sum of three input bits. It consists of three inputs and two outputs. Two of the input variables, denoted by x and y, represent the two significant bits to be added. The third input, z, represents the carry from the previous lower significant position. Two outputs are necessary because the arithmetic sum of three binary digits ranges in value from 0 to 3, and binary 2 or 3 needs two digits. The two outputs are designated by the symbols S for sum and C for carry. The binary variable S gives the value of the least significant bit of the sum. The binary variable C gives the output carry. The truth table of the full-adder is as follows:

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

The eight rows under the input variables designate all possible combinations of 1's and 0's that these variables may have. The 1's and 0's for the output variables are determined from the arithmetic sum of the input bits. When all input bits are 0's, the output is 0. The S output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1. The C output has a carry of 1 if two or three inputs are equal to 1.

The input and output bits of the combinational circuit have different interpretations at various stages of the problem. Physically, the binary signals of the input wires are considered binary digits added arithmetically to form a two-digit sum at

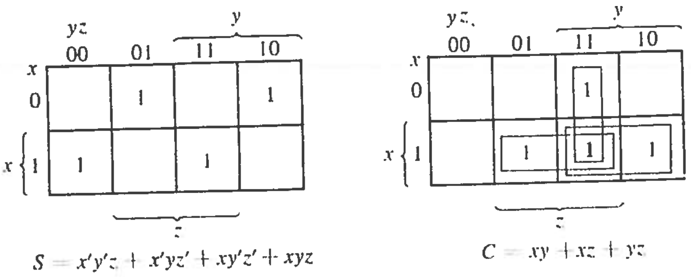


Figure 4-3 Maps for full-adder

the output wires. On the other hand, the same binary values are considered variables of Boolean functions when expressed in the truth table or when the circuit is implemented with logic gates. It is important to realize that two different interpretations are given to the values of the bits encountered in this circuit.

The input-output logical relationship of the full-adder circuit may be expressed in two Boolean functions, one for each output variable. Each output Boolean function requires a unique map for its simplification. Each map must have eight squares, since each output is a function of three input variables. The maps of Fig. 4-3 are used for simplifying the two output functions. The 1's in the squares for the maps of S and C are determined directly from the truth table. The squares with 1's for the S output do not combine in adjacent squares to give a simplified expression in sum of products. The C output can be simplified to a six-literal expression. The logic diagram for the full-adder implemented in sum of products is shown in Fig. 4-4. This implementation uses the following Boolean expressions:

$$S = x'y'z + x'yz' + xy'z' + xyz$$

$$C = xy + xz + yz$$

Other configurations for a full-adder may be developed. The product-of-sums implementation requires the same number of gates as in Fig. 4-4, with the

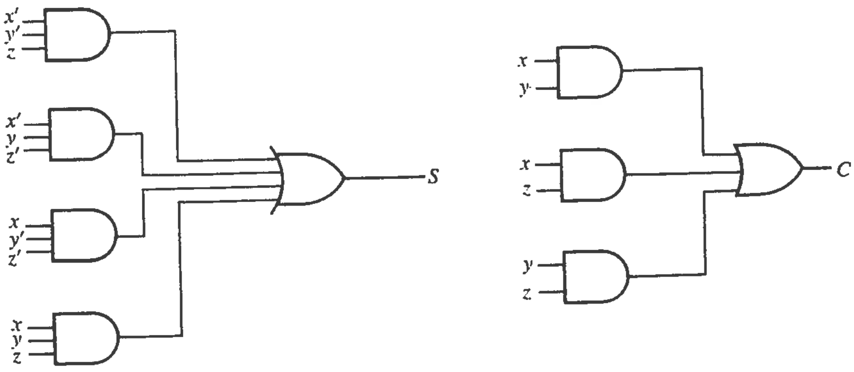


Figure 4-4 Implementation of full-adder in sum of products

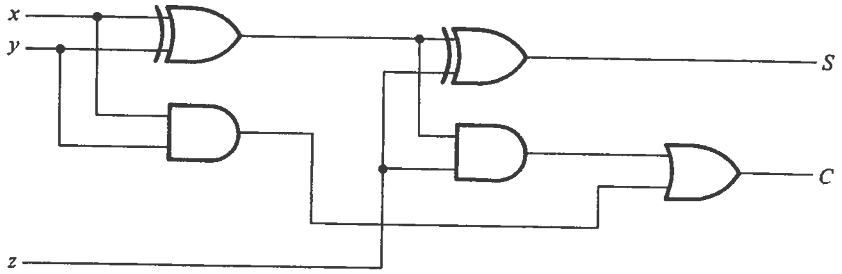


Figure 4-5 Implementation of a full-adder with two half-adders and an OR gate

number of AND and OR gates interchanged. A full-adder can be implemented with two half-adders and one OR gate, as shown in Fig. 4-5. The S output from the second half-adder is the exclusive-OR of z and the output of the first half-adder, giving:

$$\begin{aligned} S &= z \oplus (x \oplus y) \\ &= z'(xy' + x'y) + z(xy' + x'y)' \\ &= z'(xy' + x'y) + z(xy + x'y') \\ &= xy'z' + x'yz' + xyz + x'y'z \end{aligned}$$

and the carry output is:

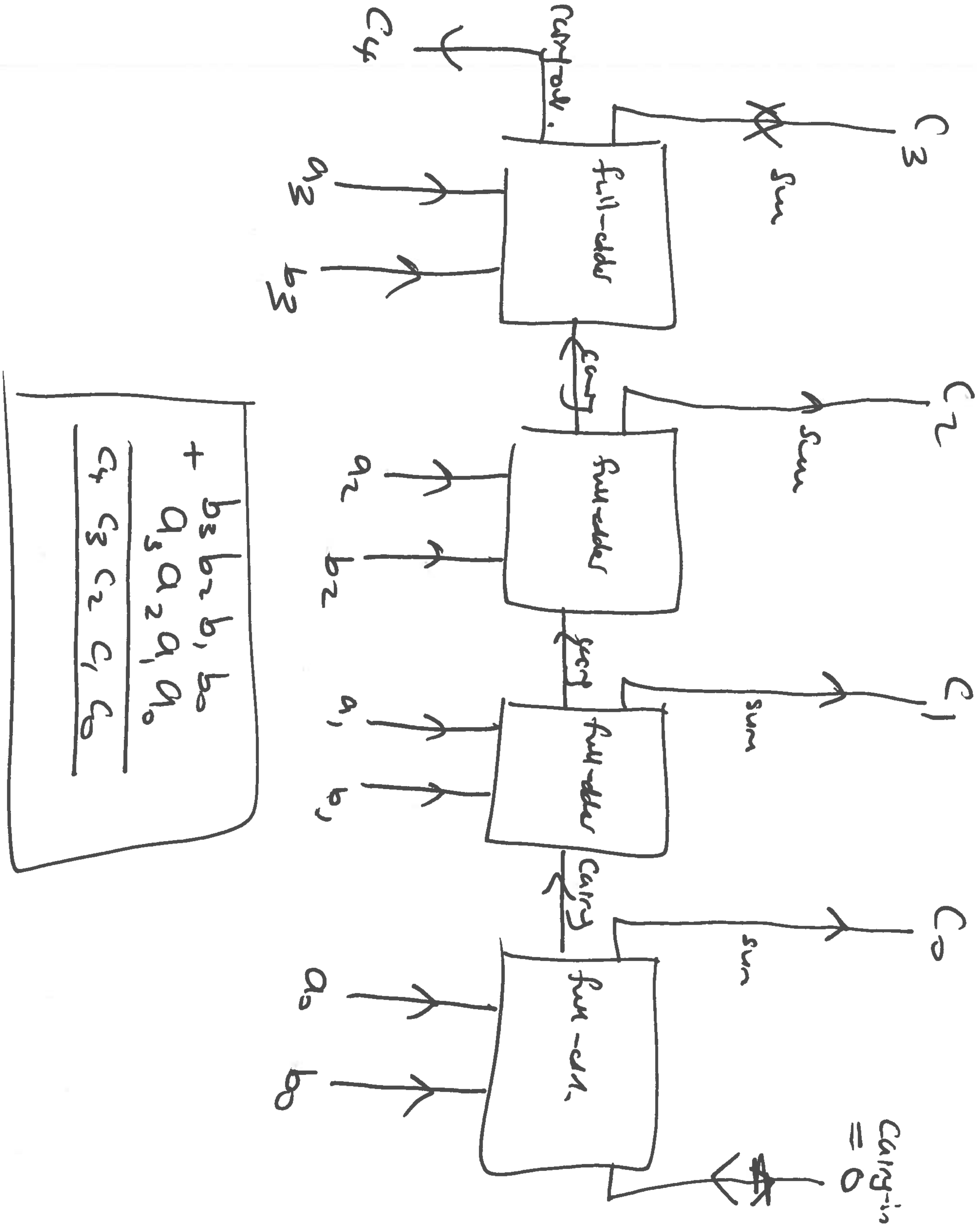
$$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$$

4-4 SUBTRACTORS

The subtraction of two binary numbers may be accomplished by taking the complement of the subtrahend and adding it to the minuend (Section 1-5). By this method, the subtraction operation becomes an addition operation requiring full-adders for its machine implementation. It is possible to implement subtraction with logic circuits in a direct manner, as done with paper and pencil. By this method, each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a difference bit. If the minuend bit is smaller than the subtrahend bit, a 1 is borrowed from the next significant position. The fact that a 1 has been borrowed must be conveyed to the next higher pair of bits by means of a binary signal coming out (output) of a given stage and going into (input) the next higher stage. Just as there are half- and full-adders, there are half- and full-subtractors.

Half-Subtractor

A half-subtractor is a combinational circuit that subtracts two bits and produces their difference. It also has an output to specify if a 1 has been borrowed. Designate the minuend bit by x and the subtrahend bit by y. To perform $x - y$, We may now construct a circuit that adds two 4-~~bi~~ bit binary nos.



A Symbolic Analysis of Relay and Switching Circuits

From Wikipedia, the free encyclopedia

A Symbolic Analysis of Relay and Switching Circuits is the title of a master's thesis written by computer science pioneer Claude E. Shannon while attending the Massachusetts Institute of Technology (MIT) in 1937. In his thesis, Shannon proved that Boolean algebra[1] could be used to simplify the arrangement of the relays that were the building blocks of the electromechanical automatic telephone exchanges of the day. Shannon went on to prove that it should also be possible to use arrangements of relays to solve Boolean algebra problems.

The utilization of the binary properties of electrical switches to perform logic functions is the basic concept that underlies all electronic digital computer designs. Shannon's thesis became the foundation of practical digital circuit design when it became widely known among the electrical engineering community during and after World War II. At the time, the methods employed to design logic circuits were *ad hoc* in nature and lacked the theoretical discipline that Shannon's paper supplied to later projects.

Professor Howard Gardner, of Harvard University, described Shannon's thesis as "possibly the most important, and also the most famous, master's thesis of the century".[2] A version of the paper was published in the 1938 issue of the *Transactions of the American Institute of Electrical Engineers*,[3] and in 1940, it earned Shannon the Alfred Noble American Institute of American Engineers Award.

References

1. ^ Caldwell, Samuel H. (1965, Sixth Printing) [1958]. *Switching Circuits and Logical Design*. New York: John Wiley & Sons. p. 34. "[Shannon] constructed a calculus based on a set of postulates which described basic switching ideas; e.g., an open circuit in series with an open circuit is an open circuit. Then he showed that his calculus was equivalent to certain elementary parts of the calculus of propositions, which in turn was derived from the algebra of logic developed by George Boole."
2. ^ Gardner, Howard (1987). *The Mind's New Science: A History of the Cognitive Revolution*. Basic Books. p. 144. ISBN 0-465-04635-5.
3. ^ Shannon, C. E. (1938). "A Symbolic Analysis of Relay and Switching Circuits". *Trans. AIEE* **57**: 713–723.

External links

- Full text at MIT (http://hdl.handle.net/1721.1/11173)

Retrieved from "http://en.wikipedia.org
/w/index.php?title=A_Symbolic_Analysis_of_Relay_and_Switching_Circuits&
oldid=507790670"

Categories: Computer science papers | Information theory | Applied mathematics

A SYMBOLIC ANALYSIS
OF
RELAY AND SWITCHING CIRCUITS

by

Claude Elwood Shannon

B.S., University of Michigan

1936

Submitted in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
from the
Massachusetts Institute of Technology
1940

Signature of Author______________________________

Department of Electrical Engineering, August 10, 1937

Signature of Professor
in Charge of Research______________________________

Signature of Chairman of Department
Committee on Graduate Students______________________

TABLE OF CONTENTS

ACKNOWLEDGMENT

The author is indebted to Professor F. L. Hitchcock, who supervised the thesis, for helpful criticism and advice.

I Introduction: Types of Problems

In the control and protective circuits of complex electrical systems it is frequently necessary to make intricate interconnections of relay contacts and switches. Examples of these circuits occur in automatic telephone exchanges, industrial motor control equipment and in almost any circuits designed to perform complex operations automatically. Two problems that occur in connection with such networks of switches will be treated here. The first, which will be called analysis, is to determine the operating characteristics of a given circuit. It is, of course, always possible to analyze any given circuit by setting up all possible sets of initial conditions (positions of switches and relays) and following through the chain of events so instigated. This method is, however, very tedious and open to frequent error.

The second problem is that of synthesis. Given certain characteristics, it is required to find a circuit incorporating these characteristics. The solution of this type of problem is not unique and it is therefore additionally desirable that the circuit requiring the least number of switch blades and relay contacts be found. Although a solution can usually be obtained by a "cut and try" method, first satisfying one requirement and then making additions until all are satisfied, the circuit so obtained will seldom be the simplest. This method also has the disadvantages of being long, and the resulting design often contains hidden "sneak circuits."

The method of solution of these problems which will be developed here may be described briefly as follows: Any circuit is represented by a set of equations, the terms of the equations representing the various relays and switches of the circuit. A calculus is developed for manipulating these equations by simple mathematical processes, most of which are similar to ordinary algebraic algorisms. This calculus is shown to be exactly analogous to the Calculus of Propositions used in the symbolic study of logic. For the synthesis problem the desired characteristics are first written as a system of equations, and the equations are then manipulated into the form representing the simplest circuit. The circuit may then be immediately drawn from the equations. By this method it is always possible to find the simplest circuit containing only series and parallel connections, and for certain types of functions it is possible to find the simplest circuit containing any type of connection. In the analysis problem the equations representing the given circuit are written and may then be interpreted in terms of the operating characteristics of the circuit. It is also possible with the calculus to obtain any number of circuits equivalent to a given circuit.

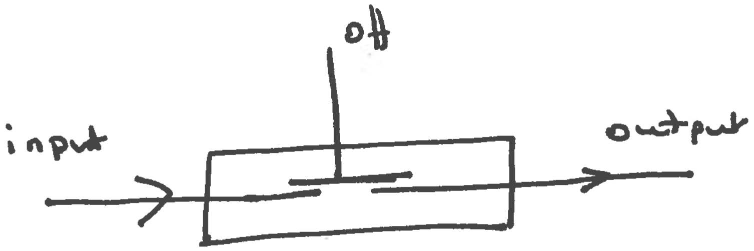
Phraseology will be borrowed from ordinary network theory for concepts in switching circuits that are roughly analogous to those of impedence networks.

3·5 Transistors

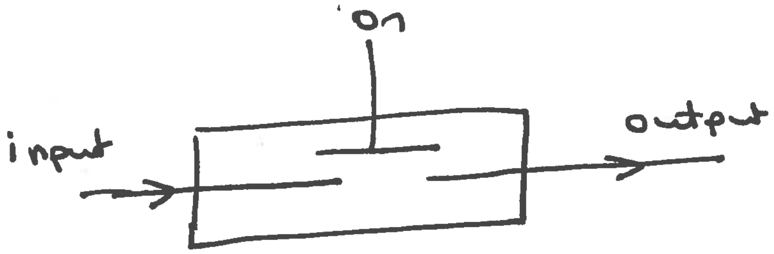
We shall now show that every combinatorial Boolean circuit can be constructed from transistors.

Electronic switches are called transistors.

Schematically, they work as follows

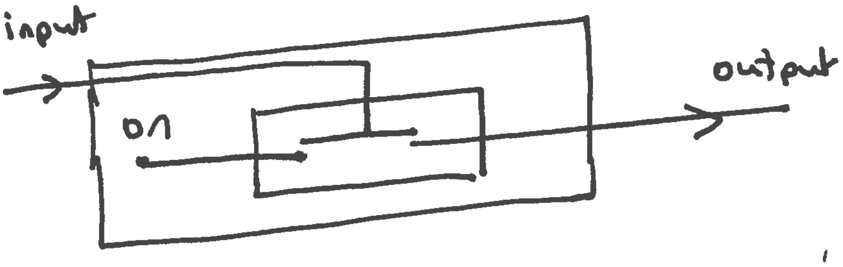


Current flows



current off

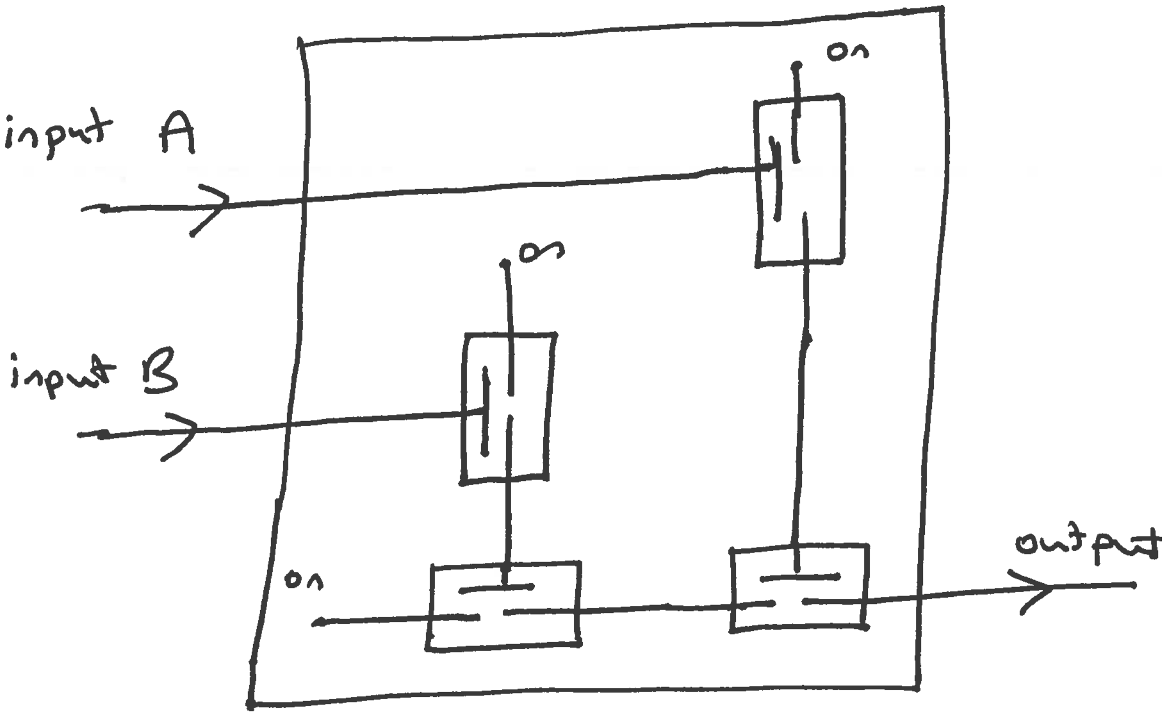
Example 1



input	output
on	off
off	on

This is the Boolean operation of complementation

Example 2



A	B	output
on	on	on
on	off	off
off	on	off
off	off	off

This is the Boolean
operation .

"Theorem" Every Combinatorial Circuit can be
Constructed from transistors .