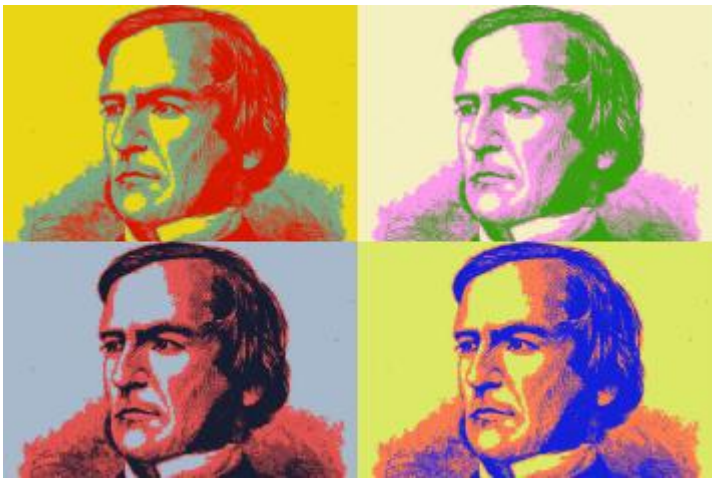
A FIRST COURSE IN LOGIC

Mark V. Lawson

October 2017



Contents

Preface

When you come to a fork in the road, take it! — Yogi Berra.

Background. These notes were written to accompany my Heriot-Watt University course *F17LP Logic and proof* designed and written in 2011. The course was in fact instigated by my colleagues in Computer Science and was therefore intended originally for first year computer science students, but the course was subsequently also offered as an option to second year mathematics students.

Coverage. In writing this course, I was particularly influenced, like many others, by Smullyan's[1] book [40]. Chapters 1 and 3 of these notes cover roughly the same material as the first 65 pages of [40] together with part of Chapter XI but I have incorporated ideas to be found in many other books and articles as well; see the bibliography for a complete list of references. Let me stress that this is very much a *first* introduction to logic. I have therefore tried to assume as few prerequisites as possible. In particular, any mathematics needed is introduced only when necessary. The real mathematical prerequisite is an ability to manipulate symbols: in other words, basic algebra. Anyone who can write programs should have this ability already. In addition, I have not tried to follow every by-way of logic. My goal was to inspire interest and curiosity about this subject and lay the foundations for further study.

Aims. This is an introduction to first order logic suitable for first or second year mathematicians and computer scientists. There are three components to this course: propositional logic, Boolean algebras and first-order logic. Logic

[1] Raymond Smullyan (1919–2017) was the doyen of logic and his work did much to bring it to a wider audience.

is the basis of proofs in mathematics — how do we know what we say is true? — and also of computer science — how do I know this program will do what I think it will do? Propositional logic deals with proofs that can be analysed in terms of the words *and, or, not, implies* — it is the logic of elementary decision making — whereas first-order logic extends this to encompass the use of the words *there exists* and *for all*. Boolean algebra, on the other hand, is an algebraic system that arises naturally from propositional logic and is the basic mathematical tool used in circuit design in computers.

Acknowledgements. My thanks to Phil Scott (Ottawa) for reading and commenting on an early draft. Thanks also to the following philanthropists: Till Tantu for Ti*k*Z, Sam Buss for bussproofs, Jeffey Mark Siskind and Alexis Dimitriadis for qtree, and Alain Matthes for tkz-graph. The following students spotted typos: Muhammad Hamza, Andrea Lin. Finally, 'From my grandfather Verus, nobility of character and eveness of temper' [2].

Corrections. These are first generation typed notes so I have no doubt that errors have crept in. If you spot them, please email me.

Starred sections. Those sections marked with asterisks will not be taught in 2017 and are non-examinable.

Introduction

> **Logic**, *n. The art of thinking and reasoning in strict accordance with the limitations and incapacities of the human misunderstanding.* — Ambrose Bierce.

Logic is the study of how we reason and, as such, is a subject of interest to philosophers, mathematicians and computer scientists. This course is aimed at mathematicians and computer scientists (but philosphers are welcome as well.) The word 'logic' itself is derived from the Greek word 'logos' one of whose myriad meanings is 'reason'.[2] Human beings used reason, when they chose to, thousands of years before anyone thought to study *how* we reason. The formal study of logic only began in classical Greece with the influential work of the philosopher Aristotle and a group of philosophers known as the Stoics, but the developments significant to this course are more recent beginning in the nineteenth century.[3]

Two names that stand out as the progenitors of modern logic are George Boole (1815–1864) and Gottlieb Frege (1848–1925). Their work, as well as that of others — see the graphic novel [10], for example, or the far weightier tome [24] — led in 1928 to the first textbook on mathematical logic *Grundzüge der theoretischen Logik* [21] by David Hilbert (1862–1943) and Wilhelm Ackermann (1896–1962). This served not only as the template for

[2]The online version of Liddell and Scott, a famous dictionary of classical Greek, to be found at `http://stephanus.tlg.uci.edu/lsj/#eid=65855&context=lsj&action=from-search`, contains two pages of meanings of the word 'logos' in very small type. The same word, with another of its meanings — 'word' — crops up in the first line of St John's Gospel.

[3]The concept of 'logic' as a discipline was largely confined to academic circles but in the UK at least this all changed on Saturday 12th July 1969 when the BBC began broadcasting *Star Trek* with the episode 'Where no man has gone before' which contained the character of Mr Spock, half-human and half-vulcan, who was guided by a philosophy inspired by logic. And yes, gentle reader, I was there.

all subsequent introductions to mathematical logic, it also posed a question always referred to in its original German: the *Entscheidungsproblem* which simply means the *decision problem*. The solution of the *Entscheidungsproblem* by Alan Turing (1912–1954) can, for once, be described without hyperbole as revolutionary.[4]

Turing was one of the most influential mathematicians of the twentieth century. He is often claimed as the father of computer science and this claim rests mainly on the paper he wrote in 1937 [47] with the multi-lingual title *On computable numbers, with an application to the Entscheidungsproblem*. In this paper, Turing describes what we now call in his honour a *universal Turing machine*, a mathematical blueprint for building a computer independent of technology.[5] Remarkably, Turing showed in his paper that there were problems that cannot be solved by computer — not because the computers weren't powerful enough but because they were intrinsically unable to solve them. Thus the limitations of computers were known before the first one had ever been built. But Turing didn't set out to invent computers, rather he set out to solve the *Entscheidungsproblem*. Roughly speaking, this asks whether it is possible to write a program that will answer all mathematical questions. Turing proved that this was impossible. The *Entscheidungsproblem* is, in fact, a question about logic and it is my hope that by the end of this course, you will understand what is meant by this problem and its significance in logic.

Introductory Exercises

The exercises below do not require any prior knowledge but introduce ideas that are important in this course

---

1. Here are two puzzles by Raymond Smullyan. On an island there are two kinds of people: *knights* who always tell the truth and *knaves* who always lie. They are indistinguishable.

[4] The first solution to this problem was by Alonzo Church using a system he introduced called lambda calculus. But Turing's solution is the more intuitively compelling. Interestingly, whereas Turing's work had an influence on hardware, Church's had an influence on software.

[5] You can access copies of this paper via the links to be found on the Wikipedia article on the *Entscheidungsproblem*.

(a) You meet three such inhabitants A, B and C. You ask A whether he is a knight or knave. He replies so softly that you cannot make out what he said. You ask B what A said and they say 'he said he is a knave'. At which point C interjects and says 'that's a lie!'. Was C a knight or a knave?

(b) You encounter three inhabitants: A, B and C.
A says 'exactly one of us is a knave'.
B says 'exactly two of us are knaves'.
C says: 'all of us are knaves'.
What type is each?

2. This question is a variation of one that has appeared in the puzzle sections of many magazines. There are five houses, from left to right, each of which is painted a different colour, their inhabitants are called Sarah, Charles, Tina, Sam and Mary, but not necessarily in that order, who own different pets, drink different drinks and drive different cars.

(a) Sarah lives in the red house.

(b) Charles owns the dog.

(c) Coffee is drunk in the green house.

(d) Tina drinks tea.

(e) The green house is immediately to the right (that is: your right) of the white house.

(f) The Oldsmobile driver owns snails.

(g) The Bentley owner lives in the yellow house.

(h) Milk is drunk in the middle house.

(i) Sam lives in the first house.

(j) The person who drives the Chevy lives in the house next to the person with the fox.

(k) The Bentley owner lives in a house next to the house where the horse is kept.

(l) The Lotus owner drinks orange juice.

(m) Mary drives the Porsche.

(n) Sam lives next to the blue house.

There are two questions: who drinks water and who owns the aardvark?

3. *Bulls and Cows* is a code-breaking game for two players: the code-setter and the code-breaker. The code-setter writes down a 4-digit secret number all of whose digits must be different. The code-breaker tries to guess this number. For each guess they make, the code-setter scores their answer: for each digit in the right position score 1 bull (1B), for each correct digit in the wrong position score 1 cow (1C); no digit is scored twice. The goal is to guess the secret number in the smallest number of guesses. For example, if the secret number is 4271 and I guess 1234 then my score will be 1B,2C. Here's an easy problem. The following is a table of guesses and scores. What are the possibilities for the secret number?

1389	$0B, 0C$
1234	$0B, 2C$
1759	$1B, 1C$
1785	$2B, 0C$

4. *Hofstadter's MU-puzzle.* A *string* is just an ordered sequence of symbols. In this puzzle, you will construct strings using the letters M, I, U where each letter can be used any number of times, or not at all. You are given the string MI which is your only input. You can make new strings only by using the following rules any number of times in succession in any order:

   (I) If you have a string that ends in I then you can add a U on at the end.

   (II) If you have a string Mx where x is a string then you may form Mxx.

   (III) If III occurs in a string then you may make a new string with III replaced by U.

   (IV) If UU occurs in a string then you may erase it.

   I shall write $x \to y$ to mean that y is the string obtained from the string x by applying one of the above four rules. Here are some examples:

   - By rule (I), $MI \to MIU$.
   - By rule (II), $MIU \to MIUIU$.

- By rule (III), $UMIIIMU \to UMUMU$.
- By rule (IV), $MUUUII \to MUII$.

The question is: can you make MU?

5. *Sudoku puzzles* have become very popular in recent years. The newspaper that first launched them in the UK went to great pains to explain that they had nothing to do with mathematics despite involving numbers. Instead, they said, they were logic problems. This of course is nonsense: logic is part of mathematics. What they should have said is that they had nothing to do with *arithmetic*. The goal is to insert digits in the boxes to satisfy two conditions: first, each row and each column must contain all the digits from 1 to 9 exactly once, and second, each 3×3 box must contain the digits 1 to 9 exactly once.

	1		4	2				5
		2		7	1		3	9
							4	
2		7	1					6
				4				
6					7	4		3
	7							
1	2		7	3		5		
3				8	2		7	

6. Consider the following algorithm.[6] The input is a positive whole number $n \geq 2$; so $n = 2, 3, 4, \ldots$ If n is even, divide it by 2 to get $\frac{n}{2}$; if n is odd, multiply it by 3 and add 1 to get $3n + 1$. Now repeat this process and only stop if you reach 1. For example, if $n = 6$ we get successively $6, 3, 10, 5, 16, 8, 4, 2, 1$ and the algorithm stops at 1. What happens if $n = 11$? What about $n = 27$? Is it true that whatever whole number you input this procedure always yields 1?

[6] An *algorithm* is a method for solving a problem that requires no ingenuity or thought to apply. Every program is an implementation of an algorithm. We shall say more about algorithms in Chapter 3.

Chapter 1

Propositional logic

1. The world is all that is the case. — Ludwig Wittgenstein.

The main goal of this course is to introduce *first-order logic* (FOL).[1] But ever since Hilbert and Ackermann's book [21], this has almost always been done by first describing the simpler logic called *propositional logic* (PL), which is the subject of this chapter, and then upgrading to full (FOL), which we do in Chapter 3. The logics we discuss are examples of *artificial languages* to be contrasted with *natural languages* such as Welsh. Natural languages are often imprecise when we try to use them in situations where precision is essential, being riddled with ambiguities.[2] In such cases, artificial languages are used. For example, programming languages are artificial languages suitable for describing algorithms to be implemented by computer. The description of an artificial language has two aspects: *syntax* and *semantics*. Syntax, or grammar, tells you what the allowable sequences of symbols are, whereas semantics tells you what they mean.

1.1 Informal propositional logic

Our goal is to construct a precise, unambiguous language that will enable us to calculate what is true or what is false. We begin by analysing everyday language.

Language consists of sentences but not all sentences will be grist to our mill. Here are some examples.

[1] Also called *predicate logic*. But this abbreviates to PL which is why I am not using it.

[2] The word 'cleave', for example, means both stick together and split apart.

1. Homer Simpson is Prime Minister.
2. The earth orbits the sun.
3. To be or not to be?
4. Out damned spot!

Sentences (1) and (2) are different from sentences (3) and (4) in that we can say of sentences (1) and (2) whether they are true or false — in this case, (1) is false[3] and (2) is true — whereas for sentences (3) and (4) it is meaningless to ask whether they are true or false, since (3) is a question and (4) is an exclamation.

> A sentence that is capable of being either true or false (though we might not know which) is called a *statement*.[4]

In mathematics and computer science, it is enough to study only statements (although if we did that in everyday life we would come across as robotic). Statements come in all shapes and sizes from the banal 'the sky is blue' to the informative 'the battle of Hastings was in 1066'. But in our work, the only thing that interests us about a statement is whether it is *true (T)* or *false (F)*. In other words, what its *truth value* is and nothing else; the phrase 'and nothing else' is important. We can now give some idea as to what the subject of logic is about.

> Logic is about deducing whether a given statement is true or false on the basis of information provided by some (other) collection of statements.

I shall say more about this later.

Some statements can be analysed into combinations of simpler statements using special kinds of words called *connectives*. The connectives that we shall be interested in are **not**, **and**, **or**, **iff** (this is not a typo) and **implies** and are described below. I have written them in bold to indicate that they will

[3] Except in satirical contexts.

[4] Or, a *proposition*. This is why 'propositional logic' is so called. But as a mathematician, I use the word 'proposition' to mean a theorem of lesser importance. I was therefore uncomfortable using it in the sense needed in this course. As Walt Whitman wrote "Do I contradict myself? Very well, then I contradict myself, I am large, I contain multitudes." But then one doesn't go to poets for logic.

sometimes be used in unusual ways. I should add that these are not the only connectives. In fact, there are in fact infinitely many of them. But in a sense we shall make precise in Section 1.6, the ones above are sufficient.

not

Let p be the statement *It is not raining*. This is related to the statement q given by *It is raining*. We know that the truth value of q will be the opposite of the truth value of p. This is because p is the *negation* of q. This is precisely described by the following *truth table*.

It is raining	It is not raining
T	F
F	T

To avoid peculiarities of English grammar, we replace the word 'not' in the first instance by the slighly less natural phrase 'It is not the case that'. Thus *It is not the case that it is raining* means the same thing as *It is not raining*.[5] We go one step further and abbreviate the phrase 'It is not the case that' by **not**. Thus if we denote a statement by p then its negation is **not** p. The above table becomes

p	**not** p
T	F
F	T

This summarizes the behaviour of negation for any statement p. What happens if we negate twice? Then we simply apply the above table twice.

p	**not not** p
T	T
F	F

Thus *It is not the case that it is not raining* should mean the same as *It is raining*. I know this sounds weird and it would certainly be an odd thing to say as anything other than a joke but we are building here a language suitable for mathematics and computer science rather than everyday usage. The word **not** is our first example of a *logical* or *propositional connective*. It

[5] If you used that phrase in everyday language you would sound odd, possibly like a lawyer.

is also what is called a *unary* connective because it is only applied to one input. The remaining connectives are called *binary* because they are applied to two inputs.

and

Under what circumstances is the statement *It is raining and it is cold* true? Well, I had better be both wet and cold. However, in everyday English the word 'and' often means 'and then'. The statement *I got up and had a shower* does not mean the same as *I had a shower and got up*. The latter sentence might be a joke: perhaps my roof leaked when I was asleep. Our goal is to eradicate the ambiguities of everyday language so we cannot allow these two meanings to coexist.[6] Therefore in logic only the first meaning is the one we use. To make it clear that I am using the word 'and' in a special sense, I shall write it in bold: like this **and**. Given two statements p and q, we can describe the truth values of the *compound statement* p **and** q in terms of the truth values assumed by p and q by means of a *truth table*.

p	q	p **and** q
T	T	T
T	F	F
F	T	F
F	F	F

This table tells us that the statement *Homer Simpson is prime minister* **and** *the earth orbits the sun* is false. In everyday life, we might struggle to know how to interpret such a sentence — if someone turned to you on the bus and said it, I think your response would be alarm rather than to register that it was false. Let me underline that the *only* meaning we attribute to the word **and** is the one described by the above truth table. Thus contrary to everyday life the statements *I got up* **and** *I had a shower* and *I had a shower* **and** *I got up* mean the same thing.

or and **xor**

The word *or* in English is more troublesome. Imagine the following set-up. You have built a voice-activated robot that can recognize shapes and

[6] Jokes are difficult to formulate in logic.

colours. It is placed in front of a white square, a black square and a black circle. You tell the robot to choose a black shape or a square. It chooses the black square. Is that good or bad? The problem is that the word *or* can mean *inclusive or*, in which case the choice is good, or it can mean *exclusive or*, in which case the choice is bad. Both meanings are useful so rather than choose one over the other we use two different words to cover the two different meanings. This is an example of *disambiguation*. We use the word **or** to mean *inclusive or*.

p	q	p **or** q
T	T	T
T	F	T
F	T	T
F	F	F

Thus p **or** q is true when *at least one* of p and q is true. We use the word **xor** to mean *exclusive or*.

p	q	p **xor** q
T	T	F
T	F	T
F	T	T
F	F	F

Thus p **xor** q is true when *exactly one* of p and q is true. Although we haven't got far into our study of logic, this is already a valuable exercise. If you use the word *or* you should know what you really mean.

iff

Our next propositional connective will be familiar to mathematics students but less so to computer science students. This is the connective *is equivalent to* also expressed as *if and only if* that we write as **iff** which is an abbreviation and not a spelling mistake and different from the word 'if'. Its meaning is simple.

p	q	p **iff** q
T	T	T
T	F	F
F	T	F
F	F	T

Observe that **not**(p **iff** q) means the same thing as p **xor** q. This is our first result in logic. By the way, I have added brackets to the first statement to make it clear that we are negating the whole statement p **iff** q and not merely p. We use brackets a lot in logic to clarify our meaning. They are not mere decoration.

implies

Our final connective is the problematical one: *if* p *then* q which we shall write as p **implies** q. Much has been written about this connective — its definition may even go back to the Stoics as mentioned in the Introduction — because it unfailingly causes problems. In order to understand it, you have to remember that binary connectives simply join statements together and the statements can be anything at all. In particular, there need be no connection whatsoever between them. In everyday language, this is unnatural because when we learn a language we also absorb rules that belong to a subject called *pragmatics*. Getting these rules wrong, however perfect your pronunciation or flawless your grammer, can be the source of humour or conflict.[7] Clearly, we cannot write these complex social rules into our formal language quite apart from the fact that in mathematics and computer science we are simply not interested in statements that have social nuance. Thus, whatever meaning we attribute to *if* p *then* q it must hold for *any* choices of p and q. In addition, we have to be able to say what the truth value of the compound statement *if* p *then* q is in terms of the truth values of p and q. To figure out what the truth table for this connective should be, we consider it from two different points of view.

1. **Implies** *as a 'contract' or 'promise'.* Suppose your parents say the following 'If you pass your driving test we shall buy you an E-type Jag'. If you do indeed pass your driving test and they do indeed buy you an E-type Jag then they will have fulfilled their promise. On the other hand, if you pass your driving test and they don't buy you an E-type Jag then you would legitimately complain that they had broken their promise. Both of these cases are easy. Suppose, however, you don't pass your driving test. If they don't buy you an E-type Jag, you cannot complain since they certainly haven't broken their promise. If they do, nevertheless, buy you an E-type

[7] As an example, consider the following situation. Parent says to teenager: what time do you call this? Teenager replies: midnight.

Jag, then you would be delighted since it would be completely unexpected. But, again, they wouldn't be breaking the terms of their promise (they just weren't telling you the whole story). In summary, the promise is kept in all situations except where you pass your test and your parents don't buy you that E-type Jag.

2. **Implies** *has to fit in with the other connectives.* Suppose I tell you that the truth value of the statement p is T and that the truth value of the statement *if* p *then* q is also T. What can we say about the truth value of q? Intuitively, we would say that q also has the truth value T. This enables us to complete the first row of the truth table below

p	q	p **implies** q
T	T	T
T	F	x
F	T	y
F	F	z

where x, y and z are truth values we have yet to determine. Assume that p is T and q is F. If the truth value for $p \to q$ were T then by row one, we would have to deduce that q was T, as well. This conflicts with our assumption that q is F. It follows that x must be F.

Next, p **iff** q should mean the same thing as (p **implies** q) **and** (q **implies** p). A quick calculation shows that this forces z to be T. We have therefore filled in three rows of the truth table below.

p	q	p **implies** q
T	T	T
T	F	F
F	T	y
F	F	T

Finally, we are left with having to decide whether y is T or F. If we chose F then we would have a truth table identical to that of **iff**. But **implies** should be different from **iff** thus we are forced to put $y = T$. This gives us

the following truth table for **implies**.

p	q	p **implies** q
T	T	T
T	F	F
F	T	T
F	F	T

Impeccable though the above analysis is, it does have some seemingly bizarre consequences. For example, the statement *Homer Simpson is Prime Minister* **implies** *the sun orbits the earth* is in fact true. This sort of thing can be offputting when first encountered and can seem to undermine what we are trying to achieve. But remember: we are using everyday words in very special ways. The key to all this is the following:

> As long as we translate between English and logic choosing the correct words to reflect the meaning we intend to convey then everything will be fine.

I have used the bold symbols **not**, **and**, etc above. But these are rooted in the English language. It would be preferable to use a notation which is independent of language. The table below shows what we shall use in this course.

English	**Symbol**	**Name**
not	$\neg$	negation
and	$\wedge$	conjunction
or	$\vee$	disjunction
xor	$\oplus$	exclusive disjunction
iff	$\leftrightarrow$	biconditional
implies	$\rightarrow$	conditional

Our symbol for **and** is essentially a capital 'A' with the cross-bar missing, and our symbol for **or** is the first letter of the Latin word *vel* which meant 'or'.

> A statement that cannot be analysed further using the propositional connectives is called an *atomic statement* or simply an

atom. Otherwise a statement is said to be *compound*. The truth value of a compound statement can be determined once the truth values of the atoms are known by applying the truth tables of the propositional connectives defined above.

Examples 1.1.1. Determine the truth values of the following statements.

1. (There is a Rhino under the table)$\vee\neg$(there is a Rhino under the table). [Always true]
2. $(1 + 1 = 3) \rightarrow (2 + 2 = 5)$. [True]
3. (Mickey Mouse is the President of the USA) $\leftrightarrow$ (pigs can fly). [Amazingly, true]

Example 1.1.2. Consider a simple traffic control system that consists of a red light and a green light. If the light is green traffic flows and if the light is red traffic stops. Let g be the statement 'the light is green', let r be the statement 'the light is red' and let s be the statement 'the traffic stops'. Then the following compound statements are all true: $r \oplus g$, $g \rightarrow \neg s$ and $r \rightarrow s$. Suppose I tell you that g is true. Then from the fact that $r \oplus g$ is true we deduce that r is false. Now $g \rightarrow \neg s$ is true and g is true and so $\neg s$ is true. Thus the traffic doesn't stop. Since $\neg s$ is true it follows that s is false. Now observe that in $r \rightarrow s$ both r and s are false but we are told that the statement $r \rightarrow s$ is true. This is entirely consistent with the way that we defined $\rightarrow$.

Example 1.1.3. My car has all kinds of warnings both audible and visual.[8] For example, 'the audible warning for headlamps sounds if the key is removed from the ignition and the driver's door is open and either the headlamps are on or the parking lamps are on.' Put p equal to the statement 'the key is removed from the ignition', put q equal to 'the driver's door is open' , put r equal to 'the headlamps are on' and put s equal to 'the parking lamps are on'. Put t equal to 'the audible warning for the headlamps sounds. Then t is true if $p \wedge q \wedge (r \vee s)$ is true. Observe that $r \vee s$ is the correct version of 'or'. We need at least the headlamps to be on or the parking lamps to be on but that should certainly include the possibility that both sets of 'lamps' are

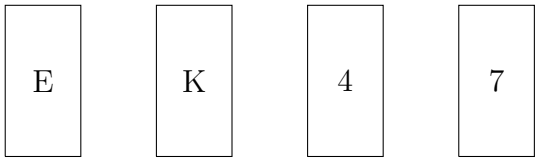
[8]The guidebook for my car is written in American English but the steering wheel is on the right side.

on. Although the guidebook uses the word 'if' I think it clear that it really means 'if and only if'. Thus we want t to be true precisely when $p \wedge q \wedge (r \vee s)$ is true. For example, I don't want the audible warning for the headlamps to sound if the 'gas' is low. Thus if $p \wedge q \wedge (r \vee s)$ is true the audible warning for the headlamps sounds and if $p \wedge q \wedge (r \vee s)$ is false then it doesn't.

What we have done so far is informal. I have just highlighted some features of everyday language. What we shall do next is formalize. I shall describe to you an artificial language called PL motivated by what we have found in this section. I shall first describe its syntax and then its semantics. Of course, I haven't shown you yet what we can actually do with this artificial language. That I shall do later.

Exercises 1.1

1. For each of the following statements, decide whether they are true or false.
   (a) The earth orbits the sun.
   (b) The moon orbits Mars.
   (c) Mars orbits the sun.
   (d) Mars orbits Jupiter
   (e) (The moon orbits Mars)$\leftrightarrow$(Mars orbits Jupiter).
   (f) (The earth orbits the sun)$\rightarrow$(Mars orbits the sun).
   (g) (The earth orbits the sun)$\oplus$(Mars orbits the sun).
   (h) (The earth orbits the sun)$\oplus$(Mars orbits Jupiter).
   (i) (The earth orbits the sun)$\vee$(Mars orbits the sun).
   (j) (The earth orbits the sun)$\vee\neg$(the earth orbits the sun).
2. The *Wason selection task*. Below are four cards each of which has a number printed on one side and a letter on the other.



The following claim is made: if a card has a vowel on one side then it has an even number on the other. What is the smallest number of cards you have to turn over to verify this claim and which cards are they?

1.2 Syntax of propositional logic

We are given a collection of symbols called *atomic statements* or *atoms*. I'll usually denote these with lower case letters $p, q, r, \ldots$ or their subscripted variants $p_1, p_2, p_3, \ldots$. A *well-formed formula* or *wff* is constructed in the following way:

(WFF1). All atoms are wff.

(WFF2). If A and B are wff then so too are $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \oplus B)$, $(A \to B)$ and $(A \leftrightarrow B)$.

(WFF3). All wff are constructed by repeated application of the rules (WFF1) and (WFF2) a finite number of times.

A wff which is not an atom is said to be a *compound statement*.

Example 1.2.1. We show that

$$(\neg((p \vee q) \wedge r))$$

is a wff.

1. p, q and r are wff by (WFF1).
2. $(p \vee q)$ is a wff by (1) and (WFF2).
3. $((p \vee q) \wedge r)$ is a wff by (1), (2) and (WFF2).
4. $(\neg((p \vee q) \wedge r))$ is a wff by (3) and (WFF2), as required

Notational convention. To make reading wff easier, I shall omit the outer brackets and also the brackets associated with $\neg$.

Examples 1.2.2.

1. $\neg p \vee q$ means $((\neg p) \vee q)$.

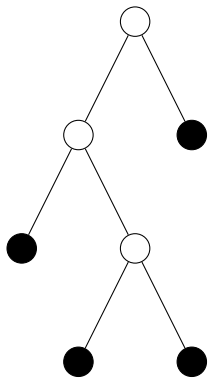
2. $\neg p \to (q \vee r)$ means $((\neg p) \to (q \vee r))$ and is different from $\neg((p \to q) \vee r)$.

I tend to bracket fairly heavily but many books on logic use fewer brackets and arrange the connectives in a hierarchy:

$$\neg, \wedge, \vee, \to, \leftrightarrow .$$

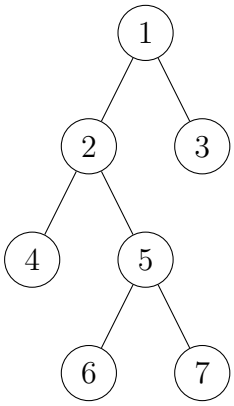
This means that if two connectives are next to each other, you apply first the one that is higher up the food-chain. We do this automatically in elementary arithmetic. For example, $a + b \cdot c$ means $a + (b \cdot c)$ because addition has higher precedence than multiplication. However, it pays to check what conventions an author is using.

There is a graphical way of representing wff that involves trees. A *tree* is a data structure consisting of circles called *nodes* or *vertices* joined by lines called *edges* such that there are no closed paths of distinct lines. In addition, the vertices are organized hierarchically. One vertex is singled out and called the *root* and is placed at the top.[9] The vertices are arranged in levels so that vertices at the same level cannot be joined by an edge. The vertices at the bottom are called *leaves*. The picture below is an example of a tree with the leaves being indicated by the filled circles. The root is the vertex at the top.



In the following tree I have numbered the vertices for convenience.

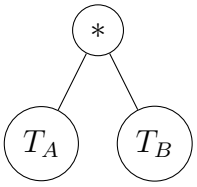
[9]Later, we shall deal with trees that are upside down versions of these and so will be arboreally correct in having their roots at the bottom.



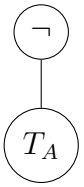
We say that vertices 2 and 3 are the *successors* of vertex 1. Likewise, vertices 4 and 5 are successors of vertex 2. Looked at the other way around, vertex 1 is the *predecessor* of vertices 2 and 3. Likewise, vertex 2 is the predecessor of vertices 4 and 5. A *path* in a tree begins at the root and then follows the edges down to another vertex. Thus $1-2-5$ is a path. A path that ends at a leaf is called a *branch*. Thus $1-2-5-7$ is a branch. A branch is therefore a maximal path.

A *parse tree of a wff* is constructed as follows. The parse tree of an atom p is the tree

Now let A and B be wff. Suppose that A has parse tree T_A and B has parse tree T_B. Let $*$ denote any of the binary propositional connectives. Then $A * B$ has the parse tree



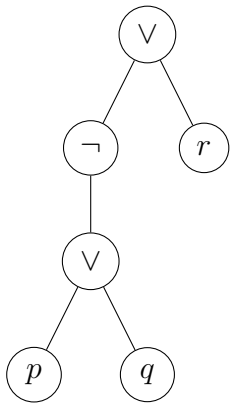
This is accomplished by joining the roots of T_A and T_B to a new root labelled by $*$. The parse tree for $\neg A$ is



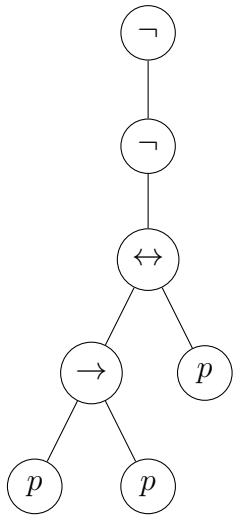
This is accomplished by joining the root of T_A to a new root labelled $\neg$. Parse trees are a way of representing wff without using brackets though we pay the price of having to work in two dimensions rather than one.

> We shall see in Chapter 2 that parse trees are in fact useful in circuit design.

Example 1.2.3. The parse tree for $\neg(p \vee q) \vee r$ is



Example 1.2.4. The parse tree for $\neg\neg((p \to p) \leftrightarrow p)$ is



I conclude this section with a twofold mathematical aside. We are often interested in collections of things. For example, the collection of wff. Rather than use the word 'collection', we will instead use the word 'set'. A *set* is

just a collection of things that we wish to view together. The things that make up the set are called its *elements*. The notion of set is an important one in mathematics, which is surprising since it seems so simple. There are a couple of pieces of notation that are used when talking about sets. To make clear what is in the set we use curly brackets { and } to mark the boundaries of the set. So, for example, the set of suits in a deck of cards is $\{\clubsuit, \spadesuit, \heartsuit, \diamondsuit\}$. But there doesn't have to be any rhyme or reason to what we put into a set. For example, I typed the following set $\{\bigstar, \P, \nabla, \blacksquare, \eth\}$ at random. A set is a single thing so it is usual to give it a name. What you call a set doesn't matter in principle, but in practice we use upper-case letters. So, I could write

$$S = \{\clubsuit, \spadesuit, \heartsuit, \diamondsuit\}.$$

The set S has as elements $\clubsuit$, $\spadesuit$, $\heartsuit$, and $\diamondsuit$. Our second piece of notation is to replace the phrase 'is an element of' by the symbol $\in$. Thus we would write '$\spadesuit \in S$' to mean '$\spadesuit$ is an element of S' and we would write '$\nabla \notin S$' to mean '∇ is not an element of S'. If A and B are two sets we say they are *equal*, and write $A = B$, if they contain exactly the same elements. Thus in order to check that $A = B$ you have to check two things:

1. If $a \in A$ then $a \in B$.

2. If $a \in B$ then $a \in A$.

It is possible for a set to have nothing in it: $\{\}$. However, you might think that I have simply forgotten to list any elements. To avoid such a misunderstanding, we represent the *empty set* by the symbol $\emptyset$. Thus $\{\} = \emptyset$. One slightly odd feature of sets is that the order in which we list the elements doesn't matter: a set is just a bag of elements not a display case.

The next part of our mathematical aside is the notion of a 'string'. Let A be a finite set. In certain circumstances, we can also call this set an *alphabet*. This term is often used when the elements of a set are going to be used to form 'strings' often with the goal of being used in communication. A *string* over the alphabet A is an ordered sequence of elements from A. For example, if $A = \{0, 1\}$ then the following are examples of strings over this alphabet: 0, 1, 00, 01, 10, 11, ... The *empty string* is the string that contains no elements of the alphabet and is denoted by ε. The total number of elements of the alphabet occurring in a string is called its *length*. Thus 00000 has length 5 and ε has length 0. Strings over the alphabet $\{0, 1\}$ are usually called

binary strings. A set of strings is called, suggestively, a *language*. Given two strings x and y we can stick them together to get a new string xy. This operation glories in the name of *concatenation*. Clearly, order matters. The concatenation of the strings 'go' and 'od' is the string 'good' whereas the concatenation of the strings 'od' and 'go' is 'odgo'.

Example 1.2.5. We now use the above terminology to describe what we have done in this section. Our starting point was the alphabet

$$A = \{p, q, r, \ldots, p_1, p_2, p_3, \ldots \neg, \wedge, \vee, \oplus, \rightarrow, \leftrightarrow, (,)\}.$$

We are not interested in all strings over this alphabet but only those strings constructed in accordance with the three rules (WFF1), (WFF2) and (WFF3). We call the set of such strings the *language of well-formed formulae*.

Exercises 1.2

---

1. Construct parse trees for the following wff.
   (a) $(\neg p \vee q) \leftrightarrow (q \rightarrow p)$.
   (b) $p \rightarrow ((q \rightarrow r) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r)))$.
   (c) $(p \rightarrow \neg p) \leftrightarrow \neg p$.
   (d) $\neg(p \rightarrow \neg p)$.
   (e) $(p \rightarrow (q \rightarrow r)) \leftrightarrow ((p \wedge q) \rightarrow r)$.

1.3 Semantics of propositional logic

An atomic statement is assumed to have one of two *truth values*: *true (T)* or *false (F)*. We now consider the truth values of those compound statements that contain exactly one of the Boolean connectives. The following *truth tables* **define** the meaning of the Boolean connectives.

p	$\neg p$
T	F
F	T

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

p	q	$p \rightarrow q$
T	T	T
$\boldsymbol{T}$	$\boldsymbol{F}$	$\boldsymbol{F}$
F	T	T
F	F	T

As we discussed in Section 1.1, the meanings of the logical connectives above are *suggested* by their meanings in everyday language, but are not the same as them. Think of our definitions as technical ones for technical purposes only.

> *It is vitally important in what follows that you learn the above truth tables by heart.*

Truth tables can also be used to work out the truth values of compound statements. Let A be a compound statement consisting of atoms $p_1, \ldots, p_n$. A specific *truth assignment* to $p_1, \ldots, p_n$ leads to a truth value being assigned to A itself by using the definitions above.

Example 1.3.1. Let $A = (p \vee q) \rightarrow (r \leftrightarrow \neg s)$. A truth assignment is given by the following table

p	q	r	s
T	F	F	T

If we insert these values into our wff we get

$$(T \vee F) \rightarrow (F \leftrightarrow \neg T).$$

We use our truth tables above to evaluate this expression in stages

$$T \rightarrow (F \leftrightarrow F), \quad T \rightarrow T, \quad T.$$

Thus the truth value of A for the above truth assignment is T.

Lemma 1.3.2. *If the compound proposition A consists of n atoms then there are 2^n possible truth assignments meaning that the truth table for A has 2^n rows.*

Proof. If $n = 1$ then there is only one atom and it has exactly two possible truth values and so the truth table for A will have exactly 2 rows. More generally, each row of a truth table corresponds to exactly one string over the alphabet $\{T, F\}$. Let's compare the number of such strings of length n with the number of strings of length $n + 1$. Each string y of length $n + 1$ is of the form Fx or Tx where x is a string of length n. It follows that every time we add an atom we double the number of rows of the truth table. So, if the truth table for a wff with n atoms has 2^n rows then the truth table for a wff with $n + 1$ atoms will have $2 \times 2^n = 2^{n+1}$ rows. But when $n = 1$ we really do have $2^n = 2^1$ rows. □

Example 1.3.3. Lemma 1.3.2 is less innocuous than it looks since it is an example of 'exponential explosion'. This is illustrated by the following story. There was once a king who commissioned a fabulous palace from an architect. So pleased was the king with the result, that he offered the architect as much gold as he could carry from the royal treasury. The architect said that just seeing his work completed was wealth enough but that if the king was really insistent on rewarding him he would be content with just the gold coins that could be placed on a chess board in the following way: one gold coin should be placed on the first square, two gold coins on the second square, four gold coins on the third square and so on. The king, not known for his mathematical acumen, readily agreed thinking this a small price to pay. In what way was the king duped by the architect?[10]

We may draw up a table, also called a *truth table*, whose rows consist of all possible truth assignments along with the corresponding truth value of A.

[10] Just to complete the story, the king did eventually realize his mistake, but being an enlightened monarch his vengeance was mild and the architect was merely required to spend the rest of his career designing houses people actually wanted to live in.

We shall use the following pattern of assignments of truth values:

...	T	T	T
...	T	T	F
...	T	F	T
...	T	F	F
...	F	T	T
...	F	T	F
...	F	F	T
...	F	F	F
...	...	...	...

Thus for the rightmost atom truth values alternate starting with T; for the next column to the left they alternate in pairs starting with TT; for the next column to the left they alternate in fours starting with $TTTT$; and so on.

Examples 1.3.4. Here are some examples of truth tables

1. The truth table for $A = \neg(p \to (p \vee q))$.

p	q	$p \vee q$	$p \to (p \vee q)$	A
T	T	T	T	F
T	F	T	T	F
F	T	T	T	F
F	F	F	T	F

2. The truth table for $B = (p \wedge (p \to q)) \to q$.

p	q	$p \to q$	$p \wedge (p \to q)$	B
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

3. The truth table for $C = (p \vee q) \wedge \neg r$.

p	q	r	$p \vee q$	$\neg r$	C
T	T	T	T	F	F
T	T	F	T	T	T
T	F	T	T	F	F
T	F	F	T	T	T
F	T	T	T	F	F
F	T	F	T	T	T
F	F	T	F	F	F
F	F	F	F	T	F

4. Given the wff $(p \wedge \neg q) \wedge r$ we could draw up a truth table but in this case we can easily figure out how it behaves. It is true if and only if p is true, $\neg q$ is true and r is true. Thus the following is a truth assignment that makes the wff true

p	q	r
T	F	T

and the wff is false for all other truth assignments. We shall generalize this example later.

Important definitions

- An atom or the negation of an atom is called a *literal*.
- We say that a wff A built up from the atomic propositions $p_1, \ldots, p_n$ is *satisfiable* if there is some assignment of truth values to the atoms in A which gives A the truth value true.
- If $A_1, \ldots, A_n$ are wff we say they are *(jointly) satisfiable* if there is a single truth assignment that makes all of $A_1, \ldots, A_n$ true.
- If a wff is always true we say that it is a *tautology*. If A is a tautology we shall write

  $$\vDash A.$$

  The symbol $\vDash$ is called the *semantic turnstile*.

- If a wff is always false we say it is a *contradiction*. If A is a contradiction we shall write
$$A \vDash .$$
Observe that contradictions are on the left (or sinister) side of the semantic turnstile.

- If a wff is sometimes true and sometimes false we refer to it as a *contingency*.

- A truth assignment that makes a wff true is said to *satisfy* the wff otherwise it is said to *falsify* the wff.

A very important problem in PL can now be stated.

The satisfiability problem (SAT)
Given a wff decide whether there is some truth assignment to the atoms that makes the wff take the value true. A program that solves SAT is called a *SAT solver*.

I shall discuss this problem in more detail later and explain why it is so significant.

It is important to remember that all questions in PL can be settled, at least in principle, by using truth tables.

Example 1.3.5. What's the problem with truth tables? Why did I say above 'in principle'? This goes back to Example 1.3.3, which we now look at in a more mathematical way. Suppose you want to construct the truth table of a wff that has 90 atoms (not a vast number by any means). Its truth table will therefore have 2^{90} rows. It is more useful to express this as a power of 10. You can check that $2 \approx 10^{0.3}$. Thus
$$2^{90} \approx (10^{0.3})^{90} = 10^{0.3 \times 90} \approx 10^{27}.$$
For the sake of argument, suppose that it takes you 10^{-9} seconds to construct each row of the truth table. Then it will take you $10^{18} = 10^{-9} \times 10^{27}$ seconds to construct the truth table. For comparison purposes, the age of the universe, give or take a few seconds, is 4.35×10^{17} seconds.

Exercises 1.3

1. Determine which of the following wff are satisfiable. For those which are, find all the assignments of truth values to the atoms which make the wff true.

   (a) $(p \wedge \neg q) \rightarrow \neg r$.

   (b) $(p \vee q) \rightarrow ((p \wedge q) \vee q)$.

   (c) $(p \wedge q \wedge r) \vee (p \wedge q \wedge \neg r) \vee (\neg p \wedge \neg q \wedge \neg r)$.

2. Determine which of the following wff are tautologies by using truth tables.

   (a) $(\neg p \vee q) \leftrightarrow (q \rightarrow p)$.

   (b) $p \rightarrow ((q \rightarrow r) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r)))$.

   (c) $(p \rightarrow \neg p) \leftrightarrow \neg p$.

   (d) $\neg(p \rightarrow \neg p)$.

   (e) $(p \rightarrow (q \rightarrow r)) \leftrightarrow ((p \wedge q) \rightarrow r)$.

3. We defined only 5 binary connectives, but there are in fact 16 possible ones. The tables below show all of them.

p	q	$\circ_1$	$\circ_2$	$\circ_3$	$\circ_4$	$\circ_5$	$\circ_6$	$\circ_7$	$\circ_8$
T	T	T	T	T	T	T	T	T	T
T	F	T	T	T	T	F	F	F	F
F	T	T	T	F	F	T	T	F	F
F	F	T	F	T	F	T	F	T	F

p	q	$\circ_9$	$\circ_{10}$	$\circ_{11}$	$\circ_{12}$	$\circ_{13}$	$\circ_{14}$	$\circ_{15}$	$\circ_{16}$
T	T	F	F	F	F	F	F	F	F
T	F	T	T	T	T	F	F	F	F
F	T	T	T	F	F	T	T	F	F
F	F	T	F	T	F	T	F	T	F

   Express each of the connectives from 1 to 16 in terms of at most $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\oplus$, p, q and brackets.

1.4 Logical equivalence

It can happen that two different-looking statements A and B have the same truth table. This means they have the same meaning. For example, 'it is not the case that it is not raining' is a bizarre way of saying 'it is raining'. In this section, we shall investigate this idea.

1.4.1 Definition

We begin with some examples.

Example 1.4.1. Compare the true tables of $p \to q$ and $\neg p \vee q$.

p	q	$p \to q$
T	T	T
T	F	F
F	T	T
F	F	T

p	q	$\neg p$	$\neg p \vee q$
T	T	F	T
T	F	F	F
F	T	T	T
F	F	T	T

They are clearly the same.

Example 1.4.2. Compare the true tables of $p \leftrightarrow q$ and $(p \to q) \wedge (q \to p)$.

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

p	q	$p \to q$	$q \to p$	$(p \to q) \wedge (q \to p)$
T	T	T	T	T
T	F	F	T	F
F	T	T	F	F
F	F	T	T	T

They are clearly the same.

Example 1.4.3. Compare the truth tables of $p \oplus q$ and $(p \vee q) \wedge \neg(p \wedge q)$.

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

p	q	$p \vee q$	$p \wedge q$	$\neg(p \wedge q)$	$(p \vee q) \wedge \neg(p \wedge q)$
T	T	T	T	F	F
T	F	T	F	T	T
F	T	T	F	T	T
F	F	F	F	T	F

They are clearly the same.

If the wff A and B have the same truth tables we say that A is *logically equivalent to* B written $A \equiv B$.

It is important to remember that $\equiv$ is not a logical connective. It is a *relation* between wff.

The above examples can now be expressed in the following way.

Examples 1.4.4.

1. $p \to q \equiv \neg p \vee q$.
2. $p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$.
3. $p \oplus q \equiv (p \vee q) \wedge \neg(p \wedge q)$.

However, it is important to observe that A and B need not have the same atoms but, in that case, the truth tables must be constructed using all the atoms that occur in either A or B. Here is an example.

Example 1.4.5. We prove that $p \equiv p \wedge (q \vee \neg q)$. We construct two truth tables with atoms p and q in both cases.

p	q	p
T	T	T
T	F	T
F	T	F
F	F	F

p	q	$p \wedge (q \vee \neg q)$
T	T	T
T	F	T
F	T	F
F	F	F

The two truth tables are the same and so the two wff are logically equivalent.

The following result is the first indication of the important role that tautologies play in propositional logic. You can also take this as the definition of logical equivalence since it avoids the issue of where the two wff use different numbers of atoms.

Proposition 1.4.6. *Let A and B be wff. Then $A \equiv B$ if and only if $\vDash A \leftrightarrow B$.*

Proof. The statement of the result is in fact two statements in one:

1. If $A \equiv B$ then $\vDash A \leftrightarrow B$.

2. If $\vDash A \leftrightarrow B$ then $A \equiv B$.

We use the fact that $X \leftrightarrow Y$ is true when X and Y have the same truth value. Let the atoms that occur in either A or B be $p_1, \ldots, p_n$. We now prove the two statements.

(1) Let $A \equiv B$ and suppose that $A \leftrightarrow B$ were not a tautology. Then there is some assignment of truth values to the atoms $p_1, \ldots, p_n$ such that A and B have different truth values. But this would imply that there was a row of the truth table of A that was different from the corresponding row of B. This contradicts the fact that A and B have the same truth tables. It follows that $A \leftrightarrow B$ is a tautology.

(2) Let $A \leftrightarrow B$ be a tautology and suppose that A and B have truth tables that differ. This implies that there is a row of the truth table of A that is different from the corresponding row of B. Thus there is some assignment of truth values to the atoms $p_1, \ldots, p_n$ such that A and B have different truth values. But this would imply that $A \leftrightarrow B$ is not a tautology. □

Example 1.4.7. Prove that $\vDash p \leftrightarrow (p \wedge (q \vee \neg q))$. This implies that $p \equiv p \wedge (q \vee \neg q)$.

p	q	$p \wedge (q \vee \neg q)$	$p \leftrightarrow (p \wedge (q \vee \neg q))$
T	T	T	T
T	F	T	T
F	T	F	T
F	F	F	T

1.4.2 Important logical equivalences

As you study PL, you will find that certain logical equivalences constantly recur. The following theorem lists the most important ones. You will be asked to prove them all in the exercises to this section.

Theorem 1.4.8.

1. $\neg\neg p \equiv p$. *Double negation.*

2. $p \wedge p \equiv p$ *and* $p \vee p \equiv p$. *Idempotence.*

3. $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$ *and* $(p \vee q) \vee r \equiv p \vee (q \vee r)$. *Associativity.*

4. $p \wedge q \equiv q \wedge p$ *and* $p \vee q \equiv q \vee p$. *Commutativity.*

5. $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$ *and* $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$. *Distributivity.*

6. $\neg(p \wedge q) \equiv \neg p \vee \neg q$ *and* $\neg(p \vee q) \equiv \neg p \wedge \neg q$. *De Morgan's laws.*

7. $p \vee (p \wedge q) \equiv p$ *and* $p \wedge (p \vee q) \equiv p$. *Absorption.*

I will refer to the above results by name in many subsequent calculations, so it is important to learn them all now.

There are some interesting patterns in the above results that involve the interplay between $\wedge$ and $\vee$:

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

and

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

There are also some important consequences of the above theorem.

- The fact that $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$ means that we can write simply $p \wedge q \wedge r$ without ambiguity because the two ways of bracketing this expression lead to the same truth table. It can be shown that as a result we can write expressions like $p_1 \wedge p_2 \wedge p_3 \wedge p_4$ (and so on) without brackets because it can be proved that however we bracket such an expression leads to the same truth table. What we have said for $\wedge$ also applies to $\vee$.

- Let $A_1, \ldots, A_n$ be wff. We abbreviate

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n$$

by

$$\bigwedge_{i=1}^{n} A_i.$$

Similarly, we abbreviate

$$A_1 \vee A_2 \vee \ldots \vee A_n$$

by

$$\bigvee_{i=1}^{n} A_i.$$

- The fact that $p \wedge q \equiv q \wedge p$ implies that the order in which we carry out a sequence of conjunctions does not matter. What we have said for $\wedge$ also applies to $\vee$.

- The fact $p \wedge p \equiv p$ means that we can eliminate repeats in conjunctions of one and the same atom. What we have said for $\wedge$ also applies to $\vee$.

Example 1.4.9. By the results above

$$p \wedge q \wedge p \wedge q \wedge p \equiv p \wedge q.$$

It is important not to overgeneralize the above results as the following two examples show.

Example 1.4.10. Observe that $p \to q \not\equiv q \to p$ since the truth assignment

p	q
T	F

makes the LHS equal to F but the RHS equal to T.

Example 1.4.11. Observe that $(p \to q) \to r \not\equiv p \to (q \to r)$ since the truth assignment

p	q	r
F	F	F

makes the left hand side equal to F but the right hand side equal to T.

1.4.3 Further examples

Our next example is an application of some of our results.

Example 1.4.12. We have defined a binary propositional connective $\oplus$ such that $p \oplus q$ is true when exactly one of p or q is true. Our goal now is to extend this to three atoms. Define $\text{xor}(p, q, r)$ to be true when exactly one of p, q or r is true, and false in all other cases. We can describe this connective in terms of the ones already defined. I claim that

$$\text{xor}(p, q, r) = (p \vee q \vee r) \wedge \neg(p \wedge q) \wedge \neg(p \wedge r) \wedge \neg(q \wedge r).$$

This can easily be verified by constructing the truth table of the RHS. Put

$$A = (p \vee q \vee r) \wedge \neg(p \wedge q) \wedge \neg(p \wedge r) \wedge \neg(q \wedge r).$$

p	q	r	$p \vee q \vee r$	$\neg(p \wedge q)$	$\neg(p \wedge r)$	$\neg(q \wedge r)$	A
T	T	T	T	F	F	F	F
T	T	F	T	F	T	T	F
T	F	T	T	T	F	T	F
$\boldsymbol{T}$	F	F	T	T	T	T	$\boldsymbol{T}$
F	T	T	T	T	T	F	F
F	$\boldsymbol{T}$	F	T	T	T	T	$\boldsymbol{T}$
F	F	$\boldsymbol{T}$	T	T	T	T	$\boldsymbol{T}$
F	F	F	F	T	T	T	F

The following properties of logical equivalence will be important when we come to show how Boolean algebras are related to PL in Chapter 2.

Proposition 1.4.13. *Let A, B and C be wff.*

1. $A \equiv A$.
2. *If* $A \equiv B$ *then* $B \equiv A$.
3. *If* $A \equiv B$ *and* $B \equiv C$ *then* $A \equiv C$.
4. *If* $A \equiv B$ *then* $\neg A \equiv \neg B$.
5. *If* $A \equiv B$ *and* $C \equiv D$ *then* $A \wedge C \equiv B \wedge D$.
6. *If* $A \equiv B$ *and* $C \equiv D$ *then* $A \vee C \equiv B \vee D$.

Proof. By way of an example, I shall prove (6). We are given that $A \equiv B$ and $C \equiv D$ and we have to prove that $A \vee C \equiv B \vee D$. That is we need to prove that from $\vDash A \leftrightarrow B$ and $\vDash C \leftrightarrow D$ we can deduce $\vDash (A \vee C) \leftrightarrow (B \vee D)$. Suppose that $(A \vee C) \leftrightarrow (B \vee D)$ is not a tautology. Then there is some truth assignment to the atoms that makes $A \vee C$ true and $B \vee D$ false or vice versa. I shall just deal with the first case here. Suppose that $A \vee C$ is true and $B \vee D$ is false. Then both B and D are false and at least one of A and C is true. If A is true then this contradicts $A \equiv B$, and if C is true then this contradicts $C \equiv D$. It follows that $A \vee C \equiv B \vee D$, as required. □

Logical equivalence can be used to *simplify* complicated compound statements as follows. Let A be a compound statement which contains occurrences of the wff X. Suppose that $X \equiv Y$ where Y is simpler than X. Let A' be the same as A except that some or all occurrences of X are replaced by Y. Then $A' \equiv A$ but A' is simpler than A.

Example 1.4.14. Let

$$A = p \wedge (q \vee \neg q) \wedge q \wedge (r \vee \neg r) \wedge r \wedge (p \vee \neg p).$$

But

$$p \wedge (q \vee \neg q) \equiv p \text{ and } q \wedge (r \vee \neg r) \equiv q \text{ and } r \wedge (p \vee \neg p) \equiv r$$

and so

$$A \equiv p \wedge q \wedge r.$$

Finally, we can also use known logical equivalences to prove new ones without having to construct truth tables.

Examples 1.4.15. Here are some examples of using known logical equivalences to show that two wff are logically equivalent.

1. We show that $p \to q \equiv \neg q \to \neg p$.

$$\begin{aligned} \neg q \to \neg p &\equiv \neg\neg q \vee \neg p \text{ by Example 4.1(1)} \\ &\equiv q \vee \neg p \text{ by double negation} \\ &\equiv \neg p \vee q \text{ by commutativity} \\ &\equiv p \to q \text{ by Example 4.1(1).} \end{aligned}$$

2. We show that $(p \to q) \to q \equiv p \vee q$.

$$\begin{aligned}(p \to q) \to q &\equiv \neg(p \to q) \vee q \text{ by Example 4.1(1)}\\ &\equiv \neg(\neg p \vee q) \vee q \text{ by Example 4.1(1)}\\ &\equiv (\neg\neg p \wedge \neg q) \vee q \text{ by De Morgan}\\ &\equiv (p \wedge \neg q) \vee q \text{ by double negation}\\ &\equiv (p \vee q) \wedge (\neg q \vee q) \text{ by distributivity}\\ &\equiv p \vee q \text{ since } \vDash \neg q \vee q.\end{aligned}$$

3. We show that $p \to (q \to r) \equiv (p \wedge q) \to r$.

$$\begin{aligned}p \to (q \to r) &\equiv \neg p \vee (q \to r) \text{ by Example 4.1(1)}\\ &\equiv \neg p \vee (\neg q \vee r) \text{ by Example 4.1(1)}\\ &\equiv \neg(p \wedge q) \vee r \text{ by associativity and De Morgan}\\ &\equiv (p \wedge q) \to r \text{ by Example 4.1(1).}\end{aligned}$$

4. We show that $p \to (q \to r) \equiv q \to (p \to r)$.

$$\begin{aligned}p \to (q \to r) &\equiv \neg p \vee (q \to r) \text{ by Example 4.1(1)}\\ &\equiv \neg p \vee (\neg q \vee r) \text{ by Example 4.1(1)}\\ &\equiv (\neg p \vee \neg q) \vee r \text{ by associativity}\\ &\equiv (\neg q \vee \neg p) \vee r \text{ by commutativity}\\ &\equiv \neg q \vee (\neg p \vee r) \text{ by associativity}\\ &\equiv \neg q \vee (p \to r) \text{ by Example 4.1(1)}\\ &\equiv q \to (p \to r) \text{ by Example 4.1(1).}\end{aligned}$$

5. We show that $(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$.

$$\begin{aligned}(p \to q) \wedge (p \to r) &\equiv (\neg p \vee q) \wedge (\neg p \vee r) \text{ by Example 4.1(1)}\\ &\equiv \neg p \vee (q \wedge r) \text{ by distributivity}\\ &\equiv p \to (q \wedge r) \text{ by Example 4.1(1).}\end{aligned}$$

6. We show that $\vDash p \to (q \to p)$ by using logical equivalences. We write $\boldsymbol{t}$ to denote a statement that is always true.

$$\begin{aligned}p \to (q \to p) &\equiv \neg p \vee (\neg q \vee p) \text{ by Example 4.1(1)}\\ &\equiv (\neg p \vee p) \vee \neg q \text{ by associativity and commutativity}\\ &\equiv \boldsymbol{t} \text{ since } \vDash \neg p \vee p.\end{aligned}$$

Exercises 1.4

1. Prove the following logical equivalences using truth tables.

   (a) $\neg\neg p \equiv p$. Double negation.

   (b) $p \wedge p \equiv p$ and $p \vee p \equiv p$. Idempotence.

   (c) $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$ and $(p \vee q) \vee r \equiv p \vee (q \vee r)$. Associativity.

   (d) $p \wedge q \equiv q \wedge p$ and $p \vee q \equiv q \vee p$. Commutativity.

   (e) $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$ and $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$. Distributivity.

   (f) $\neg(p \wedge q) \equiv \neg p \vee \neg q$ and $\neg(p \vee q) \equiv \neg p \wedge \neg q$. De Morgan's laws.

2. Let $\boldsymbol{f}$ stand for any wff which is a contradiction and $\boldsymbol{t}$ stand for any wff which is a tautology. Prove the following.

   (a) $p \vee \neg p \equiv \boldsymbol{t}$.

   (b) $p \wedge \neg p \equiv \boldsymbol{f}$.

   (c) $p \vee \boldsymbol{f} \equiv p$.

   (d) $p \vee \boldsymbol{t} \equiv \boldsymbol{t}$.

   (e) $p \wedge \boldsymbol{f} \equiv \boldsymbol{f}$.

   (f) $p \wedge \boldsymbol{t} \equiv p$.

   (g) $p \rightarrow \boldsymbol{f} \equiv \neg p$.

   (h) $\boldsymbol{t} \rightarrow p \equiv p$.

3. Show that $p \oplus (q \oplus r) \equiv (p \oplus q) \oplus r$.

4. Prove the following by using known logical equivalences (rather than using truth tables).

   (a) $(p \rightarrow q) \wedge (p \vee q) \equiv q$.

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

   (c) $p \rightarrow (q \vee r) \equiv (p \rightarrow q) \vee (p \rightarrow r)$.

5. (a) Use [48] to construct the truth table for the following wff.

$$A(p,q,r,s) = (p \vee q \vee r \vee s) \wedge \neg(p \wedge q) \wedge \neg(p \wedge r) \wedge \neg(p \wedge s) \\ \wedge \neg(q \wedge r) \wedge \neg(q \wedge s) \wedge \neg(r \wedge s).$$

   Describe in words the meaning of $A(p,q,r,s)$.

   (b) Generalize this construction to define a similar wff of the form $A(p_1,\ldots,p_n)$ where n is arbitrary.

6. This question is a proof of the *Duality Theorem* which makes precise the parallels between $\wedge$ and $\vee$. Note that we shall only deal with wff constructed using the connectives $\neg, \vee, \wedge$ in this question. If A is any such wff, we denote by A^* the wff obtained from A by replacing every occurrence of $\wedge$ in A by $\vee$, and every occurrence of $\vee$ in A by $\wedge$. Prove that $\vDash A \leftrightarrow B$ if and only if $\vDash A^* \leftrightarrow B^*$.

1.5 PL in action

Once you have absorbed the basic definitions and mastered the notation, you might well experience a sinking feeling: is that it? PL can seem like a toy and some of our examples don't exactly help this impression, but in fact it has serious applications independently of its being the foundation of the more general first-order logic that we shall study in Chapter 3. In this section, we shall describe two applications of PL.

1.5.1 PL as a 'programming language'

In this section, we shall describe an example in which PL is used as a sort of programming language. We do this by analyzing a couple of examples of simplified Sudoku-type problems in terms of PL. These illustrate the ideas needed to analyse full Sudoku in terms of PL which is set as an exercise. In fact, many important problems in mathematics and computer science can be regarded as instances of the satisfiability problem. We shall say more about this in Section 1.8.

To understand new ideas always start with the simplest examples. So, here is a childishly simple Sudoku puzzle. Consider the following grid:

where the small squares are called *cells*. The puzzle consists in filling the cells with numbers according to the following two constraints.

(C1) Each cell contains exactly one of the numbers 1 or 2.

(C2) If two cells occur in the same row then the numbers they contain must be different.

There are obviously exactly two solutions to this puzzle

1	2	and	2	1

I shall now show how this puzzle can be encoded by a wff of PL. Please note that I shall solve it in a way that generalizes so I do not claim that the solution in this case is the simplest. We first have to decide what the atoms are. To define them we shall label the cells as follows

c_{11}	c_{12}

We need four atoms that are defined as follows.

- p is the statement that cell c_{11} contains the number 1.
- q is the statement that cell c_{11} contains the number 2.
- r is the statement that cell c_{12} contains the number 1.
- s is the statement that cell c_{12} contains the number 2.

For example, if p is true then the grid looks like this

1	?

where the ? indicates that we don't care what is there. Consider now the following wff.

$$A = (p \oplus q) \wedge (r \oplus s) \wedge (p \oplus r) \wedge (q \oplus s).$$

I now describe what each of the parts of this wff are doing.

- $p \oplus q$ is true precisely when cell c_{11} contains a 1 or a 2 but not both.
- $r \oplus s$ is true precisely when cell c_{12} contains a 1 or a 2 but not both.

- $p \oplus r$ is true precisely when the number 1 occurs in exactly one of the cells c_{11} and c_{12}.

- $q \oplus s$ is true precisely when the number 2 occurs in exactly one of the cells c_{11} and c_{12}.

Here is the important consequence of all this:

> It follows that A is satisfiable precisely when the puzzle can be solved. In addition, each satisfying truth assignment can be used to read off a solution to the original puzzle.

Here is the truth table for A.

p	q	r	s	A
T	T	T	T	F
T	T	T	F	F
T	T	F	T	F
T	T	F	F	F
T	F	T	T	F
T	F	T	F	F
T	F	F	T	T
T	F	F	F	F
F	T	T	T	F
F	T	T	F	T
F	T	F	T	F
F	T	F	F	F
F	F	T	T	F
F	F	T	F	F
F	F	F	T	F
F	F	F	F	F

We observe first that the wff A is satisfiable and so the original problem can be solved. Second, here are the two satisfying truth assignments.

p	q	r	s
T	F	F	T
F	T	T	F

The first truth assignment tells us that $c_{11} = 1$ and $c_{12} = 2$, whereas the second truth assignment tells us that $c_{11} = 2$ and $c_{12} = 1$. These are, of course, the two solutions we saw earlier.

We now describe a slightly more complex example and generalize what we did above. Consider the following slightly larger grid:

3		
		2

where again the small squares are called *cells*. Some cells contain numbers at the beginning and these must not be changed. Our task is to fill the remaining cells with numbers according to the following constraints.

(C1) Each cell contains exactly one of the numbers 1 or 2 or 3.

(C2) If two cells occur in the same row then the numbers they contain must be different.

(C3) If two cells occur in the same column then the numbers they contain must be different.

It is very easy to solve this problem satisfying these constraints to obtain

3	2	1
1	3	2
2	1	3

I shall now show how this problem can be represented in PL and how its solution is a special case of the satisfiability problem. First of all, I shall label the cells in the grid as follows:

c_{11}	c_{12}	c_{13}
c_{21}	c_{22}	c_{23}
c_{31}	c_{32}	c_{33}

The label c_{ij} refers to the cell in row i and column j. PL requires atomic statements. To model this problem we shall need 27 atomic statements c_{ijk} where $1 \leq i \leq 3$ and $1 \leq j \leq 3$ and $1 \leq k \leq 3$. The atomic statement c_{ijk} is defined as follows

$$c_{ijk} = \text{the cell in row } i \text{ and column } j \text{ contains the number } k.$$

For example, the atomic statement c_{113} is true when the grid is as follows:

3	?	?
?	?	?
?	?	?

where the ?s mean that we don't care what is in that cell. In the above case, the atomic statements c_{111} and c_{112} are both false.

We shall now construct a wff A from the above 27 atoms such that A is satisfiable if and only if the above problem can be solved and such that a satisfying truth assignment can be used to read off a solution. I shall construct A in stages.

- Define $I = c_{113} \wedge c_{232}$. This wff is true precisely when the grid looks like this

3	?	?
?	?	2
?	?	?

- Each cell must contain exactly one of the numbers $1, 2, 3$. For each $1 \leq i \leq 3$ and $1 \leq j \leq 3$ the wff

  $$\mathrm{xor}(c_{ij1}, c_{ij2}, c_{ij3})$$

  is true when the cell in row i and column j contains exactly one of the numbers $1, 2, 3$. Put B equal to the conjunction of all of these wff. Thus

  $$B = \bigwedge_{i=1}^{i=3} \bigwedge_{j=1}^{j=3} \mathrm{xor}(c_{ij1}, c_{ij2}, c_{ij3})$$

  where the notation means that you take the conjunction of all the terms $\mathrm{xor}(c_{ij1}, c_{ij2}, c_{ij3})$ for all possible values of i and j where $1 \leq i \leq 3$ and $1 \leq j \leq 3$. Thus there are nine terms to be conjoined from $\mathrm{xor}(c_{111}, c_{112}, c_{113})$ through to $\mathrm{xor}(c_{331}, c_{332}, c_{333})$. Then B is true precisely when each cell of the grid contains exactly one of the numbers $1, 2, 3$.

- In each row, each of the numbers $1, 2, 3$ must occur exactly once. For each $1 \leq i \leq 3$, define

  $$R_i = \mathrm{xor}(c_{i11}, c_{i21}, c_{i31}) \wedge \mathrm{xor}(c_{i12}, c_{i22}, c_{i32}) \wedge \mathrm{xor}(c_{i13}, c_{i23}, c_{i33}).$$

Then R_i is true when each of the numbers $1, 2, 3$ occurs exactly once in the cells in row i. Define $R = \bigwedge_{i=1}^{i=3} R_i$.

- In each column, each of the numbers $1, 2, 3$ must occur exactly once. For each $1 \leq j \leq 3$, define

$$C_j = \mathrm{xor}(c_{1j1}, c_{2j1}, c_{3j1}) \wedge \mathrm{xor}(c_{1j2}, c_{2j2}, c_{3j2}) \wedge \mathrm{xor}(c_{1j3}, c_{2j3}, c_{3j3}).$$

  Then C_j is true when each of the numbers $1, 2, 3$ occurs exactly once in the cells in column j. Define $C = \bigwedge_{j=1}^{i=3} C_i$.

- Put $A = I \wedge B \wedge R \wedge C$. Then by construction A is satisfiable precisely when the original problem is satisfiable and a satisfying truth assignment to the atoms can be used to read off a solution as follows. Precisely the following atoms are true:

$$c_{113}, c_{122}, c_{131}, c_{211}, c_{223}, c_{232}, c_{312}, c_{321}, c_{333}$$

  and all the remainder are false.

Don't be perturbed by the number of atoms in the above examples nor by the labour needed to write down the wff A. The point is that the problem can be faithfully described by a wff in PL and that the solution of the problem is achieved via a satisfying assignment of the atoms. Thus although PL is quite an impoverished language, it can nevertheless be used to describe quite complex problems. The 'trick' lies in the choice of the atoms. We shall place this example in a more general context in Section 1.8. In many ways, PL is like an *assembly language* and it is perfectly adapted to studying a particular class of problems that are widespread and important. There is a final point. People solve Sudoku puzzles rather quickly (and even do so for entertainment) and are clearly not solving them using truth tables but are instead applying logical rules. We shall say more about this approach later on in this chapter.

1.5.2 *PL models computers*

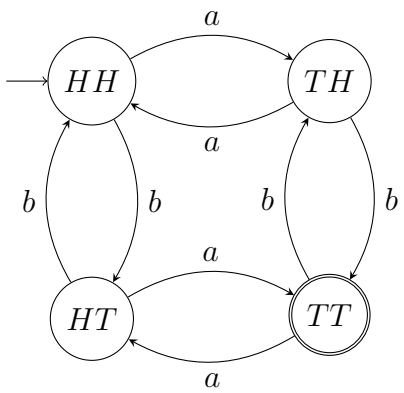
In this section, we shall show that PL can be used to describe finite state automata. These can be viewed as the simplest mathematical models of

computers.[11] We shall meet them again in Chapter 2, where they are used to describe the behaviour of computer circuits where there is memory. My aim here is to convey the idea of such simple machines and show that their behaviour may be modelled using PL. The following example is plagiarized from my book [27].

Given two coins, there are four possible ways of arranging them depending on which is heads (H) and which is tails (T):

$$\mathrm{HH}, \mathrm{TH}, \mathrm{HT}, \mathrm{TT}.$$

We call these the *states* of our system. Now consider the following two operations: 'flip the first coin,' which I shall denote by a, and 'flip the second coin,' which I shall denote by b. Assume that initially the coins are laid out as HH. This is our *initial state*. I am interested in all the possible ways of applying the operations a and b so that the coins are laid out as TT which will be our *terminal state*. The following diagram illustrates the relationships between the states and the two operations.



I have marked the start state with an inward-pointing arrow, and the terminal state by a double circle. If we start in the state HH and input the string *aba* Then we pass through the following states:

$$\mathrm{HH} \xrightarrow{a} \mathrm{TH} \xrightarrow{b} \mathrm{TT} \xrightarrow{a} \mathrm{HT}.$$

Thus the overall effect of starting at HH and inputting the string *aba* is to end up in the state HT. We say that a string over the alphabet $\{a, b\}$ is *accepted*

[11] They are, as it happens, special kinds of Turing machines which are the more refined mathematical models of computers. In any event, they are another legacy of Turing's work.

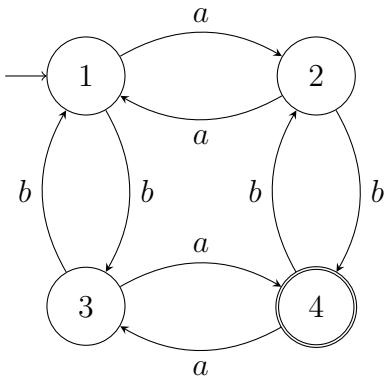
if it labels a path from the initial state to a terminal state. It should be clear that in this case a string is accepted if and only if the number of as is odd and the number of bs is odd. This is the *language accepted by this machine.*

If H is interpreted as 1 and T as 0 then our four states can be regarded as the possible contents of two memory locations in a computer: 00, 10, 01, 11. The input a flips the input in the first location and the input b flips the input in the second location.

More generally, a finite state automaton has a finite number of states, represented in a diagram by circles that are usually labelled in some way, in which one state is singled out as the initial state and one or more states are identified as terminal states. Associated with the automaton is a finite alphabet $A = \{a_1, \ldots, a_n\}$. For each state s and each input letter a_i from A there is a uniquely determined next state t so that $s \xrightarrow{a_i} t$.

We shall now describe how the behaviour of a finite state automaton can be encoded using PL. Recall that the wff $\mathrm{xor}(p_1, p_2, p_3, p_4)$ is true when exactly one of p_1, p_2, p_3, p_4 is true and false otherwise. See Question 5 of Exercises 1.4 for an explicit description of a wff that does this job.

Here is our original automaton but I have now numbered the states. The point is, we don't need to know the internal workings of the states — that they are represented by two coins — since the diagram tells us exactly how the automaton behaves.



Our automaton changes state and so there is an implicit assumption that time will play a role. For us, time will be discretized and so will take values $t = 0, 1, 2, 3, \ldots$. At each time t, the automaton will be in exactly one of four states. Denote the state of the automaton at time t by $q(t)$. We can

therefore write

$$\text{xor}\,((q(t)=1),(q(t)=2),(q(t)=3),(q(t)=4))$$

which does indeed say that at time t the automaton is in exactly one of its four states. If the automaton is in state $q(t)$ then its next state will depend on the input at time t. We denote the input at time t by $i(t)$. We can write

$$(i(t)=a)\oplus(i(t)=b)$$

because at time t either an a is input or a b is input, but not both.

> *It's important to observe that there are potentially infinitely many atoms (because the clock ticks: 0,1,2,3,...) but in processing a given input string we shall use only finitely many of them.*

We now have to describe the behaviour of the automaton in terms of PL. This is given by the following list of wff where there is one wff in each group for each value of t.

- $1 \xrightarrow{a} 2$ is described by the following wff

$$(q(t)=1)\wedge(i(t)=a)\rightarrow(q(t+1)=2).$$

- $1 \xrightarrow{b} 3$ is described by the following wff

$$(q(t)=1)\wedge(i(t)=b)\rightarrow(q(t+1)=3).$$

- $2 \xrightarrow{a} 1$ is described by the following wff

$$(q(t)=2)\wedge(i(t)=a)\rightarrow(q(t+1)=1).$$

- $2 \xrightarrow{b} 4$ is described by the following wff

$$(q(t)=2)\wedge(i(t)=b)\rightarrow(q(t+1)=4).$$

- $3 \xrightarrow{a} 4$ is described by the following wff

$$(q(t)=3)\wedge(i(t)=a)\rightarrow(q(t+1)=4).$$

- $3 \xrightarrow{b} 1$ is described by the following wff
$$(q(t) = 3) \wedge (i(t) = b) \rightarrow (q(t+1) = 1).$$
- $4 \xrightarrow{a} 3$ is described by the following wff
$$(q(t) = 4) \wedge (i(t) = a) \rightarrow (q(t+1) = 3).$$
- $4 \xrightarrow{b} 2$ is described by the following wff
$$(q(t) = 4) \wedge (i(t) = b) \rightarrow (q(t+1) = 2).$$

Finally, at $t = 0$ we start the automaton in its initial state. Thus $q(0) = 1$ is true.

Suppose we input a string of length 2. To do this, we need to specify $i(0)$ and $i(1)$. For example, if we input the string ab then the following wff is true: $(i(0) = a) \wedge (i(1) = b)$. The behaviour of our automaton in processing this string is now completely described by the conjunction of the following wff which we call A:

- Initial state: $q(0) = 1$.
- Input constraints:
$$((i(0) = a) \oplus (i(0) = b)) \wedge ((i(1) = a) \oplus (i(1) = b)).$$
- Unique state at each instant of time $t = 0$, $t = 1$ and $t = 2$:
$$\begin{aligned} & \operatorname{xor}((q(0) = 1), (q(0) = 2), (q(0) = 3), (q(0) = 4)) \\ \wedge \quad & \operatorname{xor}((q(1) = 1), (q(1) = 2), (q(1) = 3), (q(1) = 4)) \\ \wedge \quad & \operatorname{xor}((q(2) = 1), (q(2) = 2), (q(2) = 3), (q(2) = 4)) . \end{aligned}$$
- First set of possible state transitions:
$$\begin{aligned} & (q(0) = 1) \wedge (i(0) = a) \rightarrow (q(1) = 2) \\ \wedge \quad & (q(0) = 1) \wedge (i(0) = b) \rightarrow (q(1) = 3) \\ \wedge \quad & (q(0) = 2) \wedge (i(0) = a) \rightarrow (q(1) = 1) \\ \wedge \quad & (q(0) = 2) \wedge (i(0) = b) \rightarrow (q(1) = 4) \\ \wedge \quad & (q(0) = 3) \wedge (i(0) = a) \rightarrow (q(1) = 4) \\ \wedge \quad & (q(0) = 3) \wedge (i(0) = b) \rightarrow (q(1) = 1) \\ \wedge \quad & (q(0) = 4) \wedge (i(0) = a) \rightarrow (q(1) = 3) \\ \wedge \quad & (q(0) = 4) \wedge (i(0) = b) \rightarrow (q(1) = 2). \end{aligned}$$

- Second set of possible state transitions:

$$\begin{aligned}
& (q(1) = 1) \wedge (i(1) = a) \rightarrow (q(2) = 2) \\
\wedge\ & (q(1) = 1) \wedge (i(1) = b) \rightarrow (q(2) = 3) \\
\wedge\ & (q(1) = 2) \wedge (i(1) = a) \rightarrow (q(2) = 1) \\
\wedge\ & (q(1) = 2) \wedge (i(1) = b) \rightarrow (q(2) = 4) \\
\wedge\ & (q(1) = 3) \wedge (i(1) = a) \rightarrow (q(2) = 4) \\
\wedge\ & (q(1) = 3) \wedge (i(1) = b) \rightarrow (q(2) = 1) \\
\wedge\ & (q(1) = 4) \wedge (i(1) = a) \rightarrow (q(2) = 3) \\
\wedge\ & (q(1) = 4) \wedge (i(1) = b) \rightarrow (q(2) = 2).
\end{aligned}$$

- All the wff above are generic and can easily be written down for any string of length n where, here, $n = 2$. Finally, there is the actual input string of length 2.

$$(i(0) = a) \wedge (i(1) = b).$$

There is exactly one truth assignment making A true. This assigns the truth value T to the atoms $q(0) = 1$, $q(1) = 2$ and $q(2) = 4$ with all other truth assignments to the wff of the form $q(t) = k$ being false. Observe that a string of length n is accepted if and only if the state of the automaton at time $t = n$ is one of the terminal states. This, too, can easily be encoded by means of a wff in PL.

> There are different kinds of automata. The kind defined in this section are called *finite state acceptors*. I shall use the term *finite state automaton* to mean simply an acceptor with initial and terminal states not marked. A *Mealy machine* is a finite state automaton with an initial state marked and, in addition, each transition also contains an output. A *Moore machine* is a finite state automaton with an initial state marked and, in addition, each state also contains an output. We shall say a little more about the various kinds of finite state automata in Section 2.4.

Exercises 1.5

---

1. The goal of this question is to construct a wff A for solving the following Sudoku-type problem. We use a 2×2 array, labelled as follows.

c_{11}	c_{12}
c_{21}	c_{22}

The atoms are:

- $p =$ 'c_{11} contains 1.'
- $q =$ 'c_{11} contains 2.'
- $r =$ 'c_{12} contains 1.'
- $s =$ 'c_{12} contains 2.'
- $t =$ 'c_{21} contains 1.'
- $u =$ 'c_{21} contains 2.'
- $v =$ 'c_{22} contains 1.'
- $w =$ 'c_{22} contains 2.'

(a) Construct A.

(b) How many rows will the truth table for A contain?

(c) For how many of these rows will the value of A be T?

(d) How will these rows relate to the solution of the Sudoku?

(e) Use the truth table generator [48] for your wff and check whether your answers to the above questions were correct.

2. This question will talk you through the PL solution to full Sudoku. Each Sudoku is a 9×9 grid of *cells*. Each cell is located by giving its *row number* i and its *column number* j. Thus we may refer to the cell c_{ij} where $1 \leq i \leq 9$ and $1 \leq j \leq 9$. Each cell can contain a digit k where $1 \leq k \leq 9$. There will therefore be $9 \times 9 \times 9 = 729$ atoms defined as follows:

$$p_{i,j,k} = \text{'}c_{ij} \text{ contains the digit } k\text{'}.$$

(a) To initialize the problem, define I to be a wff in the above atoms that describes the initial distribution of digits in the cells. Write down the wff I in the case of the Sudoku puzzle in Question 6 of the Introductory Exercises. How many atoms are needed?

(b) Write down the wff $E_{i,j}$ which says that c_{ij} contains exactly one digit.

(c) Write down the wff E which says that every cell contains exactly one digit.

(d) Write down the wff R_i which says that row i contains each of the digits exactly once.

(e) Write down the wff R which says that in each row, each digit occurs exactly once in that row.

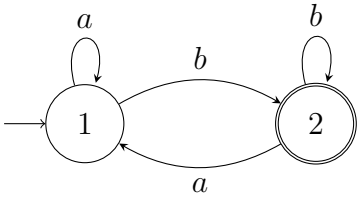
(f) Write down the wff C which says that in each column, each digit occurs exactly once in that column.

(g) The Sudoku grid is also divided into 9 *blocks* each containing 3×3 cells. Number these blocks $1, 2, \ldots, 9$ from top to bottom and left to right. Write down the wff W_l which says that in block l each of the nine digits occurs exactly once.

(h) Write down the wff W which says that in each block, each of the digits occurs exactly once.

(i) Put $P = I \wedge E \wedge R \wedge C \wedge W$. Detemine under what circumstances the wff P is satisfiable.

3. Write down the wff that describe the following finite state acceptor.



Show how your wff leads to the correct processing of the input string ab.

1.6 Adequate sets of connectives

We defined our version of PL using the following six connectives

$$\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus.$$

This is not a unique choice: for example, many books on logic do not include $\oplus$. In this section, we shall explore what is actually needed to define PL.

It is easy to show that

$$p \oplus q \equiv \neg(p \leftrightarrow q).$$

We have already proved that

$$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$$

and also that

$$p \rightarrow q \equiv \neg p \vee q.$$

We have therefore proved the following.

Proposition 1.6.1. *Every wff in PL is logically equivalent to one that uses only the logical connectives $\neg$, $\vee$ and $\wedge$.*

Although it would be more efficient to use only the above three logical connectives, it would also be less user-friendly. Significantly, however, those books that do not take $\oplus$ as a basic logical connective are not sacrificing any expressive power.

At this point, we introduce some terminology. We say that a set of logical connectives is *adequate* if every wff is logically equivalent to a wff that uses only logical connectives from that set. In these terms, we proved above that the connectives $\neg$, $\vee$, $\wedge$ form an adequate set.

We can, if we want, be even more miserly in the number of logical connectives we use. The following two logical equivalences can be proved using double negation and De Morgan.

- $p \vee q \equiv \neg(\neg p \wedge \neg q)$.
- $p \wedge q \equiv \neg(\neg p \vee \neg q)$.

From these we can deduce the following.

Proposition 1.6.2.

1. *The connectives $\neg$ and $\wedge$ together form an adequate set.*
2. *The connectives $\neg$ and $\vee$ together form an adequate set.*

Example 1.6.3. We show that the following wff are equivalent to wff using only the connectives $\neg$ and $\wedge$.

1. $p \vee q \equiv \neg(\neg p \wedge \neg q)$.
2. $p \to q \equiv \neg p \vee q \equiv \neg(\neg\neg p \wedge \neg q) \equiv \neg(p \wedge \neg q)$.
3. $p \leftrightarrow q \equiv (p \to q) \wedge (q \to p) \equiv \neg(p \wedge \neg q) \wedge \neg(q \wedge \neg p)$.

At this point, you might wonder if we can go one better. Indeed, we can but we have to define some new binary connectives. Define

$$p \downarrow q = \neg(p \vee q)$$

called *nor*. Define

$$p \mid q = \neg(p \wedge q)$$

called *nand*.

Proposition 1.6.4. *The binary connectives $\downarrow$ and $\mid$ on their own are adequate.*

Proof. We prove first that $\downarrow$ is adequate on its own. Observe that

$$\neg p \equiv \neg(p \vee p) \equiv p \downarrow p$$

and

$$p \wedge q \equiv \neg\neg p \wedge \neg\neg q \equiv \neg(\neg p \vee \neg q) \equiv (\neg p) \downarrow (\neg q) \equiv (p \downarrow p) \downarrow (q \downarrow q).$$

But since $\neg$ and $\wedge$ form an adequate set of connectives we can now construct everything using $\downarrow$ alone.

We now prove that $\mid$ is adequate on its own. Observe that

$$\neg p \equiv \neg(p \wedge p) \equiv p \mid p$$

and

$$p \vee q \equiv \neg\neg p \vee \neg\neg q \equiv \neg(\neg p \wedge \neg q) \equiv (\neg p) \mid (\neg q) \equiv (p \mid p) \mid (q \mid q).$$

□

It would be possible to develop the whole of PL using for example just **nor**, and some mathematicians have done just that, but this renders PL truly non-user-friendly.

Example 1.6.5. We find a wff logically equivalent to $p \to q$ that uses only nors.

$$\begin{aligned} p \to q &\equiv \neg p \vee q \\ &\equiv \neg\neg(\neg p \vee q) \\ &\equiv \neg(\neg(\neg p \vee q)) \\ &\equiv \neg(\neg p \downarrow q) \\ &\equiv \neg((p \downarrow p) \downarrow q) \\ &\equiv ((p \downarrow p) \downarrow q) \downarrow ((p \downarrow p) \downarrow q.) \end{aligned}$$

Exercises 1.6

---

1. Write each of the logical connectives $\neg, \wedge, \vee, \to, \leftrightarrow, \oplus$ in terms of **nand** only. Now, repeat this exercise using **nor** only.

2. (a) Show that $\neg$ and $\to$ together form an adequate set of connectives.

   (b) Show that $\to$ and $\boldsymbol{f}$ together form an adequate set of connectives. You should interpret $\boldsymbol{f}$ as being any contradiction.

3. Let $*$ be a binary connective that is supposed to be adequate on its own. There are 16 possible truth tables for $*$.

   (a) Explain why $T * T = F$.

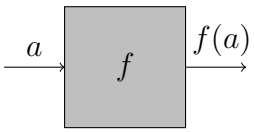
   (b) Explain why $F * F = T$.

   (c) There are now four possible ways to complete the truth table for $*$. By examining what they are, deduce that the only possible values for $*$ are $\downarrow$ and $|$.

1.7 Truth functions

We now have the ideas required to prove a result significant in circuit design. It will also tell us that PL is 'functionally complete', a phrase explained later in this section. But before I can do this, I need to introduce a couple of notions from mathematics.

Let A and B be sets. A *function* from A to B is a rule f that determines for each element of A a uniquely specified element of B. We often write $f\colon A \to B$ as a way of expressing the fact that f is a function from A to B where the set A, called the *domain*, is the set of allowable inputs, and the set B, called the *codomain*, is the set that contains all the outputs (and possibly more besides). If $a \in A$ then the *value* of f at a is written $f(a)$.



Functions are the bread and butter of mathematics and the dollars and cents of computer science — it is sometimes useful to think of a programme as computing a function that transforms input to outputs. However, in this section we shall only consider a very simple class of functions and to define those we need some further definitions.

In set notation $\{a, b\} = \{b, a\}$ since the order in which we list the elements of the set is immaterial. But there are lots of situations where we *do* want order to matter. The notation to do this is different. An *ordered pair* (a, b) is a pair of elements a and b where a is the *first element* and b is the *second element*. Thus, in general, $(a, b) \neq (b, a)$ unless $a = b$, of course. More generally, we can define *ordered triples* (a, b, c) and *ordered* 4*-tuples* (a, b, c, d) and, more generally, *ordered* n*-tuples* $(a_1, \ldots, a_n)$. We can use these ideas to define a way of combining sets. Let A and B be sets. Then define the set

$$A \times B = \{(a, b)\colon a \in A, b \in B\}.$$

This is called the *product of* A *and* B. More generally, if $A_1, \ldots, A_n$ are sets then the set set

$$A_1 \times \ldots \times A_n = \{(a_1, \ldots, a_n)\colon a_i \in A_i\}.$$

If $A = A_1 = \ldots = A_n$ then we write A^n for the set of all n-tuples whose elements are taken from A.

We can now make the key definition of this section. Put $\mathbb{T} = \{T, F\}$. *Any* function $f\colon \mathbb{T}^n \to \mathbb{T}$ is called a *truth function*. Because the domain of this function is $\mathbb{T}^n$ we say that it has n *arguments*; this simply means that the function has n inputs. Although I have used mathematical terms to define truth functions, what they are in practice is easy to explain. The set $\mathbb{T}^n$ is essentially the set of all possible lists of n truth values. As we have seen, there are 2^n of those. For each such list of ordered truth values, the function f assigns a single truth value. Thus each truth function determines a truth table and, conversely, each truth table defines a truth function. However, when I say 'truth table' I am not assuming that there is a wff that has that truth table. But the essence of the main theorem of this section is, that in fact, there is.

Example 1.7.1. Here is an example of a truth function $f\colon \mathbb{T}^3 \to \mathbb{T}$.

T	T	T	T
T	T	F	F
T	F	T	F
T	F	F	T
F	T	T	T
F	T	F	F
F	F	T	F
F	F	F	F

For example, $f(T, F, F) = T$ and $f(F, F, F) = F$.

Theorem 1.7.2 (Functional completeness)**.** *For each truth function with n arguments there is a wff with n atoms whose truth table is the given truth function.*

Proof. We shall prove this result in three steps constructing a wff A.

Step 1. Suppose that the truth function always outputs F. Define

$$A = (p_1 \wedge \neg p_1) \wedge \ldots \wedge p_n.$$

Then A has a truth table with 2^n rows that always outputs F.

Step 2. Suppose that the truth function outputs T *exactly once*. Let $v_1, \ldots, v_n$, where $v_i = T$ or F, be the assignment of truth values which yields the output T. Define a wff A as follows. It is a conjunction of exactly one of p_1 or $\neg p_1$, of p_2 or $\neg p_2$, ..., of p_n or $\neg p_n$ where p_i is chosen if $v_i = T$ and

$\neg p_i$ is chosen if $v_i = F$. I shall call A a *conjunctive clause* corresponding to the pattern of truth values $v_1, \ldots, v_n$. The truth table of A is the given truth function.

Step 3. Suppose that we are given now an arbitrary truth function not covered in steps 1 and 2 above. We construct a wff A whose truth table is the given truth function by taking a disjunction of all the conjunctive clauses constructed from each row of the truth function that outputs T. □

The proof of the above theorem is best explained by means of an example.

Example 1.7.3. We construct a wff that has as truth table the following truth function.

T	T	T	$\boldsymbol{T}\,(1)$
T	T	F	F
T	F	T	F
T	F	F	$\boldsymbol{T}\,(2)$
F	T	T	$\boldsymbol{T}\,(3)$
F	T	F	F
F	F	T	F
F	F	F	F

We need only consider the rows that output T, which I have highlighted. I have also included a reference number that I shall use below. The conjunctive clause corresponding to row (1) is

$$p \wedge q \wedge r.$$

The conjunctive clause corresponding to row (2) is

$$p \wedge \neg q \wedge \neg r.$$

The conjunctive clause corresponding to row (3) is

$$\neg p \wedge q \wedge r.$$

The disjunction of these conjunctive clauses is

$$A = (p \wedge q \wedge r) \vee (p \wedge \neg q \wedge \neg r) \vee (\neg p \wedge q \wedge r).$$

You should check that the truth table of A is the truth function given above.

Exercises 1.7

1. The following truth table gives the behaviour of three truth functions (a), (b) and (c). In each case, find a wff whose truth table is equal to the corresponding truth function.

p	q	r	(a)	(b)	(c)
T	T	T	F	F	T
T	T	F	F	T	F
T	F	T	T	F	T
T	F	F	F	T	F
F	T	T	F	F	T
F	T	F	F	F	F
F	F	T	F	F	T
F	F	F	F	F	F

1.8 Normal forms

I could, if I were feeling mischievous, write $\neg\neg\neg\neg\neg p$ instead of $\neg p$. Writing the former rather than the latter would not be normal, but neither would it be disallowed. However, good communication depends on agreements on how we do things. This section is about particular ways of writing wff called *normal forms*.

1.8.1 Negation normal form (NNF)

A wff is in *negation normal form (NNF)* if it is constructed using only $\wedge$, $\vee$ and literals. Recall that a *literal* is either an atom or the negation of an atom.

Proposition 1.8.1. *Every wff is logically equivalent to a wff in NNF.*

Proof. Let A be a wff in PL. First, replace any occurrences of $x \oplus y$ by $\neg(x \leftrightarrow y)$. Second, replace any occurrences of $x \leftrightarrow y$ by $(x \to y) \wedge (y \to x)$. Third, replace all occurrences of $x \to y$ by $\neg x \vee y$. Fourth, use De Morgan's laws to push all occurrences of negation through brackets. Finally, use double negation to ensure that only literals occur. □

Example 1.8.2. We convert $\neg(p \to (p \wedge q))$ into NNF using the method outlined in the proof of the above result.

$$\begin{aligned} \neg(p \to (p \wedge q)) &\equiv \neg(\neg p \vee (p \wedge q)) \\ &\equiv \neg\neg p \wedge \neg(p \wedge q) \\ &\equiv \neg\neg p \wedge (\neg p \vee \neg q) \\ &\equiv p \wedge (\neg p \vee \neg q). \end{aligned}$$

1.8.2 Disjunctive normal form (DNF)

We now come to the first of the two important normal forms. A wff that can be written as a disjunction of one of more terms each of which is a conjunction of one or more literals is said to be in *disjunctive normal form (DNF)*. Thus a wff in DNF has the following schematic shape

$$(\wedge \text{ literals}) \vee \ldots \vee (\wedge \text{ literals}).$$

Examples 1.8.3. Some special cases of DNF are worth highlighting because they often cause confusion.

1. A single atom p is in DNF.
2. A term such as $(p \wedge q \wedge \neg r)$ is in DNF.
3. The expression $p \vee q$ is in DNF. You should think of it as $(p) \vee (q)$.

Proposition 1.8.4. *Every wff is logically equivalent to one in DNF.*

Proof. Let A be a wff. Construct the truth table for A. Now apply Theorem 1.7.2. The wff that results is in DNF and logically equivalent to A. □

The proof above can be also be used as a method for constructing DNF though it is a little laborious. Another method is to use logical equivalences. Let A be a wff. First convert A to NNF and then if necessary use the distributive laws to convert to a wff which is in DNF.

Example 1.8.5. We show how to convert $\neg(p \to (p \wedge q))$ into DNF using a sequence of logical equivalences. The first step is to replace $\to$. We use the fact that $x \to y \equiv \neg x \vee y$. This gives us $\neg(\neg p \vee (p \wedge q))$. Now use De Morgan's

laws to push negation inside the brackets. This yields $\neg\neg p \wedge \neg(p \wedge q)$ and then $\neg\neg p \wedge (\neg p \vee \neg q)$. We now apply double negation to get $p \wedge (\neg p \vee \neg q)$. This is in NNF. Finally, we apply one of the distributive laws to get the $\vee$ out of the brackets. This yields $(p \wedge \neg p) \vee (p \wedge \neg q)$. This wff is in DNF and so

$$\neg(p \to (p \wedge q)) \equiv (p \wedge \neg p) \vee (p \wedge \neg q).$$

1.8.3 Conjunctive normal form (CNF)

A wff is in *conjunctive normal form (CNF)* if it is a conjunction of one or more terms each of which is a disjunction of one or more literals. It therefore looks a bit like the reverse of (DNF).

$$(\vee \text{ literals}) \wedge \ldots \wedge (\vee \text{ literals}).$$

Proposition 1.8.6. *Every wff is logically equivalent to one in CNF.*

Proof. Let A be our wff. Write $\neg A$ in DNF by Proposition 1.8.4. Now negate both sides of the logical equivalence and use double negation where necessary to obtain a wff in CNF. □

Example 1.8.7. A wff A has the following truth table.

T	T	T	T
T	T	F	T
T	F	T	F
T	F	F	T
F	T	T	T
F	T	F	F
F	F	T	T
F	F	F	T

The truth table for $\neg A$ is

T	T	T	F
T	T	F	F
T	F	T	T
T	F	F	F
F	T	T	F
F	T	F	T
F	F	T	F
F	F	F	F

It follows that

$$\neg A \equiv (p \wedge \neg q \wedge r) \vee (\neg p \wedge q \wedge \neg r)$$

is the DNF for $\neg A$. Negating both sides we get

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

This is the CNF for A.

Sometimes using logical equivalences is more efficient.

Example 1.8.8. The wff $(\neg x \wedge \neg y) \vee (\neg x \wedge z)$ is in DNF. It can easily be converted into CNF using one of the distributivity laws.

$$\begin{aligned}(\neg x \wedge \neg y) \vee (\neg x \wedge z) &\equiv ((\neg x \wedge \neg y) \vee \neg x) \wedge ((\neg x \wedge \neg y) \vee z) \\ &\equiv (\neg x \vee \neg x) \wedge (\neg y \vee \neg x) \wedge (\neg x \vee \neg z) \wedge (\neg y \wedge z).\end{aligned}$$

1.8.4 Horn formulae

Computer programs are written using artificial or formal languages called *programming languages*. There are two big classes of such languages: *imperative* and *declarative*. In an imperative language, you have to specify in detail how the solution to a problem is to be accomplished. In other words, you have to tell the computer *how to* solve the problem. In a declarative language, you tell the computer *what* you want to solve and leave the details of how to accomplish this to the software. Declarative languages sound like magic but some of them, such as PROLOG, are based on ideas drawn from first-order logic. This language has applications in AI (artificial intelligence). We describe here the wff in PL that are important in PROLOG.

A wff in CNF is called a *Horn formula*[12] if each group of disjunctions contains at most one positive literal. This seems a little arbitrary but looks more natural when we use implication. Here are some examples.

1. $\neg p \vee q \equiv p \to q$.
2. $\neg p \vee \neg q \vee r \equiv p \wedge q \to r$.
3. The wff $\neg p \vee \neg q$ is at first glance more puzzling but if we introduce the wff $\boldsymbol{f}$ to mean a statement that is always false, we have the following

$$\neg p \vee \neg q \equiv \neg p \vee \neg q \vee f \equiv p \wedge q \to \boldsymbol{f}.$$

[12] Named after the American mathematician Alfred Horn (1918–2001).

4. Letting $\boldsymbol{t}$ be a wff that is always true, we have that

$$p \equiv f \vee p \equiv \neg t \vee p \equiv \boldsymbol{t} \rightarrow p.$$

The above examples suggest that every Horn formula is locally equivalent to a conjunction of terms each of which has one of the following forms:

- $p_1 \wedge \ldots \wedge p_n \rightarrow p$. Wff of this type are called *rules*.
- $p_1 \wedge \ldots \wedge p_n \rightarrow \boldsymbol{f}$. Wff of this type are called *goals*.
- $\boldsymbol{t} \rightarrow p$. Wff of this type are called *facts*.

We call this the *implicational form* of the Horn formula.

Example 1.8.9. Horn formulae arise naturally as rules of various kinds. For example, what I can remember about my childhood consists of rules such as these:

- 'I must not run with scissors'.
- 'If it is cold then I must wear a thick coat'.
- 'If I eat my Brussels sprouts and I eat my toad-in-the-hole then I can eat my trifle'.

The state transitions of finite state automata, described in Section 1.5.2, are also described by Horn formulae since they are of the form 'current state and current input implies next state'.

Example 1.8.10. Consider the wff

$$H = (p \vee \neg s \vee \neg u) \wedge (q \vee \neg r) \wedge (\neg q \vee \neg s) \wedge s \wedge (\neg s \vee u).$$

This is a Horn formula and it can be written in its implicational form as follows.

$$H = (s \wedge u \rightarrow p) \wedge (r \rightarrow q) \wedge (q \wedge s \rightarrow \boldsymbol{f}) \wedge (\boldsymbol{t} \rightarrow s) \wedge (s \rightarrow u).$$

One of the reasons Horn formulae are interesting is that there is a fast method for deciding whether they are satisfiable or not; in the next section, we shall explain why this is not true for arbitrary wff.

Fast algorithm for Horn formula satisfiability

Let the Horn formula be $H = H_1 \wedge \ldots \wedge H_m$ where each H_i is either a rule, a goal or a fact. Let the atoms occurring in H be $p_1, \ldots, p_n$. The algorithm begins by initially assigning all the atoms the truth value F. If the algorithm terminates successfully, some of these might be changed to the truth value T and the wff H will be satisfied by this truth assignment. During the algorithm we shall *mark* atoms; this can be done by placing a dot above them, like so $\dot{p}$. The algorithm is as follows.

1. For all H_i of the form $\boldsymbol{t} \to p$ mark all occurrences of the atom p in H.
2. The following procedure is now repeated until it can now longer be applied
   - If there is an H_i of the form $p_1 \wedge \ldots \wedge p_n \to p$ where all of $p_1, \ldots, p_n$ have been marked and p has not been marked then mark every occurrence of p.
   - If there is an H_i of the form $p_1 \wedge \ldots \wedge p_n \to \boldsymbol{f}$ where all of $p_1, \ldots, p_n$ have been marked then terminate the algorithm and output that H is not satisfiable.
3. Output that H is satisfiable; all marked atoms should be assigned the truth value T with the remaining atoms being assigned the truth value F, and this results in a truth assignment that satisfies H.

Before we prove that the above algorithm really does what I claim it does, we give an example of it in operation.

Example 1.8.11. We apply the above algorithm to the Horn formula

$$H = (s \wedge u \to p) \wedge (r \to q) \wedge (q \wedge s \to \boldsymbol{f}) \wedge (\boldsymbol{t} \to s) \wedge (s \to u).$$

We apply part (1) of the algorithm and so mark all occurrences of the atom s.

$$H = (\dot{s} \wedge u \to p) \wedge (r \to q) \wedge (q \wedge \dot{s} \to \boldsymbol{f}) \wedge (\boldsymbol{t} \to \dot{s}) \wedge (\dot{s} \to u).$$

We now apply part (2) of the algorithm repeatedly. First, we mark all occurrences of the atom u.

$$H = (\dot{s} \wedge \dot{u} \to p) \wedge (r \to q) \wedge (q \wedge \dot{s} \to \boldsymbol{f}) \wedge (\boldsymbol{t} \to \dot{s}) \wedge (\dot{s} \to \dot{u}).$$

Second, we mark all occurrences of the atom p.

$$H = (\dot{s} \wedge \dot{u} \to \dot{p}) \wedge (r \to q) \wedge (q \wedge \dot{s} \to \boldsymbol{f}) \wedge (\boldsymbol{t} \to \dot{s}) \wedge (\dot{s} \to \dot{u}).$$

At this point, the algorithm terminates successfully. A satisfying truth assignment is therefore

p	q	r	s	u
T	F	F	T	T

If you run the truth table generator [48] on this example, you will find that this truth assignment is the first one that satisfies H starting from the bottom of the truth table although in this case it is in fact the only satisfying truth assignment.

Theorem 1.8.12. *The algorithm above for deciding whether a Horn formula is satisfiable works.*

Proof. Clearly, the algorithm will terminate one way or another. Assume that the algorithm terminates with output that H is satisfiable. We prove that the truth assignment constructed works. The wff H is a conjunction of wff H_i each of which must be satisfied. Each of the H_i has one of three forms.

- Any wff of the form $\boldsymbol{t} \to p$ is satisfied by step (1).
- Any wff of the form $p_1 \wedge \ldots \wedge p_n \to q$ is satisfied in one of two ways. If all the atoms $p_1, \ldots, p_n$ are assigned the truth value T then q must also be assigned the truth value T by step (2). Clearly, in this way $p_1 \wedge \ldots \wedge p_n \to q$ is satisfied. On the other hand, if not all the atoms $p_1, \ldots, p_n$ are assigned the truth value T then $p_1 \wedge \ldots \wedge p_n$ is false and so $p_1 \wedge \ldots \wedge p_n \to q$ is true irrespective of the truth value of q.

- Finally, because the algorithm terminates successfully, in any wff of the form $q_1 \wedge \ldots \wedge q_m \rightarrow \boldsymbol{f}$ at least one of the atoms $q_1, \ldots, q_m$ will be assigned the truth value F. It follows again that $q_1 \wedge \ldots \wedge q_m \rightarrow \boldsymbol{f}$ is satisfiable.

Assume now that the algorithm terminates with the output that H is unsatisfiable. Suppose that in fact H were satisfiable. Choose the first truth assignment that satisfies H working up from the bottom of the truth table for H. We call this the *smallest* satisfying truth assignment and denote it by τ. By assumption, one of the H_i must be of the form $q_1 \wedge \ldots \wedge q_m \rightarrow \boldsymbol{f}$ where the algorithm assigns the truth value T to all the atoms $q_1, \ldots, q_m$. However, τ must assign the truth value T to all atoms p where $\boldsymbol{t} \rightarrow p$ occurs in H and it must assign the value T to q if it assigns the value T to all the atoms $p_1, \ldots, p_n$ in the wff $p_1 \wedge \ldots \wedge p_n \rightarrow q$. Thus τ must assign the truth value T to all the atoms $q_1, \ldots, q_m$ but this is a contradiction. □

Exercises 1.8

---

1. Use known logical equivalences to transform each of the following wff first into NNF and then into DNF.

   (a) $(p \rightarrow q) \rightarrow p$.

   (b) $p \rightarrow (q \rightarrow p)$.

   (c) $(q \wedge \neg p) \rightarrow p$.

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

   (e) $p \rightarrow (q \wedge r)$.

   (f) $(p \vee q) \wedge (r \rightarrow s)$.

2. Use known logical equivalences to transform each of the following into CNF.

   (a) $(p \rightarrow q) \rightarrow p$.

   (b) $p \rightarrow (q \rightarrow p)$.

   (c) $(q \wedge \neg p) \rightarrow p$.

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

(e) $p \rightarrow (q \wedge r)$.

(f) $(p \vee q) \wedge (r \rightarrow s)$.

3. Let $A = ((p \wedge q) \rightarrow r) \wedge (\neg(p \wedge q) \rightarrow r)$.

   (a) Draw the truth table for A.

   (b) Construct DNF using (a).

   (c) Draw the truth table for $\neg A$.

   (d) Construct DNF for $\neg A$ using (c).

   (e) Construct CNF for A using (d).

4. Let $A = ((p \wedge q) \rightarrow r) \wedge (\neg(p \wedge q) \rightarrow r)$.

   (a) Write A in NNF.

   (b) Use known logical equivalences applied to (a) to get DNF.

   (c) Use logical equivalences applied to (b) to get CNF.

   (d) Simplify A as much as possible using known logical equivalences.

5. Write $p \leftrightarrow (q \leftrightarrow r)$ in NNF.

6. For each of the following Horn formulae, write them in implicational form and then use the algorithm described in the text to determine whether they are satisfiable or not. In each case, check your answer using the truth table generator [48].

   (a) $(p \vee \neg q) \wedge (q \vee \neg r)$.

   (b) $(p \vee \neg q \vee \neg r) \wedge (\neg s \vee \neg u) \wedge (\neg p \vee \neg q \vee r) \wedge (p) \wedge (q)$.

   (c) $(p) \wedge (q) \wedge (\neg p \vee \neg q) \wedge (r \vee \neg p)$.

1.9 P = NP? (Or how to win a million dollars)

In this course, I cannot go in to the details of this question but I can at least sketch out what it means and why it is important.

The question whether **P** is equal to **NP** is the first of the seven *Millennium problems* that were posed by the Clay Mathematics Institute in 2000. Anyone who solves one of these problems wins a million dollars. So far, only one of these problems has been solved: namely, the *Poincaré Conjecture* by Grigori Perelman who turned down the prize money. The other six problems require advanced mathematics just to understand what they are saying — except one. This is the question of whether **P** is equal to **NP**. I shall begin this sketch by explaining in intuitive terms what we mean by **P** and **NP**.

How long does it take a program to solve a problem? As it stands, this is too vague to admit an answer so we need to make it more precise. For concreteness, imagine a program that takes as input a whole number n and produces as output either the result that 'n is prime' or 'n is not prime'. So, if you input 10 to the program it would tell you it was not prime but if you input 17 it would tell you that it was prime. Clearly, how long the program takes to solve this question depends on how big the number is that you input. A number with hundreds of digits is clearly going to take a lot longer for the program to work than one with just a few digits.

> *Thus to say how long a program takes to solve a problem must refer to the length of the input to that program.*

Now, for all inputs of a fixed length, say m, the program might take different amounts of time to produce an output depending on which input of length m was chosen.

> *We agree to take the longest amount of time over all inputs of length m as a measure of how long it takes to process any input of length m.*

This begs the question of what we mean by 'time'. Your fancy MacBook Pro may be a lot faster than my Babbage Imperial. So instead of time, we count the number of basic computational steps needed to transform input to output. It turns out that we don't really have to worry too much about what this means but it can be made precise using Turing machines.

Thus with each program we can try to calculate its *time complexity profile*. This will be a function $m \mapsto f(m)$ where m is the length of the input and $f(m)$ is the maximum number of steps needed to transform any input of size m into an output. Calculating the time complexity profile exactly of even

simple programs requires a lot of mathematical analysis, but fortunately we don't need an exact answer merely a good estimate. A program that has the time complexity profile $m \mapsto am$, where a is a number, is said to run in *linear time*, if the time complexity profile is $m \mapsto am^2$ it is said to run in *quadratic time*, if the time complexity profile is $m \mapsto am^3$ it is said to run in *cubic time*. More generally, if the time complexity profile is $m \mapsto am^n$ for some n it is said to run in *polynomial time*. All the basic algorithms you learnt at school for adding, subtracting, multiplying and dividing numbers run in polynomial time.

> *We define the class* **P** *to be all those problems that can be solved in polynomial time.*

These are essentially nice problems with nice (meaning fast) programs to solve them. Let me stress that just because you have an algorithm for a problem that is slower than polynomial time does not mean your problem does not have a polynomial time algorithm. If you claim this then you have to prove it. What would constitute a nasty problem? This would be one whose time complexity profile looked like $m \mapsto 2^m$. This is nasty because just by increasing the size of the input by 1 doubles the amount of time needed to solve the problem. We now isolate those nasty problems that have a nice feature: that is, that any purported solution can be *checked* quickly: that is, checked in polynomial time. A really nasty problem would be one where it is even difficult just to check a purported solution.

> *We define the class* **NP***, of non-deterministic polynomial time problems, to be all those problems whose solutions can be checked in polynomial time.*

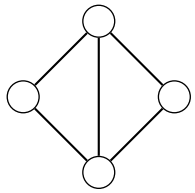
Clearly, **P** is contained in **NP** but there is no earthly reason why they should be equal. Now here is the rub: currently (2017) no one has been able to prove that they are not equal. This explains what, in essence, the question whether **P** is equal to **NP** means. It does not, however, explain the significance of this question. To understand this, we must delve deeper.

In 1971, Stephen Cook came up with an idea for resolving this question that shed new light on the nature of algorithms. His idea was that inside **NP** there should be a nastiest problem with an important, and ironically nice, property: namely, if that problem could be shown to be in **P** then everything

in **NP** would also have to be in **P** thus showing that **P** = **NP**. On the other hand, if it could be shown that this problem wasn't in **P** then we would have shown that **P** $\neq$ **NP**. Thus Cook's problem would provide a sort of litmus-test for equality. Such a problem is called **NP**-*complete*.

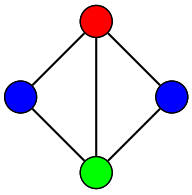
Given that we don't even know all the problems in **NP**, Cook's idea might have sounded simply too good to be true. But in fact, he was able to prove a specific problem to be **NP**-complete: that problem is SAT — the *satisfiability problem in PL*. This probably sounds mysterious but in fact once you start studying complexity theory the mystery disappears. The reason that it is possible to prove that SAT is **NP**-complete boils down to the fact that PL is a sufficiently rich language to describe the behaviour of computers. A special case of this was discussed in Section 1.5.2.

Cook's result, known as *Cook's theorem*, is remarkable enough and explains the central importance of the satisfiability problem in theoretical computers science. But there is more. Thousands of interesting problems that we would like to solve quickly have been shown to be **NP**-complete and so equivalent to SAT[13]. The *travelling salesman problem* is one well known example. I shall describe another interesting one here: *is a graph k-colourable?* A *graph* consists of *vertices* which are represented by circles and *edges* which are lines joining the circles (here always joining different circles). Vertices joined by an edge are said to be *adjacent*. The following is an example of a graph.



By a *colouring* of a graph we mean an assignment of colours to the vertices so that adjacent vertices have different colours. By a *k-colouring* of a graph we mean a colouring that uses at most k colours. Depending on the graph and k that may or may not be possible. The following is a 3-colouring of the above graph.

[13]The standard reference to all this is [15].



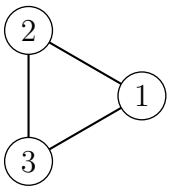
However, it is not possible to find a 2-colouring of this graph because there is a triangle of vertices. The crucial point is this: if you do claim to have found a k-colouring of a graph I can easily check whether you are right. Thus the problem of k-colouring a graph is in **NP**. On the other hand, finding a k-colouring can involve a lot of work, including much back-tracking. It can be proved that this problem is **NP**-complete and so equivalent to SAT.

This is not quite the end of the story. We don't know whether **P** is equal to **NP** but there are special cases where we know there are fast algorithms. For example, we saw in Section 1.8.3 that there is a fast algorithm for deciding whether a Horn formula is satisfiable or not. The point is that in the context of particular problem the wff that arise may have extra properties and these might be susceptible to a fast method for determining satisfiability. In this case, whether **P** is equal to **NP** may actually be irrelevant.

Exercises 1.9

---

1. The picture below shows a graph.



   The vertices are to be coloured either blue or red under certain constraints that will be described by means of a wff in PL denoted by A. The following is a list of atoms and what they mean.

   - p: vertex 1 is blue.
   - q: vertex 1 is red.

- r: vertex 2 is blue.
- s: vertex 2 is red.
- u: vertex 3 is blue.
- v: vertex 3 is red.

Here is the wff A constructed from these atoms.

$$(p \oplus q) \wedge (r \oplus s) \wedge (u \oplus v) \wedge$$
$$(q \rightarrow (r \wedge u)) \wedge (p \rightarrow (s \wedge v)) \wedge$$
$$(s \rightarrow (p \wedge u)) \wedge (r \rightarrow (q \wedge v)) \wedge$$
$$(v \rightarrow (p \wedge r)) \wedge (u \rightarrow (q \wedge s))$$

(a) Translate A into English in as pithy and precise a way as possible.

(b) Use [48] to construct a truth table of A.

(c) Interpret this truth table.

1.10 Valid arguments

> *An argument is a connected series of statements intended to establish a proposition.* No it isn't. *Yes it is! It's not just contradiction.* Look, if I argue with you, I must take up a contrary position. *Yes, but that's not just saying 'No it isn't.'* Yes it is! *No it isn't!* Yes it is! *Argument is an intellectual process. Contradiction is just the automatic gainsaying of any statement the other person makes. (Short pause)* No it isn't. — Monty Python.

We have so far viewed PL as a low-level description language. This is its significance in the question as to whether **P** equals **NP** or not. We shall now describe the other important application of PL which was actually the reason why PL was introduced in the first place. This is PL as a language for describing correct reasoning.

1.10.1 Definitions and examples

We begin with some examples.

Example 1.10.1. Someone makes the following announcement: 'In the pub I will either drink cider or I will drink apple juice. But I won't drink cider.' What do you deduce? You deduce that they will drink apple juice.

Example 1.10.2. The newspaper headline proclaims 'Either Smith or Jones will be Prime Minister. If Smith wins we are doomed. If Jones wins we are doomed.' What do you deduce? Clearly, with a sinking heart, you deduce that we are doomed.

Example 1.10.3. If it is raining then I get wet. I am not wet. I deduce that it is not raining.

None of these examples is at all exceptional. I would guess that in every case, you had no trouble seeing that what we deduce follows from the preceding statements. But what do we mean by 'follows from'? This seems like something that would be impossible to define, like good art. In fact, we can use the logic we have defined so far to explain why, in each case, it makes sense to say 'follows from'.

Let $A_1, \ldots, A_n$ be wff which we regard as *assumptions* or *premisses*.[14] Let B be a wff which we would like to regard as a *consequence* of $A_1, \ldots, A_n$. That is, we would like to say that B *follows from* $A_1, \ldots, A_n$. We can express this relationship between some assumptions and their conclusion using PL as follows.

> We require that whenever all of $A_1, \ldots, A_n$ are true then B must be true as well. Equivalently, it is impossible for $A_1, \ldots, A_n$ to be true and B to be false. If B is a consequence of $A_1, \ldots, A_n$ we write
>
> $$A_1, \ldots, A_n \vDash B$$
>
> and we say this is a *valid argument*.

You should go back and check that each of our three examples is a valid argument in this sense. This definition encapsulates many important examples of logical reasoning but we shall see later that there are also examples of logical reasoning that cannot be captured by PL. It is this that will lead us to the generalization of PL called *first-order logic*, or FOL.

[14] At this point, I am obliged to tell the story about the feuding neighbours who could never agree because they were arguing from different premises. This is a variant of a *bon mot* attributed to the Reverend Sydney Smith, amongst others.

Examples 1.10.4. Here are some examples of valid arguments.

1. $p, p \to q \vDash q$. We show that this is a valid argument. Here is the truth table for $p \to q$.

p	q	$p \to q$
T	T	T
T	F	F
F	T	T
F	F	T

   We are only interested in the cases where both p and $p \to q$ are true.

p	q	$p \to q$
$\boldsymbol{T}$	T	$\boldsymbol{T}$

   We see that if p and $p \to q$ are true then q must be true.

2. $p \to q, \neg q \vDash \neg p$. We show this is a valid argument.

p	q	$\neg p$	$\neg q$	$p \to q$
T	T	F	F	T
T	F	F	T	F
F	T	T	F	T
F	F	T	T	T

   We are only interested in the cases where both $p \to q$ and $\neg q$ are true.

p	q	$\neg p$	$\neg q$	$p \to q$
F	F	T	**T**	**T**

   We see that if $p \to q$ and $\neg q$ are true then $\neg p$ must be true.

3. $p \vee q, \neg p \vDash q$. We show this is a valid argument.

p	q	$\neg p$	$p \vee q$
T	T	F	T
T	F	F	T
F	T	T	T
F	F	T	F

We are only interested in the cases where both $p \vee q$ and $\neg p$ are true.

p	q	$\neg p$	$p \vee q$
F	T	$\boldsymbol{T}$	$\boldsymbol{T}$

We see that if $p \vee q$ and $\neg p$ are true then q must be true.

4. $p \to q, q \to r \vDash p \to r$. We show this is a valid argument.

p	q	r	$p \to q$	$q \to r$	$p \to r$
T	T	T	T	T	T
T	T	F	T	F	F
T	F	T	F	T	T
T	F	F	F	T	F
F	T	T	T	T	T
F	T	F	T	F	T
F	F	T	T	T	T
F	F	F	T	T	T

We are only interested in the cases where both $p \to q$ and $q \to r$ are true.

p	q	r	$p \to q$	$q \to r$	$p \to r$
T	T	T	T	T	T
F	T	T	T	T	T
F	F	T	T	T	T
F	F	F	T	T	T

We see that in every case $p \to r$ is true. Thus the argument is valid.

We may always use truth tables in the way above to decide whether an argument is valid or not but it is quite laborious. The following result, however, enables us to do this in a very straightforward way.

Proposition 1.10.5. *The argument*

$$A_1, \ldots, A_n \vDash B$$

is valid precisely when

$$\vDash (A_1 \wedge \ldots \wedge A_n) \to B.$$

Proof. The result is proved in two steps.

1. We prove that $A_1, \ldots, A_n \vDash B$ precisely when $A_1 \wedge \ldots \wedge A_n \vDash B$. This means that we can restrict to the case of exactly one wff on the left hand side of the semantic turnstile. Suppose that $A_1, \ldots, A_n \vDash B$ is a valid argument and that it is not the case that $A_1 \wedge \ldots \wedge A_n \vDash B$ is a valid argument. Then there is some assignment of truth values to the atoms that makes $A_1 \wedge \ldots \wedge A_n$ true and B false. But this means that each of $A_1, \ldots, A_n$ is true and B is false that contradicts that we are given that $A_1, \ldots, A_n \vDash B$ is a valid argument. Suppose that $A_1 \wedge \ldots \wedge A_n \vDash B$ is a valid argument but that $A_1, \ldots, A_n \vDash B$ is not a valid argument. Then some assignment of truth values to the atoms makes $A_1, \ldots, A_n$ true and B false. This means that $A_1 \wedge \ldots \wedge A_n$ is true and B is false. But this contradicts that we are given that $A_1 \wedge \ldots \wedge A_n \vDash B$ is a valid argument.

2. We prove that $A \vDash B$ precisely when $\vDash A \rightarrow B$. Suppose that $A \vDash B$ is a valid argument and that $A \rightarrow B$ is not a tautology. Then there is some assignment of the truth values to the atoms that makes A true and B false. But this contradicts that $A \vDash B$ is a valid argument. Suppose that $A \rightarrow B$ is a tautology and $A \vDash B$ is not a valid argument. Then there is some assignment of truth values to the atoms that makes A true and B false. But then the truth values of $A \rightarrow B$ is false from the way implication is defined and this is a contradiction.

□

Example 1.10.6. We show that $p, p \rightarrow q \vDash q$ is a valid argument by showing that $\vDash (p \wedge (p \rightarrow q)) \rightarrow q$ is a tautology.

p	q	$p \rightarrow q$	$p \wedge (p \rightarrow q)$	$(p \wedge (p \rightarrow q)) \rightarrow q$
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

1.10.2 *The game of mathematics*

The idea of proof arose first in classical Greece. Proofs are the essence of mathematics. The notion of proof is so powerful that it has been borrowed by

computer scientists. To conclude this section, I shall give a few of examples of mathematical proofs where the ideas of this section can be clearly seen. For more background information on proofs, see the first two chapters of [28], and for a more thorough treatment of how logic is used in mathematics see [11].

Back in the Introductory Exercises, I included a question about Sudoku. This may have seemed odd but in Section 1.5 and Section 1.9, I explained that these puzzles are prototypes for a wide class of problems which are known as NP-complete. I now want to look at Sudoku from another perspective, that is as a toy model for how mathematics works. This will lead us to a more 'dynamic' way of thinking about logic which has the idea of an *argument* at its core.

When you solve a Sudoku puzzle you work as follows. First, you study the initial information you have been given; that is, the squares that have already been filled in with numbers. To solve the puzzle, you have to accept these as a given. Next, you use reason to try to fill in other squares in accordance with the rules of the game. Sometimes, filling a square with a number leads to a conflict, or as we would say in mathematics a *contradiction*, so that you know your choice of number was incorrect. This process is 'dynamic' rather than the 'static' approach adopted in Section 1.5 via the satisfiability problem.

My description for how a Sudoku puzzle is solved is a good metaphor for how mathematics itself works. Each domain of mathematics is characterized by its basic assumptions or *axioms*. These are akin to the initial information given in a Sudoku puzzle. Next, by using reason and appealing to previously proved results you attempt to prove new results. Logic is the very grammar of reason. But just as a knowledge of Russian grammar won't make you a Tolstoy, so too a knowledge of logic won't make you a Gauss. But if you want to do mathematics, you have to acquire an understanding of logic, just as if you want to write you have to acquire an understanding of grammar.

We conclude with a few examples from mathematics.

Example 1.10.7. How would you prove a statement of the form $p \to q$? We know that the only case where this statement is false is where p is true and q is false. So, it is enough to assume that p is true and, on the basis of that assumption (and any mathematical truths that are useful), prove that q has to be true. In mathematics, we are always working with arguments of the form $\Gamma, p \vDash q$ where Γ is the set of all mathematical truths relevant to the

mathematical domain in question and any axioms that need to be assumed to work in that domain.

Example 1.10.8. Suppose we are trying to prove that the statement p is true. One way to do this is to cosntruct a valid argument of the form $\Gamma \models \neg p \rightarrow \boldsymbol{f}$ where $\boldsymbol{f}$ is any contradiction. But the only way for the statement $\neg p \rightarrow \boldsymbol{f}$ to be true is if $\neg p$ is false which implies that p is true. This is called *proof by contradiction.*

Example 1.10.9. Let n be a natural number. So, $n = 0, 1, 2, \ldots$. We say it is *even* if it is exactly divisible by two. We say that n is *odd* if '$\neg$ (n is even)'. This definition is not very useful and a better, but equivalent, one is to say that n is odd if the reaminder 1 is left when n is divided by 2. I prove first that the square of an even number is even and the square of an odd number is odd. Let n be even. Then by definition, we can write $n = 2m$. Squaring both sides we have that $n^2 = 4m^2 = 2(2m^2)$. It follows that n^2 is even. Now, let n be odd. By definition, we can write $n = 2m + 1$. Squaring both sides we have that $n^2 = 4m^2 + 4m + 1 = 2(2m^2 + 2m) + 1$. It follows that n^2 is odd. So, far, so good.

Now we turn to a different but related question. Suppose I tell you that n^2 is even. What can you deduce about n? Hopefully, you can see, intuitively, that n is even, but we can use what we have learnt in this section to prove it. Let p be the statement 'n is odd'. Let q be the statement 'n^2 is odd'. Thus $\neg q$ is the statement that 'n^2 is even' and $\neg p$ is the statement that 'n is even'. The following argument is valid

$$p \rightarrow q, \neg q \models \neg p.$$

Thus the following argument is valid

n odd implies n^2 is odd, n^2 is even therefore n is even.

But we have proved that 'n odd implies n^2 is odd' is always true and I have told you that n^2 is even. It follows that n is even.

We have therefore prove that if the square of a natural number is even then that natural number is also even.

The above example is very simple but has a profound consequence as we shall see in the next example.

Example 1.10.10. Define a real number to be *rational* if it can be written as a fraction $\frac{a}{b}$. A real number that is *not* rational is called *irrational*. Observe that the word 'irrational' is really 'in-rational' where 'in' just means 'not'.[15] Rational numbers are 'graspable' or 'intelligible' since they can be described by two natural numbers; this is why they are 'rational' (the more usual meaning of that word). Irrational numbers are just the opposite. I shall prove that $\sqrt{2}$ is irrational; I shall explain why this is interesting afterwards.

Let p be the statement '$\sqrt{2}$ is rational'. Let q be the statement 'a and b are both even'. I shall first prove the statement $p \to q$. To do this, it is enough to assume, by Example 1.10.7, that p is true and then deduce that q is true.

1. Assume p is true. That is, assume that $\sqrt{2} = \frac{a}{b}$ for some natural numbers a and b.

2. Square both sides of the equation $\sqrt{2} = \frac{a}{b}$ to obtain $2 = \frac{a^2}{b^2}$. Then rearrange to obtain $2b^2 = a^2$. I can do this by using previously proved results in mathematics.

3. a^2 is even by definition.

4. a is even by the result we proved in Example 1.10.10.

5. $a = 2x$ for some natural number x by definition of the word 'even'.

6. It follows that $b^2 = 2x^2$ using previously proved results in mathematics.

7. b^2 is even by definition.

8. b is even by the result we proved in Example 1.10.10.

9. Since a and b both even it follows that the statement q is true.

The above argument did not depend on a and b apart from the fact that $\sqrt{2} = \frac{a}{b}$. We now choose a and b so that at least one is odd. We can always do this, since if a and b are both even we can divide out by the largest power of 2 that divides them both. Thus if $a = 2^m a'$ and $b = 2^n b'$, where 2^m is the

[15] In the same way that 'impossible' is 'in-possible'. The way the letter 'n' changes in each case is an example of a *rule of sandhi*, a term originating in Sanskrit grammar.

largest power of 2 that divides a and where 2^n is the largest power of 2 that divides b, then the following cases arise:

$$\frac{a}{b} = \frac{a'}{b'}$$

if $m = n$, where both a' and b' are odd;

$$\frac{a}{b} = \frac{a'}{2^{n-m}b'}$$

if $n > m$, where a' is odd and $2^{n-m}b'$ is even; and

$$\frac{a}{b} = \frac{2^{m-n}a'}{b'}.$$

if $n < m$, where $2^{m-n}a'$ is even and b' is odd. Observe that $\neg q$ is logically equivalent to the statement 'at least one of a and b is odd' because

$$\neg((a \text{ is even }) \wedge (b \text{ is even })) \equiv (a \text{ is odd}) \vee (b \text{ is odd}).$$

Thus we can assume that $\neg q$ is true by choosing a and b appropriately but, as before, we can then prove that $p \to q$ is true. Under these assumptions $p \to (q \wedge \neg q)$ is also true because

$$p \to s, p \to t \vDash p \to (s \wedge t)$$

is a valid argument. But $q \wedge \neg q$ is always false. Thus p must be false and so $\neg p$ is true by Example 1.10.8. It follows that $\sqrt{2}$ is irrational.

Why whould you care about this result? It implies that you can never get an exact rational value for $\sqrt{2}$. If you calculate $\sqrt{2}$ on your calculator the number that appears is in fact a rational number. For example, my calculator tells me that

$$\sqrt{2} = 1 \cdot 414213562$$

but this is just the rational number

$$\frac{1414213562}{1000000000}$$

in disguise. We proved above that this cannot be the exact value of $\sqrt{2}$. Despite this, my calculator believes that when you square $1 \cdot 414213562$ you get 2. The moral of this is that you cannot always believe what your calculator tells you.

Exercises 1.10

1. Determine which of the following really are valid arguments.

   (a) $p \to q \vDash \neg q \vee \neg p$.

   (b) $p \to q, \neg q \to p \vDash q$.

   (c) $p \to q, r \to s, p \vee r \vDash q \vee s$.

   (d) $p \to q, r \to s, \neg q \vee \neg s \vDash \neg p \vee \neg r$.

2. This question introduces the idea of *resolution* important in some computer science applications of logic. Let x_1, x_2, x_3 and y_2, y_3 be two sets of literals. Show that the following is a valid argument

$$(x_1 \vee x_2 \vee x_3), (\neg x_1 \vee y_2 \vee y_3) \vDash x_2 \vee x_3 \vee y_2 \vee y_3.$$

3. This question explores some of the properties of the symbol $\vDash$. But first we extend the notation a little. We define

$$A_1, \ldots, A_m \vDash B_1, \ldots, B_n$$

   to mean the same thing as

$$A_1, \ldots, A_m \vDash B_1 \vee \ldots \vee B_n$$

   (a) Show that $A_1, \ldots, A_m, X \vDash B_1, \ldots, B_n, X$ is always a valid argument.

   (b) Show that if $A_1, \ldots, A_m, X, Y \vDash B_1, \ldots, B_n$ is a valid argument so too is $A_1, \ldots, A_m, X \wedge Y \vDash B_1, \ldots, B_n$.

   (c) Show that if $A_1, \ldots, A_m \vDash B_1, \ldots, B_n, X, Y$ is a valid argument so too is $A_1, \ldots, A_m \vDash B_1, \ldots, B_n, X \vee Y$.

   (d) Show that if $A_1, \ldots, A_m, X \vDash B_1, \ldots, B_n$ is a valid argument so too is $A_1, \ldots, A_m \vDash \neg X, B_1, \ldots, B_n$.

   (e) Show that if $A_1, \ldots, A_m \vDash B_1, \ldots, B_n, X$ is a valid argument so too is $A_1, \ldots, A_m, \neg X \vDash B_1, \ldots, B_n$.

(f) Show that if

$$A_1, \ldots, A_m \vDash B_1, \ldots, B_n, X \text{ and } A_1, \ldots, A_m \vDash B_1, \ldots, B_n, Y$$

are valid arguments so too is

$$A_1, \ldots, A_m \vDash B_1, \ldots, B_n, X \wedge Y.$$

(g) Show that if

$$A_1, \ldots, A_m, X \vDash B_1, \ldots, B_n \text{ and } A_1, \ldots, A_m, Y \vDash B_1, \ldots, B_n$$

are valid arguments so too is

$$A_1, \ldots, A_m, X \vee Y \vDash B_1, \ldots, B_n.$$

(h) Show that if

$$A_1, \ldots, A_m \vDash B_1, \ldots, B_n, X \text{ and } A_1, \ldots, A_m, Y \vDash B_1, \ldots, B_n$$

are valid arguments so too is

$$A_1, \ldots, A_m, X \to Y \vDash B_1, \ldots, B_n.$$

1.11 Truth trees

All problems about PL can be answered using truth tables. But there are two problems.

1. The method of truth tables is hard work.
2. The method of truth tables does not generalize to first-order logic.

In this section, we shall describe an algorithm that is often more efficient than truth tables but which, more significantly, can also be generalized to first-order logic. This is the method of truth trees. It will speed up the process of answering the following questions.

- Determining whether a wff is satisfiable or not.
- Determining whether a set of wff is satisfiable or not.

- Determining whether a wff is a tautology or not.
- Determining whether an argument is valid or not.
- Converting a wff into DNF.

It is based on the following ideas.

- Given a wff A that we wish to determine is satisfiable or not we start by assuming A is satisfiable and work backwards.
- We use a data structure, a tree, to keep track efficiently of all possibilities that occur.
- We break A into smaller pieces and so the algorithm is a *divide and conquer* algorithm.
- What is not true is false. Thus we need only keep track of what is true and any other cases will automatically be false.

Truth trees are often called *(semantic) tableaux* in the literature.

1.11.1 The truth tree algorithm

The starting point is to consider the various possible shapes that a wff A can have. Observe that since $X \oplus Y \equiv \neg(X \leftrightarrow Y)$, I shall therefore not mention $\oplus$ explicitly in what follows. There are therefore nine possibilities for A.

1. $X \wedge Y$.
2. $\neg(X \vee Y)$.
3. $\neg(X \rightarrow Y)$.
4. $\neg\neg X$.
5. $X \vee Y$.
6. $\neg(X \wedge Y)$.
7. $X \rightarrow Y$.
8. $X \leftrightarrow Y$.

9. $\neg(X \leftrightarrow Y)$.

We now introduce a graphical way of representing the truth values of A in terms of the truth values for X and Y. We introduce two kinds of graphical rules.

- The *α-rule* is non-branching/and-like. If α is a wff then this rule looks like

$$\begin{array}{c} \alpha \\ | \\ \alpha_1 \\ \alpha_2 \end{array}$$

  This means that a truth assignment satisfies α precisely when it satisfies both α_1 and α_2.

- The *β-rule* is branching/or-like. If β is a wff then this rule looks like

$$\begin{array}{c} \beta \\ \diagup \quad \diagdown \\ \beta_1 \quad \beta_2 \end{array}$$

  This means that a truth assignment satisfies β precisely when it satisfies at least one of β_1 or β_2.

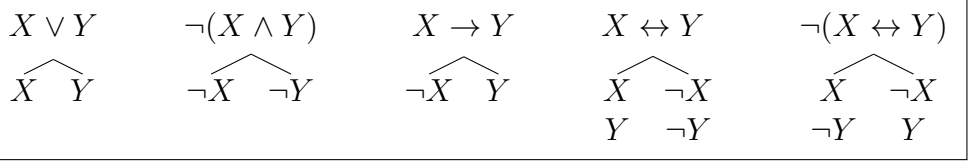
In the above two rules, we call α_1 and α_2 the *descendants* of α, and likewise for β_1 and β_2 with respect to β.

We now list the truth tree rules for the nine forms of the wff given above.

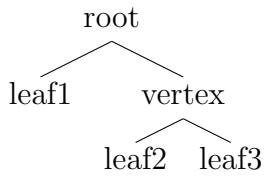
α-rules

$X \wedge Y$	$\neg(X \vee Y)$	$\neg(X \rightarrow Y)$	$\neg\neg X$
$\vert$	$\vert$	$\vert$	$\vert$
X	$\neg X$	X	X
Y	$\neg Y$	$\neg Y$	

β-rules



A truth tree for a wff A will be constructed by combining the small trees above in such a way that we shall be able to determine when A is satisfiable and how. The following definition dealing with trees is essential. Recall that a *branch* of a tree is a path that starts at the root and ends at a leaf. For example in the tree below



there are three branches. The key idea in what follows can now be stated:

> *truth flows down the branches and different branches represent alternative possibilities and so should be regarded as being combined by means disjunctions.*

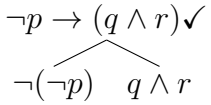
I shall describe the full truth tree algorithm once I have worked my way through some illustrative examples.

> Don't confuse *parse trees* which are about syntax with *truth trees* which are about semantics.

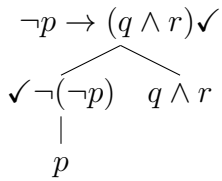
Example 1.11.1. Find all truth assignments that make the wff $A = \neg p \to (q \wedge r)$ true using truth trees. The first step is to place A at the root of what will be the truth tree

$$\neg p \to (q \wedge r)$$

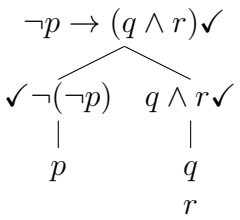
We now use the branching rule for $\to$ to get



I have used the check symbol ✓to indicate that I have *used* the occurrence of the wff $\neg p \to (q \wedge r)$. It is best to be systematic and so I shall work from left-to-right. We now apply the truth tree rule for double negation to get



where once again I have used the check symbol ✓to indicate that that occurrence of a wff has been used. Finally, I apply the truth tree rule for conjunction to get



There are now no further truth tree rules I can apply and so we say that the truth tree is *finished*. We deduce that the root A is true precisely when either p is true or (q and r are both true). In other words, $A \equiv p \vee (q \wedge r)$, which is in DNF. You can easily check that this contains exactly the same information as the rows of the truth table of A that output T. By our idea above we know that all other truth values must be F.

Before giving some more complex examples, let me highlight some important points (all of which can be proved).

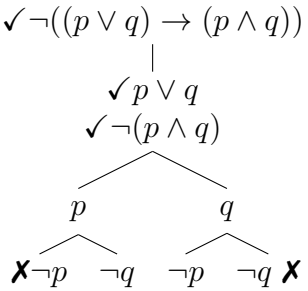
- It is the branches of the truth tree that contain information about the truth table. Each branch contains information about one or more rows of the truth table.
- It follows that all the literals on a branch must be true.
- Thus if an atom and its negation occur on the same branch then there is a contradiction. That branch is then *closed* by placing a ✗ at its leaf. No further growth takes place at a closed leaf.

The next example illustrates an important point about applying β-rules.

Example 1.11.2. Find all the truth assignments that satisfy

$$\neg((p \vee q) \to (p \wedge q)).$$

Here is the truth tree for this wff.



The truth tree is finished and there are two open branches (those branches not marked with a ✗.) The first branch tells us that p and $\neg q$ are both true and the second tells is that $\neg p$ and q are both true. It follows that the wff has the DNF

$$(p \wedge \neg q) \vee (\neg p \wedge q).$$

The key point to observe in this example is that when I applied the β-rule to the wff $\neg(p \wedge q)$ *I applied it to all branches that contained that wff.* This is crucially important since *possibilities multiply.*

The above example leads to the following *strategy.*

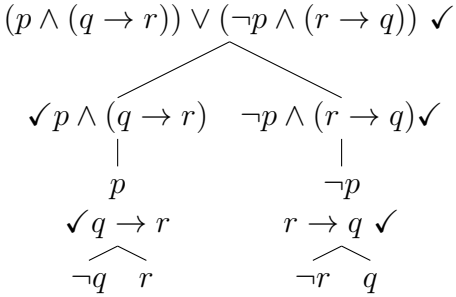
> Apply α-rules before β-rules since the application of β-rules leads to the tree gaining more branches and subsequent applications of any rule must be appended to every branch.

Here are some further examples which should be carefully worked through.

Example 1.11.3. Find all satisfying truth assignments to

$$(p \wedge (q \to r)) \vee (\neg p \wedge (r \to q)).$$

Here is the truth tree of this wff.



There are four branches and all branches are open. These lead to the DNF

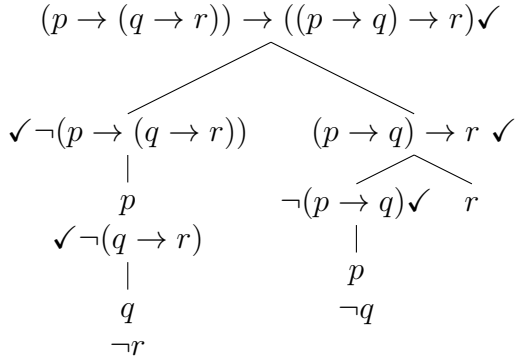
$$(p \wedge \neg q) \vee (p \wedge r) \vee (\neg r \wedge \neg p) \vee (q \wedge \neg p).$$

The satisfying truth assignments can now easily be found.

Example 1.11.4. Write

$$(p \to (q \to r)) \to ((p \to q) \to r)$$

in DNF. Here is the truth tree of this wff.



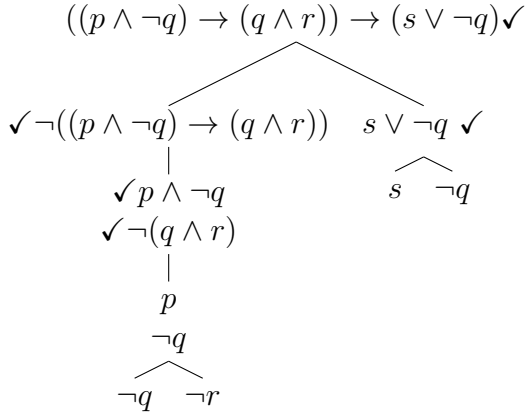
There are three branches all open. These lead to the DNF

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

Example 1.11.5. Write

$$[(p \wedge \neg q) \to (q \wedge r)] \to (s \vee \neg q),$$

which contains four atoms, in DNF. Here is the truth tree of this wff.



There are four branches all open. These lead to the DNF

$$(p \wedge \neg q) \vee (\neg r \wedge \neg q \wedge p) \vee s \vee \neg q.$$

We say that a set of wff $A_1, \ldots, A_n$ is *satisfiable* if there is at least one single truth assigment that makes them all true. This is equivalent to saying that $A_1 \wedge \ldots \wedge A_n$ is satisfiable. Thus to show that $A_1, \ldots, A_n$ is satisfiable simply list these wff as the root of a truth tree.

It's important to remember that truth trees are an algorithm for finding those truth assignments that make a wff true. This leads to the following important result.

Proposition 1.11.6. *To show that X is a tautology show that the truth tree for $\neg X$ has the property that every branch closes.*

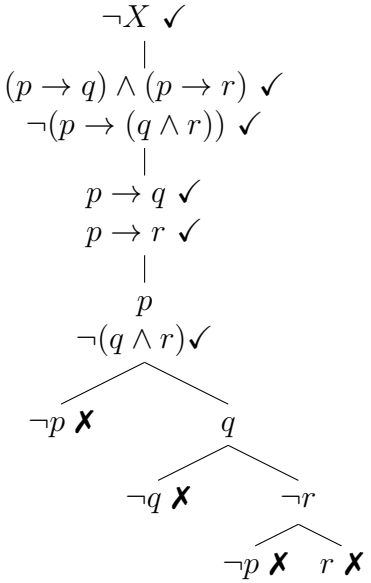
Proof. If the truth tree for $\neg X$ has the property that every branch closes then $\neg X$ is not satisfiable. This means that $\neg X$ is a contradiction. Thus X is a tautology. □

Here are some examples of showing that a wff is a tautology.

Example 1.11.7. Determine whether

$$X = ((p \to q) \wedge (p \to r)) \to (p \to (q \wedge r))$$

is a tautology or not. We begin the truth tree with $\neg X$.

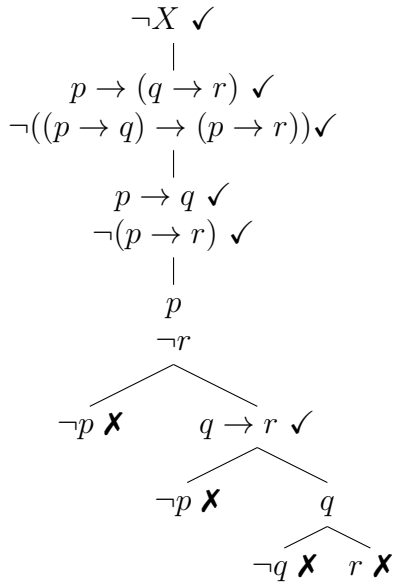


The tree for $\neg X$ closes and so $\neg X$ is a contradiction. Thus X is a tautology.

Example 1.11.8. Determine whether

$$X = (p \to (q \to r)) \to ((p \to q) \to (p \to r))$$

is a tautology or not.

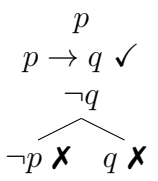


The tree for $\neg X$ closes and so $\neg X$ is a contradiction. Thus X is a tautology.

We can also use truth trees to determine whether an argument is valid or not. To see how, we use the result proved earlier that $A_1, \ldots, A_n \vDash B$ precisely when $\vDash (A_1 \wedge \ldots \wedge A_n) \to B$. Thus to show that an argument is valid using a truth tree, we could place $\neg[(A_1 \wedge \ldots \wedge A_n) \to B]$ at its root. But this is logically equivalent to $A_1 \wedge \ldots \wedge A_n \wedge \neg B$. Thus we need only list $A_1, \ldots, A_n, \neg B$ at the root of our truth tree. If every branch closes the corresponding argument is valid, and if there are open branches then the argument is not logically valid.

Examples 1.11.9. We use truth trees to show that our simple examples of arguments really are valid.

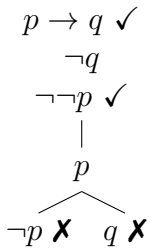
1. $p, p \to q \vDash q$ is a valid argument[16].



[16]Known as *modus ponens.*

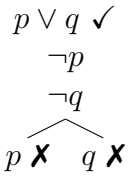
The tree closes and so the argument is valid.

2. $p \to q, \neg q \vDash \neg p$ is a valid argument[17].



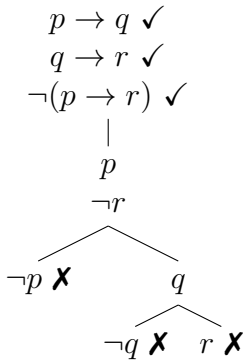
The tree closes and so the argument is valid.

3. $p \vee q, \neg p \vDash q$ is a valid argument[18].



The tree closes and so the argument is valid.

4. $p \to q, q \to r \vDash p \to r$ is a valid argument[19].



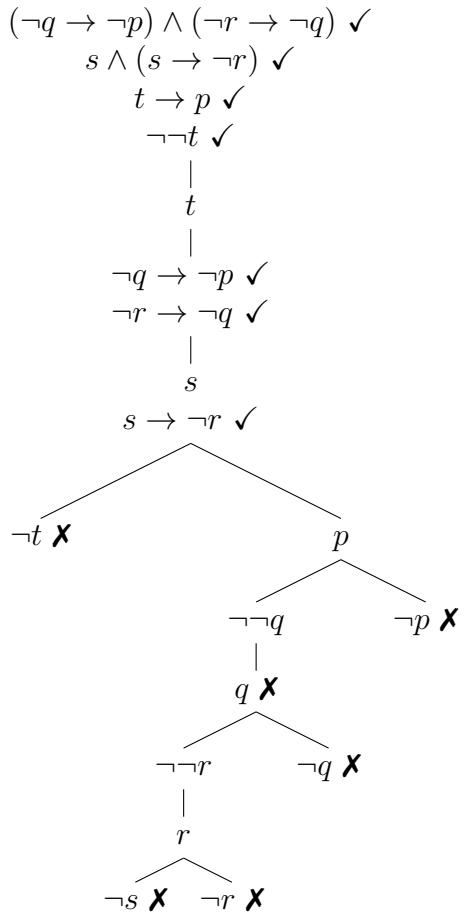
The tree closes and so the argument is valid.

Example 1.11.10. Show that the following

$$(\neg q \to \neg p) \wedge (\neg r \to \neg q), s \wedge (s \to \neg r), t \to p \vDash \neg t$$

is a valid argument.

[17]Known as *modus tollens.*
[18]Known as *disjunctive syllogism.*
[19]Known as *hypothetical syllogism.*



The tree closes and so the argument is valid.

Truth tree algorithm

Input: wff X.

Procedure: Place X at what will be the root of the truth tree. Depending on its shape, apply either an α-rule or a β-rule. Place a ✓against X to indicate that it has been *used*.

Now repeat the following:

- *Close* any branch that contains an atom and its negation by placing a cross ✗ beneath the leaf defining that branch.
- If all branches are closed then *stop* since the tree is now *finished and closed*.
- It not all branches are closed but the tree contains only literals or used wff then *stop* since the tree is now *finished and open*.
- If the tree is not finished then choose an unused wff Y which is not a literal. Now do the following: for each open branch that contains Y append the effect of applying either the α-rule or β-rule to Y, depending on which is appropriate, to the leaf of that branch.

Output:

- If the tree is finished and closed then X is a contradiction.
- If the tree is finished and open then X is satisfiable. We may find all the truth assignments that make X true as follows: for each open branch in the finished tree for X assign the value T to all the literals in that branch. If any atoms are missing from this branch then they may be assigned truth values arbitrarily.

Applications of truth trees

1. To prove that X is a *tautology*, show that the finished truth tree for $\neg X$ is closed.

2. To prove that X is *satisfiable*, show that the finished truth tree for X is open.

3. To prove that $A_1, \ldots, A_n \models B$ is a *valid argument*, place

$$A_1, \ldots, A_n, \neg B$$

at the root of the tree and show that the finished truth tree is closed.

4. To put X into *DNF*, construct the truth-tree for X. Assume that when finished it is open. For each open branch i, construct the conjunction of the literals that appear on that branch C_i. Then form the disjunction of the C_i.

1.11.2 *The theory of truth trees*

In this section, we shall prove that truth trees really do the job we have claimed for them. Before we prove our main theorem, we prove two lemmas that do all the heavy lifting for us. There is an extra piece of notation I shall use. Let X be a wff with atoms $p_1, \ldots, p_n$. Assigning truth values to these atoms will ultimately lead to a truth value being assigned to X. If we call the truth assignment to these atoms τ then the corresponding truth value of X is denoted by $\tau(X)$.

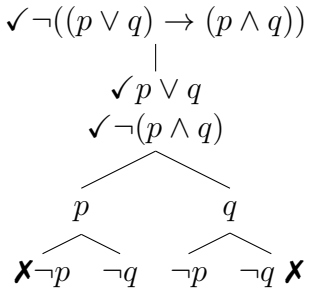
Example 1.11.11. Let $X = \neg((p \wedge q) \rightarrow r)$ and let τ be the following truth assignment

p	q	r
T	T	F

Then $\tau(p \wedge q) = T$ and $\tau((p \wedge q) \rightarrow r) = F$ so that $\tau(X) = T$.

The first result we shall prove states that if a finished truth tree with root X has an open branch then there is a truth assignment, constructed from that branch, that satisfies X. It is well to begin with a concrete example.

Example 1.11.12. We return to the truth tree for the wff $\neg((p \vee q) \rightarrow (p \wedge q))$.



The truth tree is finished and there are two open branches (those branches not marked with a ✗.) Here are the sets of formulae in each of the two open branches

$$H_1 = \{\neg((p \vee q) \rightarrow (p \wedge q)), p \vee q, \neg(p \wedge q), p, \neg q\}$$

and

$$H_2 = \{\neg((p \vee q) \rightarrow (p \wedge q)), p \vee q, \neg(p \wedge q), q, \neg p\}.$$

We focus on the first set H_1 though what we shall say applies equally well to H_2. Observe first that the set H_1 does not contain an atom and the negation of that atom. This is because H_1 arises from an open branch. Next observe, that if an α-wff is in H_1 then both α_1 and α_2 are also in H_1. In this instance, the only α-wff is $\neg((p \vee q) \rightarrow (p \wedge q))$ and here $\alpha_1 = p \vee q$ and $\alpha_2 = \neg(p \wedge q)$, both of which belong to H_1. Finally, if a β-wff belongs to H_1 then at least one of β_1 or β_2 belongs to H_1. In this case, there are two β-wff in H_1: namely, the wff $p \vee q$ and $\neg(p \wedge q)$ From the first wff the β_1-wff, p, belongs to H_1, and from the second wff the β_2-wff, $\neg q$, belongs to H_1. So, why are these properties of H_1 significant? Assign p the truth value T and assign q the truth value F. Call this the *initial truth assignment*. Now we 'climb the ladder': it follows that both p and $\neg q$ are true; it therefore follows that $p \vee q$ and $\neg(p \wedge q)$ are true; it therefore follows that $\neg((p \vee q) \rightarrow (p \wedge q))$ is true. Thus our initial truth assignment leads to every wff in H_1 being assigned the truth value T. This example works in general.

A (finite) set H of wff is said to be a *Hintikka set* if it satisfies the following three conditions.

(H1) No atom and its negation belongs to H.

(H2) If an α-wff belongs to H then both α_1 and α_2 belong to H.

(H3) If a β-wff belongs to H then at least one of β_1 or β_2 belongs to H.

Hintikka sets are named after the logician Jaakko Hintikka (1929–2015).

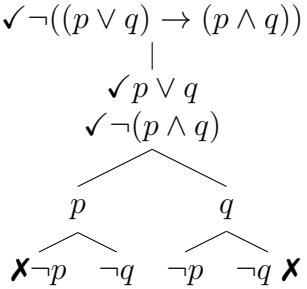
The key result about Hintikka sets is the following lemma which is nothing else than Example 1.11.12 generalized.

Lemma 1.11.13 (Hintikka sets)**.** *Let H be a Hintikka set. Then there is an assignment of truth values to the atoms in H that makes every wff in H true.*

Proof. Let the literals occurring in H be $l_1, \ldots, l_n$. If $l_i = p_i \in H$ then assign p_i the truth value T; if $l_i = \neg p_i \in H$ then assign p_i the truth value F. By (H1), each atom will be assigned exactly one truth value. Any other atoms that occur in the wff in H can be assigned truth values arbitrarily. Call this the *initial truth assignment*. All literals in H now have the truth value T by design. We now 'climb the ladder'. Suppose that α is an α-wff in H. Then by (H2) both α_1 and α_2 belong to H. If α_1 and α_2 are both true under our initial truth assignment so too will α. Suppose that β is an β-wff in H. Then by (H3) at least one of β_1 and β_2 belongs to H. Which ever one it is, if the initial truth assignment makes it true then so too will be β. By climbing the ladder in this way from the initial truth assignment, through the literals, and ever more complex wff in H we reach the point where every wff in H has been shown to be true under the initial truth assignment. □

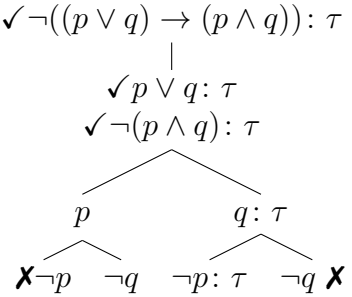
Our second result starts with a wff X and some truth assignment τ making X true. We prove that there is at least one branch in any truth tree with root X which is true under τ. Again, this is best motivated by an example.

Example 1.11.14. We begin with the wff $X = \neg((p \vee q) \to (p \wedge q))$. A truth assignment τ making X true is $p = F$ and $q = T$. The truth tree for X is as follows.



I claim that there is at least one open branch such that all wff on that branch are true under the valuation τ. You can easily check that τ is true

for all wff on the second open branch. To illustrate this, I have written the $: \tau$ next to the wff in the second open branch.



Lemma 1.11.15. *Let X be a wff and let τ be a truth assignment to the atoms of X that makes X true. Then for any completed truth tree for X there is at least one open branch such that all wff on that branch are true under τ.*

Proof. Construct a finished truth tree with root X. Write $: \tau$ next to X. I shall describe a process for labelling wff in the truth tree with the decoration $: \tau$. If Y is a wff so labelled then $\tau(Y) = T$. When this labelling process is complete we shall show that there will be at least one open branch of the truth tree in which every wff on that branch is labelled $: \tau$; this will prove the lemma. Suppose that there is a path, not a branch, in the truth tree from root to the wff Y such that every wff on that path has been labelled with $: \tau$.

- If Y is an α-formula then both of its descendants will be labelled with $: \tau$.
- If Y is a β-formula then at least one of its descendants will be labelled with $: \tau$.

Thus every path, not a branch, can be extended. It follows that there must be a branch on which every wff is labelled with $: \tau$. But then it follows that this branch must be open because it cannot contain an atom and its negation since both would be assigned the value T by τ which is impossible. □

We use Lemmas 1.11.13 and 1.11.15 to prove the main theorem of this section.

Theorem 1.11.16 (Truth trees work)**.**

1. *Let X be a tautology. Then every finished truth tree with root $\neg X$ is closed.*

2. *If some finished truth tree with root $\neg X$ is closed then X is a tautology.*

Proof. (1) Suppose that there is a finished truth tree with root $\neg X$ which is not closed. Then there is an open branch. This open branch is a Hintikka set that contains $\neg X$. By Lemma 1.11.13, there is a truth assignment that makes $\neg X$ true, but this is impossible since $\neg X$ is a contradiction. It follows that there can be no such open branch and so any finished truth tree with root $\neg X$ is closed.

(2) Suppose that there is a finished truth tree T with root $\neg X$ which is closed but that X is not a tautology. Then there is some truth assignment τ making $\neg X$ true. Then by Lemma 1.11.15, there is at least one open branch for T along which τ is true. But this contradicts the fact that T is closed. It follows that X must be a tautology. □

Exercises 1.11

---

1. Determine whether the following arguments are valid or not using truth trees.

   (a) $p \to q, r \to s, p \vee r \models q \vee s$.

   (b) $p \to q, r \to s, \neg q \vee \neg s \models \neg p \vee \neg r$.

   (c) $p \to q, r \to s, p \vee \neg s \models q \vee \neg r$.

2. Show that the following are tautologies using truth trees.

   (a) $q \to (p \to q)$.

   (b) $[(p \to q) \wedge (q \to r)] \to (p \to r)$.

   (c) $[(p \to q) \wedge (p \to r)] \to (p \to (q \wedge r))$.

   (d) $[((p \to r) \wedge (q \to r)) \wedge (p \vee q)] \to r$.

3. In [32, Chapter 1], Winnie-the-Pooh makes the following argument.

(a) There is a buzzing.

(b) If there is a buzzing then somebody is making the buzzing.

(c) If somebody is making the buzzing then somebody is bees.

(d) If somebody is bees then there is honey.

(e) If there is honey then there is honey for Pooh to eat.

(f) Therefore there is honey for Pooh to eat.

Using the letters $p, q, r \ldots$ express this argument in symbolic form and use truth trees to determine whether Pooh's argument is valid.

1.12 *Sequent calculus*

Truth trees can be used to *check* whether a wff is a tautology or not whereas in this section, we describe a method for *generating* tautologies. These two methods complement each other or, as mathematicians like to say, the two methods are *dual* to each other. To make our life a little easier, in this section I will only use the connectives $\neg, \wedge, \vee, \rightarrow$ but as we know from Section 1.6 this will not affect the expressiveness of our logic. The method we describe will involve a change in perspective. First, we shall extend the meaning of the notation introduced in Section 1.10. Let $A_1, \ldots, A_m, B_1, \ldots, B_n$ be wff. We define

$$A_1, \ldots, A_n \vDash B_1, \ldots, B_n$$

to mean the same thing as

$$A_1, \ldots, A_n \vDash B_1 \vee \ldots \vee B_n.$$

In itself, this is a difference without a distinction but it serves to make the symbol $\vDash$ symmetrical fore and aft. Second, rather than working with tautologies directly and valid arguments indirectly, I will work with arguments, in our extended sense, directly. This was an approach to mathematical logic developed by Gerhard Gentzen (1909–1945) who is the presiding genius of this section. Thus given the wff $A_1, \ldots, A_m$ on the left hand side and the wff $B_1, \ldots, B_n$ on the right hand side, we want to know whether $A_1, \ldots, A_n \vDash B_1, \ldots, B_n$ is a valid argument. In order not to pre-empt the answer to this question, we now introduce a new symbol $\Rightarrow$ whose role is simply to act as a punctuation mark between the set of left hand wff and the

set of right hand wff. More formally, by a *sequent* is meant an expression of the form

$$A_1, \ldots, A_n \Rightarrow B_1, \ldots, B_n$$

where $A_1, \ldots, A_m$ and $B_1, \ldots, B_n$ should each be interpreted as sets of wff; this means that the order of the wff on the left or of those on the right does not matter and, in addition, repeats on the left or on the right can be introduced or removed as required. A sequent of the form $\mathbf{U} \Rightarrow \emptyset$ will simply be written $\mathbf{U} \Rightarrow$. Similarly, a sequent of the form $\emptyset \Rightarrow \mathbf{V}$ will simply be written $\Rightarrow \mathbf{V}$. Our goal is to introduce some rules that will enable us to generate sequents that represent valid arguments. We call this a *proof* of a sequent. The big reveal will be a theorem that says what can be proved is true and what is true can be proved.[20] Truth is an elusive concept: for the politician, infinitely malleable, for the scientist, the Holy Grail. In classical Greece, mathematicians figured out a way of establishing truth. Begin with statements that you know to be true; such statements are called *axioms*. Then find *rules* that are known to preserve truth. Clearly, if you start with axioms and apply the rules a finite number of times then the statements you end up with will all certainly be true; this is called *soundness*. The crucial question is whether *all* truths can be found in this way. If they can, this is called *completeness*. We shall apply this approach to generating valid arguments and prove that we have, indeed, snared all and only the valid ones.

I now define a system I shall call $\mathscr{S}$, where the 'S' stands for 'Smullyan' though we have two extra negation rules compared to what you will find in [40]. For convenience, I shall use bold letters, such as $\mathbf{U}$, to mean a set of wff, possibly empty, whereas I shall use ordinary letters, such as X, to mean individual wff. Our *axioms* are all sequents of the form

$$\mathbf{U}, X \Rightarrow \mathbf{V}, X.$$

Observe that the corresponding argument

$$\mathbf{U}, X \vDash \mathbf{V}, X$$

is valid because if all the wff on the left hand side are true this means, in particular, that X is true; it follows that the disjunction of the wff on the

[20] This is reminiscent of, but less poetical than, Keats's *"Beauty is truth, truth beauty" — that is all ye know on earth, and all ye need to know,* which, from a logical point of view, is highly questionable.

right hand side is also true because X also occurs there. Our *rules* will have one of the following two shapes. The first is

$$\frac{S_1}{S_2}$$

which means that if the sequent S_1 is given as an assumption then the sequent S_2 can be deduced. The second is

$$\frac{S_1 \qquad S_2}{S_3}$$

which means that if the two sequents S_1 and S_2 are given as assumptions then the sequent S_3 can be deduced. The rules themselves follow. Apart from the rule for negation, each rule comes in two flavours: (1) one assumption or (2) two assumptions. The rule for negation is either (r) right-moving or (l) left-moving.

Conjunction

$$\frac{\mathbf{U}, X, Y \Rightarrow \mathbf{V}}{\mathbf{U}, X \wedge Y \Rightarrow \mathbf{V}}$$

$$\frac{\mathbf{U} \Rightarrow \mathbf{V}, X \qquad \mathbf{U} \Rightarrow \mathbf{V}, Y}{\mathbf{U} \Rightarrow \mathbf{V}, X \wedge Y}$$

Conjunction (1) simply expresses the fact that the comma on the left hand side of a valid argument can be interpreted as conjunction. Conjunction (2) needs a little arguing. Suppose that all the wff in $\mathbf{U}$ are assumed true. Then, by assumption, either $\mathbf{V}$ is true or X is true, and either $\mathbf{V}$ is true or Y is true. If $\mathbf{V}$ is not true then both X and Y are true and so $X \wedge Y$ is true. It follows that if $\mathbf{U}$ is true then either $\mathbf{V}$ is true or $X \wedge Y$ is true.

Disjunction

$$\frac{\mathbf{U} \Rightarrow \mathbf{V}, X, Y}{\mathbf{U} \Rightarrow \mathbf{V}, X \vee Y}$$

$$\frac{\mathbf{U}, X \Rightarrow \mathbf{V} \qquad \mathbf{U}, Y \Rightarrow \mathbf{V}}{\mathbf{U}, X \vee Y \Rightarrow \mathbf{V}}$$

Disjunction (1) simply expresses the fact that the comma on the right hand side can be interpreted as disjunction. Disjunction (2) needs a little arguing. Suppose that all the wff in $\mathbf{U}$ are true and $X \vee Y$ is true. Thus either $\mathbf{U}, X$ are all true or $\mathbf{U}, Y$ are all true. In both cases, all the wff in $\mathbf{V}$ are true.

For negation, there are two right-moving rules and two left-moving rules. I have split the negation rules up into two boxes for reasons that I shall explain towards the end of this section.

Negation (appearing)

$$\frac{\mathbf{U}, X \Rightarrow \mathbf{V}}{\mathbf{U} \Rightarrow \mathbf{V}, \neg X}$$

$$\frac{\mathbf{U} \Rightarrow \mathbf{V}, X}{\mathbf{U}, \neg X \Rightarrow \mathbf{V}}$$

Negation (disappearing)

$$\frac{\mathbf{U}, \neg X \Rightarrow \mathbf{V}}{\mathbf{U} \Rightarrow \mathbf{V}, X}$$

$$\frac{\mathbf{U} \Rightarrow \mathbf{V}, \neg X}{\mathbf{U}, X \Rightarrow \mathbf{V}}$$

Both Negation (r) and Negation (l) are straightforward.

Implication

$$\frac{\mathbf{U}, X \Rightarrow \mathbf{V}, Y}{\mathbf{U} \Rightarrow \mathbf{V}, X \to Y}$$

$$\frac{\mathbf{U} \Rightarrow \mathbf{V}, X \qquad \mathbf{U}, Y \Rightarrow \mathbf{V}}{\mathbf{U}, X \to Y \Rightarrow \mathbf{V}}$$

Implication (1) follows by applying Negation (r) to X and then applying Disjunction (1). Implication (2) follows by applying Negation (l) to X and then applying Disjunction (2).

A *proof* of a sequent is a tree of sequents, I shall call it a *proof tree*, but this time the tree has the root at the bottom and the leaves at the top in such a way that the leaves are axioms and the root is the sequent we are trying to prove and only the rules above are used in constructing the tree. The axioms, rules and notion of proof we have just described form what is called *sequent calculus*.

Example 1.12.1. The following is a very simple proof in sequent calculus with two leaves only and one application of Implication (2).

$$\frac{p \Rightarrow q, p \qquad p, q \Rightarrow q}{p, p \to q \Rightarrow q}$$

This shows that modus ponens is a valid argument.

> Let me restate our goal. We want to prove that an argument $\mathbf{U} \vDash \mathbf{V}$ is valid if and only if the sequent $\mathbf{U} \Rightarrow \mathbf{V}$ can be proved in sequent calculus.

To achieve this goal, we introduce a modification of $\mathscr{S}$, called $\mathscr{S}^*$, and a modification of truth trees, called block truth trees.

The system $\mathscr{S}^*$ consists of sequents of the form $\mathbf{S} \Rightarrow$ called $*$-*sequents*. The $*$-*axioms* are those sequents of the form $\mathbf{S} \Rightarrow$ where $\mathbf{S}$ contains both an atom and its negation. The $*$-*rules* are listed below.

$*$-Conjunction

$$\frac{\mathbf{S}, X, Y \Rightarrow}{\mathbf{S}, X \wedge Y \Rightarrow}$$

$$\frac{\mathbf{S}, \neg X \Rightarrow \qquad \mathbf{S}, \neg Y \Rightarrow}{\mathbf{S}, \neg(X \wedge Y) \Rightarrow}$$

$*$-Disjunction

$$\frac{\mathbf{S}, \neg X, \neg Y \Rightarrow}{\mathbf{S}, \neg(X \vee Y) \Rightarrow}$$

$$\frac{\mathbf{S}, X \Rightarrow \qquad \mathbf{S}, Y \Rightarrow}{\mathbf{S}, X \vee Y \Rightarrow}$$

$*$-Negation

$$\frac{\mathbf{S}, X \Rightarrow}{\mathbf{S}, \neg\neg X \Rightarrow}$$

$*$-Implication

$$\frac{\mathbf{S}, X, \neg Y \Rightarrow}{\mathbf{S}, \neg(X \rightarrow Y) \Rightarrow}$$

$$\frac{\mathbf{S}, \neg X \Rightarrow \qquad \mathbf{S}, Y \Rightarrow}{\mathbf{S}, X \rightarrow Y \Rightarrow}$$

Using $*$-axioms and $*$-rules we get $*$-*proofs*.

We now describe the slight modification of truth trees, called *block truth trees*.

> In truth trees, truth runs down the branches whereas in block truth trees, truth lives at the leaves.

Rather than α-rules and β-rules, there are A-rules and B-rules. Let $\mathbf{S}$ be a set of wff. Then the two types of rules are as follows.

- The *A-rule*

$$\begin{array}{c} \mathbf{S}, \alpha \\ | \\ \mathbf{S}, \alpha_1, \alpha_2 \end{array}$$

- The *B-rule*

$$\begin{array}{ccc} & \mathbf{S}, \beta & \\ \swarrow & & \searrow \\ \mathbf{S}, \beta_1 & & \mathbf{S}, \beta_2 \end{array}$$

Block truth trees are constructed and used just like truth trees except that now a vertex is closed if it contains an atom and its negation.

Example 1.12.2. We construct the block truth tree with root

$$\neg((p \vee q) \to (p \wedge q)).$$

It is as follows

$$\begin{array}{c} \neg((p \vee q) \to (p \wedge q)) \\ | \\ p \vee q, \neg(p \wedge q) \\ \swarrow \qquad \searrow \\ p, \neg(p \wedge q) \qquad q, \neg(p \wedge q) \\ \swarrow \quad \searrow \qquad \swarrow \quad \searrow \\ \text{✗}p, \neg p \quad p, \neg q \quad q, \neg p \quad q, \neg q \text{ ✗} \end{array}$$

There is no real need to employ checks to show which wff have been used since in block truth trees one is always moving down carrying all the information one needs. The price paid is that more wff have to be written down.

Suppose that a finished block truth tree with root $\neg X$ is closed. Then each leaf is a set of wff that contains an atom and its negation.

We now write down explicitly what the A-rules and the B-rules look like.

A-rules

$$\begin{array}{cccc} \mathbf{S}, X \wedge Y & \mathbf{S}, \neg(X \vee Y) & \mathbf{S}, \neg(X \to Y) & \mathbf{S}, \neg\neg X \\ | & | & | & | \\ \mathbf{S}, X, Y & \mathbf{S}, \neg X, \neg Y & \mathbf{S}, X, \neg Y & \mathbf{S}, X \end{array}$$

B-rules

$$\begin{array}{ccc} \mathbf{S}, X \vee Y & \mathbf{S}, \neg(X \wedge Y) & \mathbf{S}, X \to Y \\ \mathbf{S}, X \quad \mathbf{S}, Y & \mathbf{S}, \neg X \quad \mathbf{S}, \neg Y & \mathbf{S}, \neg X \quad \mathbf{S}, Y \end{array}$$

We now come to the key observation. The *-rules and the A-and B-rules are the same except that the *-rules are the A-and B-rules turned upside down. We now have all the ingredients needed to prove the main theorem of this section.

Theorem 1.12.3. *(Soundness and completeness) We have that*

$$A_1, \ldots, A_n \vDash B_1, \ldots, B_n$$

is a valid argument if and only if the sequent

$$A_1, \ldots, A_n \Rightarrow B_1, \ldots, B_n$$

can be proved in the system $\mathscr{S}$.

Proof. Soundnesss amounts to showing that if the sequent

$$A_1, \ldots, A_n \Rightarrow B_1, \ldots, B_n$$

can be proved then

$$A_1, \ldots, A_n \vDash B_1, \ldots, B_n$$

is a valid argument. How to prove this is the job of Question 3 of Exercises 1.10.

This leaves us with proving completeness which amounts to showing that if

$$A_1, \ldots, A_n \vDash B_1, \ldots, B_n$$

is a valid argument then the sequent

$$A_1, \ldots, A_n \Rightarrow B_1, \ldots, B_n$$

can be proved.

We shall prove this by piggy-backing off our work on block truth trees. We start with the block truth tree with root

$$A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_n.$$

By assumption, the finished tree will be closed. Turn this tree upside-down. Replace each set $\mathbf{S}$ of wff at each vertex by the corresponding sequent $\mathbf{S} \Rightarrow$. This tree is now a *-proof the sequent

$$A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_n \Rightarrow .$$

Now, every $*$-proof of a sequent $\mathbf{S} \Rightarrow$ can easily be converted into a proof of that sequent: one simply uses Negation (l) and (r) to shuffle symbols between the left hand side of the arrow and the right hand side. In this way, the $*$-axioms can be proved from the axioms. It also follows that we can prove the sequent

$$A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_n \Rightarrow$$

By using Negation (r), we can therefore obtain a proof of $A_1, \ldots, A_n \Rightarrow B_1, \ldots, B_n$. □

Example 1.12.4. We shall illustrate the proof of the above theorem by means of a concrete example. We prove the sequent

$$p \to q, \neg q \Rightarrow \neg p.$$

First, we construct the block truth tree that has as its root

$$p \to q, \neg q, \neg\neg p.$$

This is just

$$\begin{array}{c} p \to q, \neg q, \neg\neg p \\ | \\ p \to q, \neg q, p \\ \diagup \quad \diagdown \\ \neg p, \neg q, p \ \text{✗} \qquad q, \neg q, p \ \text{✗} \end{array}$$

Now turn this tree upside down — or right side up — and write everything as $*$-sequents.

$$\dfrac{\dfrac{\neg p, \neg q, p \Rightarrow \qquad q, \neg q, p \Rightarrow}{p \to q, \neg q, p \Rightarrow}}{p \to q, \neg q, \neg\neg p \Rightarrow}$$

Finally, we use the Negation rules to obtain a proof in sequent calculus that has leaves labelled by axioms and has a root labelled with the sequent we are trying to prove.

$$\dfrac{\dfrac{\dfrac{\dfrac{\neg q, p \Rightarrow p}{\neg p, \neg q, p \Rightarrow} \qquad \dfrac{q, p \Rightarrow q}{q, \neg q, p \Rightarrow}}{p \to q, \neg q, p \Rightarrow}}{p \to q, \neg q, \neg\neg p \Rightarrow}}{p \to q, \neg q, \Rightarrow \neg p}$$

Remark 1.12.5. It is possible to prove that any sequent that can be proved in the system $\mathscr{S}$ can also be proved in the same system with the rule Negation (disappearing) omitted. A formal proof can be found in [6, Lemma 14.15 and Corollary 14.16].

Example 1.12.6. It is always possible to prove sequents using the method of the proof of Theorem 1.12.3 but there is a better way. We adapt the methodology of truth trees to sequents and work up from the sequent we are trying to prove (as the root) to the axioms (as the leaves).[21] To illustrate this approach we prove the sequent

$$(q \to p) \wedge (p \vee q) \Rightarrow p.$$

We therefore start with this sequent as the root and set about growing the tree above it as part of a *proof search*. The sequent rules should now be read from bottom to top. Our immediate goal at each step is to simplify the current sequent. The obvious rule to apply is Conjunction (1) (in reverse). This yields

[21] Historically, of course, it was the other way around.

$$\frac{(q \to p), (p \vee q) \Rightarrow p}{(q \to p) \wedge (p \vee q) \Rightarrow p}$$

The top sequent is simpler than the bottom one because it has one less connective. At this point we have a choice, but we apply the rule Implication (2) (in reverse). This yields

$$\frac{\dfrac{p \vee q \Rightarrow p, q \qquad p, p \vee q \Rightarrow p}{(q \to p), (p \vee q) \Rightarrow p}}{(q \to p) \wedge (p \vee q) \Rightarrow p}$$

At this point, observe that $p, p \vee q \Rightarrow p$ is an axiom. We therefore do not want to analyse this any further. To finish off, we apply the rule Disjunction (2). This yields

$$\frac{\dfrac{\dfrac{p \Rightarrow p, q \qquad q \Rightarrow p, q}{p \vee q \Rightarrow p, q} \qquad p, p \vee q \Rightarrow p}{(q \to p), (p \vee q) \Rightarrow p}}{(q \to p) \wedge (p \vee q) \Rightarrow p}$$

Our search for a proof ends because all leaves are axioms. Having worked backwards to find the proof, we can now read the proof forwards from leaves to root.

The next example shows what happens when at least one of the leaves obtained in the tree is not an axiom. Observe that $\Rightarrow X$ will be proved in the sequent calculus if and only if X is a tautology.

Example 1.12.7. The proof tree for the sequent

$$\Rightarrow (p \to q) \wedge (\neg p \to \neg q)$$

is as follows

$$\frac{\dfrac{p \Rightarrow q}{\Rightarrow p \to q} \qquad \dfrac{\dfrac{\dfrac{q \Rightarrow p}{q, \neg p \Rightarrow}}{\neg p \Rightarrow \neg q}}{\Rightarrow \neg p \to \neg q}}{\Rightarrow (p \to q) \wedge (\neg(p \to \neg q)}$$

However, neither leaf is an axiom. This means that our sequent cannot be proved. But we can say more. Consider the leaf $p \Rightarrow q$. Assign the value T to p and F to q. This truth assignment makes the wff $(p \to q) \wedge (\neg p \to \neg q)$ take the value F. This means that it is not a tautology.

We defined sequent calculus using what we called system $\mathscr{S}$. Gentzen himself used a system he called **LK**. However, it can be shown that everything that can be proved in $\mathscr{S}$ can be proved, when suitably interpreted, in the system **LK**, and vice-versa.

Exercises 1.12

---

1. Prove the following sequents.

   (a) $\Rightarrow \neg\neg p \to p$.

   (b) $p \vee q, \neg p \Rightarrow q$.

   (c) $p \to q, q \to r \Rightarrow p \to r$.

   (d) $p \to q, r \to s, p \vee r \Rightarrow q \vee s$.

Chapter 2

Boolean algebras

> *"He's out" said Pooh sadly, "That's what it is. He's not in."* —
> Winnie-the-Pooh.

Unlike the other two chapters, this one is not about logic per se but about the algebra that arises from PL. I have included it because the main application is to circuit design: specifically, the kinds of circuits that are needed to build computers. It therefore provides a bridge between logic and the real world. Only Section 2.1 is a prerequisite for Chapter 3.

2.1 Set theory

Sets were briefly touched upon in Section 1.2, but now we need to study them in more detail. Set theory, invented by Georg Cantor (1845–1918) in the last quarter of the nineteenth century, begins with two deceptively simple definitions on which everything is based.

1. A *set* is a collection of objects, called *elements*, which we wish to view as a whole.

2. Two sets are *equal* precisely when they contain the same elements.

Because a set is regarded as a single entity it can be given a name. It is customary to use capital letters as names such as $A, B, C \ldots$ or fancy capital letters such as $\mathbb{N}, \mathbb{Z} \ldots$ with the elements of the set usually being 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 '$x \notin A$.' To indicate that some things are to be regarded as a set rather than just as isolated individuals, we enclose them in 'curly brackets' { and } (officially these are called braces). Thus the set of suits in a pack of cards is $\{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}$.

A set should be regarded as a bag of elements, and so the order of the elements within the set is not important. Thus $\{a, b\} = \{b, a\}$. Perhaps, more surprisingly, repetition of elements is ignored. Thus $\{a, b\} = \{a, a, a, b, b, a\}$. Sets can be generalized to what are called *multisets* where repetition is recorded but we shall not need this concept in this course.

The set $\{\}$ is empty and is called the *empty set*. If I just wrote $\{\}$ you might think that I had absent-mindedly forgotten to list the elements so the empty set is given a special symbol $\emptyset$. But it is important to remember that the symbol $\emptyset$ means exactly the same thing as $\{\}$. In particular, observe that $\emptyset \neq \{\emptyset\}$ since the empty set contains no elements whereas the set $\{\emptyset\}$ contains one element.

A set is *finite* if it only has a finite number of elements, otherwise it is *infinite*. A set with exactly one element is called a *singleton set*. Thus $\{a\}$ is an example of a singleton set. The element of this singleton set is a. But $a \neq \{a\}$ because the left hand side is an element and the right hand side is a set that contains one element: namely, a.[1]

We can sometimes define infinite sets by using curly brackets but then, because we cannot 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 big finite sets where there is an obvious pattern. However, the most common way of describing a set is to say what properties an element must have to belong to it — in other words, what the membership conditions of the set are. By a *property* we mean a sentence containing a variable such as x so that the sentence becomes true or false depending on what we substitute for x. We shall say much more about properties in Chapter 3 since they are an important ingredient in first-order logic. For example, the sentence 'x is an even natural number' is true when x is replaced by 2 and false when x is

[1] If I have a sheep as pet, I could hardly be said to have a flock of sheep, the word 'flock' being used instead of the word 'set' when dealing with sheep collections. On the other hand, if I do have a flock of sheep but, sadly, all bar one of the sheep were eaten by ravenous wolves then I do have a flock, albeit consisting of one lone sheep. Incidently, English has the odd feature that it uses different words to mean set or collection in different situations. Thus a herd is a set of cows, a school is a set of whales and a mumuration is a set of starlings.

replaced by 3. If we abbreviate 'x is an even natural number' by $E(x)$ then the set of even natural numbers is the set of all natural numbers n such that $E(n)$ is true. This set is written $\{x\colon\ E(x)\}$ or $\{x \mid E(x)\}$. More generally, if $P(x)$ is any property then $\{x\colon\ P(x)\}$ means 'the set of all things x that satisfy the condition P'. The following notation will be useful when we come to study first-order logic.

Examples 2.1.1.

1. The set $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ of all *natural numbers*. Caution is required here since some books eccentrically do not regard 0 as a natural number.

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*. That is those numbers that can be written as quotients of integers with non-zero denominators.

4. The set $\mathbb{R}$ of all *real numbers*. That is 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 into B. We say that B is a *subset* of A, denoted by $B \subseteq A$. In mathematics, the word 'choose', unlike in polite society, also includes the possibility of *choosing nothing* and the possibility of *choosing everything*. In addition, there does not need to be any rhyme or reason to your choices: you can pick elements 'at random' if you want. If $A \subseteq B$ and $A \neq B$ then we say that A is a *proper subset* of B.

Examples 2.1.2.

1. $\emptyset \subseteq A$ for every set A, where we choose no elements from A.

2. $A \subseteq A$ for every set A, where we choose all the elements from A.

3. $\mathbb{N} \subseteq \mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R}$. Observe that $\mathbb{Z} \subseteq \mathbb{Q}$ because an integer n is equal to the rational number $\frac{n}{1}$.

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 $\mathbb{N}$.

Example 2.1.3. The notion of subset enables us to describe an important method of proof used in mathematics. I will use the following property of the natural numbers: every non-empty subset has a smallest element. Suppose that we wish to prove an infinite number of statements $p_0, p_1, p_2, \ldots$. This seems to be impossible but there are situations where not only is it possible, it is routine. Define X to be the subset of $\mathbb{N}$ of those i where p_i is known to be true. We make two assumptions

1. $0 \in X$.

2. If $N \in X$ then $N + 1 \in X$.

I claim that if these two assumptions hold then $X = \mathbb{N}$ which proves that p_i is true for all $i \in \mathbb{N}$. I now prove the claim. Suppose that $X \neq \mathbb{N}$. Then the set $\mathbb{N} \setminus X$ is non-empty. It therefore has a smallest element. Call it m. By assumption, $m - 1 \in X$. But by property (2), if $m - 1 \in X$ then $m \in X$, which is a contradiction. It follows that $X = \mathbb{N}$, as required.

This result leads to a method of proof is called *proof by induction*. Here is a simple example. Let p_n be the statement '$3^n - 1$ is even' where $n \in \mathbb{N}$. When $n = 0$ we have that $3^0 - 1 = 0$ which is even. Thus condition (1) above holds. We now verify that condition (2) is true. Assume that $3^N - 1$ is even. We prove that $3^{N+1} - 1$ is even. To do this requires a little mathematical manipulation. Observe that

$$3^{N+1} - 1 = 3 \cdot 3^N - 1 = 2 \cdot 3^N + \left(3^N - 1\right).$$

But, by assumption, $3^N - 1$ is even, and so $3^{N+1} - 1$ is also even.

This basic notion of induction can be extended. Let $m \geq 1$ be a natural number. Define

$$\mathbb{N}^{\geq m} = \mathbb{N} \setminus \{0, \ldots, m - 1\}.$$

Then any subset of $\mathbb{N}^{\geq m}$ that contains p, this time, but still satisfies condition (2) above, must actually be equal to $\mathbb{N}^{\geq m}$. This modified form of induction can be used to prove all the statements $p_m, p_{m+1}, p_{m+3}, \ldots$.

Here is a simple example. Define $0! = 1$ and $n! = n(n-1)!$ when $n \geq 1$. Let p_n be the statement '$n! > 2^n$'. We prove that p_n is true for all natural numbers $n \geq 4$. (You can check that the result is *not* true for $n = 0, 1, 2, 3$.)

When $n = 4$ we have that $n! = 24$ and $2^4 = 16$. Thus p_4 is true. We now prove that if p_N is true so too is p_{N+1}. Observe that

$$(N+1)! = (N+1)N! > (N+1)2^N > (1+1)2^N = 2^{N+1}$$

where we have used the fact that $N > 1$.

Although the theory behind proof by induction is simple, actually using it requires mathematical insight.

Example 2.1.4. The idea that sets are defined by properties is a natural one, but there are murky logical depths. It seems obvious that given a property $P(x)$, there is a corresponding set $\{x\colon P(x)\}$ of all those things that have that property. But we now describe a famous result in the history of mathematics called *Russell's Paradox*, named after Bertrand Russell (1872–1970), which shows that just because something is obvious does not make it true. Define $\mathscr{R} = \{x\colon x \notin x\}$. In other words: the set of all sets that do not contain themselves as an element. For example, $\emptyset \in \mathscr{R}$. We now ask the question: is $\mathscr{R} \in \mathscr{R}$? There are only two possible answers and we investigate them both.

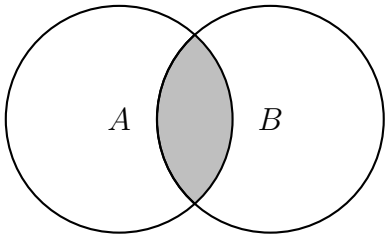
1. Suppose that $\mathscr{R} \in \mathscr{R}$. This means that $\mathscr{R}$ must satisfy the entry requirements to belong to $\mathscr{R}$ which it can only do if $\mathscr{R} \notin \mathscr{R}$.

2. Suppose that $\mathscr{R} \notin \mathscr{R}$. Then it satisfies the entry requirement to belong to $\mathscr{R}$ and so $\mathscr{R} \in \mathscr{R}$.

Thus exactly one of $\mathscr{R} \in \mathscr{R}$ and $\mathscr{R} \notin \mathscr{R}$ must be true but assuming one implies the other. We therefore have an honest-to-goodness contradiction. Our only way out is to conclude that, whatever $\mathscr{R}$ might be, it is not a set. This contradicts the obvious statement we began with. If you want to understand how to escape this predicament, you will have to study *set theory*. Disconcerting as this might be, imagine how much more so it was to the mathematician Gottlob Frege (1848–1925). He was working on a book which based the development of mathematics on sets when he received a letter from Russell describing this paradox thereby undermining what Frege was attempting to achieve.

We now define three operations on sets that are based on the PL logical connectives $\wedge$, $\vee$ and $\neg$. They are called *Boolean operations*, named after George Boole. Let A and B be sets.

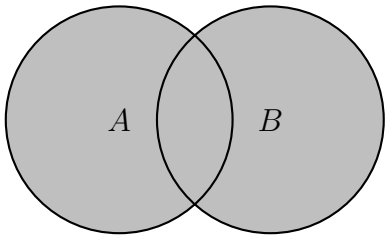
Define a set, called the *intersection* of A and B, denoted by $A \cap B$, whose elements consist of all those elements that belong to A **and** B.



More formally, we can write

$$A \cap B = \{x \colon (x \in A) \wedge (x \in B)\}.$$

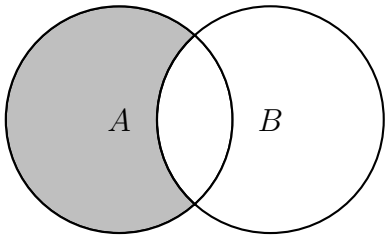
Define a set, called the *union* of A and B, denoted by $A \cup B$, whose elements consist of all those elements that belong to A **or** B.



More formally, we can write

$$A \cup B = \{x \colon (x \in A) \vee (x \in B)\}.$$

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



More formally, we can write

$$A \setminus B = \{x \colon (x \in A) \wedge \neg(x \in B)\}.$$

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

The diagrams used to illustrate the above definitions are called *Venn diagrams*, after John Venn (1834–1923), where a set is represented by a region in the plane.

Example 2.1.5. Let $A = \{1, 2, 3, 4\}$ and $B = \{3, 4, 5, 6\}$. Determine $A \cap B$, $A \cup B$, $A \setminus B$ and $B \setminus A$.

- $A \cap B$. 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 is not an element of B; 2 is not an element of B; 3 and 4 are elements of B. Thus $A \cap B = \{3, 4\}$.
- $A \cup B$. 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 $A \cup B = \{1, 2, 3, 4, 5, 6\}$.
- $A \setminus B$. We have to find the elements of A that do not belong to B. Read the elements of A from left-to-right comparing them with the elements of B: 1 does not belong to B; 2 does not 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 do not belong to A. This set is equal to $\{5, 6\}$.

Sets A and B are said to be *disjoint* if $A \cap B = \emptyset$.

Theorem 2.1.6 (Properties of Boolean operations)**.** *Let A, B and C be any sets.*

1. $A \cap (B \cap C) = (A \cap B) \cap C$. *Intersection is associative.*
2. $A \cap B = B \cap A$. *Intersection is commutative.*
3. $A \cap \emptyset = \emptyset = \emptyset \cap A$. *The empty set is the zero for intersection.*
4. $A \cup (B \cup C) = (A \cup B) \cup C$. *Union is associative.*
5. $A \cup B = B \cup A$. *Union is commutative.*
6. $A \cup \emptyset = A = \emptyset \cup A$. *The empty set is the identity for union.*
7. $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$. *Intersection distributes over union.*
8. $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$. *Union distributes over intersection.*

9. $A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$. *De Morgan's law part one.*

10. $A \setminus (B \cap C) = (A \setminus B) \cup (A \setminus C)$. *De Morgan's law part two.*

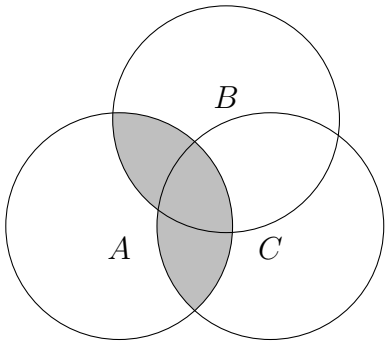
11. $A \cap A = A$. *Intersection is idempotent.*

12. $A \cup A = A$. *Union is idempotent.*

Proof. I shall just look at property (7) as an example of what can be done. To *illustrate* this property, we can use Venn diagrams. The Venn diagram for

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

is given below.



This is exactly the same as the Venn diagram for

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

It follows that

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

at least as far as Venn diagrams are concerned.

To *prove* property (7), we have to proceed more formally and use PL. We use the fact that

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

Our goal is to prove that

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

To do this, we have to prove that the set of elements belonging to the lefthand side is the same as the set of elements belonging to the righthand side. An

element x either belongs to A or it does not. Similarly, it either belongs to B or it does not, and it either belongs to C or it does not. Define p to be the statement '$x \in A$'. Define q to be the statement '$x \in B$'. Define r to be the statement '$x \in C$'. If p is true then x is an element of A, and if p is false then x is not an element of A. Using now the definitions of the Boolean operations, it follows that $x \in A \cap (B \cup C)$ precisely when the statement $p \wedge (q \vee r)$ is true. Similarly, $x \in (A \cap B) \cup (A \cap C)$ precisely when the statement $(p \wedge q) \vee (p \wedge r)$ is true. But these two statements have the same truth tables. It follows that an element belongs to the lefthand side precisely when it belongs to the righthand side. Consequently, the two sets are equal. □

The fact that $A \cap (B \cap C) = (A \cap B) \cap C$ means that we can just write $A \cap B \cap C$ unambiguously without brackets. Similarly, we can write $A \cup B \cup C$ unambiguously. This can be extended to any number of unions and any number of intersections.

We finish off with two definitions that will be important in the next section. Fix a set X. The set whose elements are all the subsets of X is called the *power set* of X and is denoted by $\mathsf{P}(X)$. It is important to remember that the power set of a set X contains both $\emptyset$ and X as elements.

Example 2.1.7. We find all the subsets of the set $X = \{a, b, c\}$ and so the power set of X. First there is the subset with no elements, the empty set. Then there are the subsets that contain exactly one element: $\{a\}, \{b\}, \{c\}$. Then the subsets containing exactly two elements: $\{a, b\}, \{a, c\}, \{b, c\}$. Finally, there is the whole set X. It follows that X has 8 subsets and so

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

Let $A \subseteq X$. Define $\overline{A} = X \setminus A$. This is called the *complement* of A. Observe that for this operation to be well-defined, we have to know the set X that we are working within.

Exercises 2.1

---

1. Let $A = \{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}$, $B = \{\spadesuit, \diamondsuit, \clubsuit, \heartsuit\}$ and $C = \{\spadesuit, \diamondsuit, \clubsuit, \heartsuit, \clubsuit, \diamondsuit, \heartsuit, \spadesuit\}$. Is it true or false that $A = B$ and $B = C$? Explain.

2. Find all subsets of the set $\{a, b, c, d\}$.

3. Let $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Write down the following subsets of X:

   (a) The subset A of even elements of X.

   (b) The subset B of odd elements of X.

   (c) $C = \{x\colon\ x \in X \text{ and } x \geq 6\}$.

   (d) $D = \{x\colon\ x \in X \text{ and } x > 10\}$.

   (e) $E = \{x\colon\ x \in X \text{ and } x \text{ is prime}\}$.

   (f) $F = \{x\colon\ x \in X \text{ and } (x \leq 4 \text{ or } x \geq 7)\}$.

4. Write down how many elements each of the following sets contains.

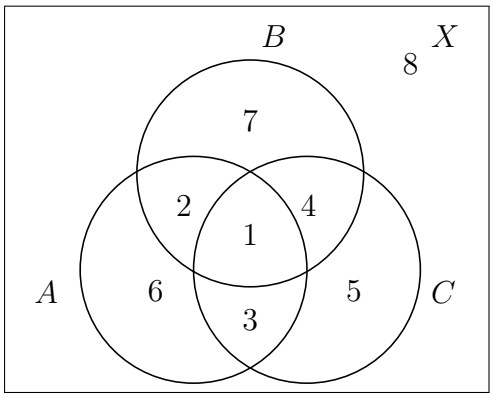
   (a) $\emptyset$.

   (b) $\{\emptyset\}$.

   (c) $\{\emptyset, \{\emptyset\}\}$.

   (d) $\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}$.

5. This question concerns the following diagram where $A, B, C \subseteq X$. Use Boolean operations to describe each of the eight regions. For example, region (1) is $A \cap B \cap C$.



6. Let $A, B, C \subseteq X$. Draw Venn diagrams to illustrate each of the following sets.

(a) $A \cup (B \cap C) \cup (\overline{A} \cap B) \cup (A \cap \overline{B} \cap C)$.

(b) Show that

$$(\overline{A} \cap \overline{B} \cap C) \cup (\overline{A} \cap B \cap C) \cup (A \cap \overline{B})$$

and

$$(A \cap \overline{B}) \cup (\overline{A} \cap C)$$

are equal.

(c) Show that

$$(A \cap B \cap C) \cup (\overline{A} \cap B \cap C) \cup \overline{B} \cup \overline{C}$$

and X are equal.

7. (a) Prove that

$$a \vee (b \wedge c) \vee (\neg a \wedge b) \vee (a \wedge \neg b \wedge c)$$

and

$$a \vee b$$

are logically equivalent.

(b) Prove that

$$(\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge c) \vee (a \wedge \neg b)$$

and

$$(a \wedge \neg b) \vee (\neg a \wedge c)$$

are logically equivalent.

(c) Prove that

$$(a \wedge b \wedge c) \vee (\neg a \wedge b \wedge c) \vee \neg b \vee \neg c$$

is a tautology.

8. What is the connection between Questions 6 and 7 above?

2.2 Boolean algebras

In 1854, George Boole wrote a book [5] where he tried to show that some of the ways in which we reason could be studied mathematically. This led, perhaps indirectly [18], to what are now called Boolean algebras and was also important in the development of propositional logic and set theory. To motivate the definition of Boolean algebras, I shall begin by listing some logical equivalences in propositional logic which were proved in Exercises 1.4. I shall use $\boldsymbol{t}$ to mean any tautology and $\boldsymbol{f}$ 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. $p \vee \boldsymbol{f} \equiv p$.
4. $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$.
5. $p \wedge q \equiv q \wedge p$.
6. $p \wedge \boldsymbol{t} \equiv p$.
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. $p \vee \neg p \equiv \boldsymbol{t}$.
10. $p \wedge \neg p \equiv \boldsymbol{f}$.

In this section we shall make the following substitutions in the above list:

PL	Boolean algebra
$\equiv$	$=$
$\vee$	$+$
$\wedge$	$\cdot$
$\neg$	$^-$
$\boldsymbol{t}$	1
$\boldsymbol{f}$	0

to get exactly the definition of a Boolean algebra. This will be an *algebraic system* rather than a *logical system*. It is adapted to dealing with truth tables and so with circuits, as we shall see.

2.2.1 Definition and examples

Formally, a *Boolean algebra* is defined by the following data $(B, +, \cdot, {}^{-}, 0, 1)$ 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) deals with the properties of $a \mapsto \bar{a}$, called *complementation*. On a point of notation, we shall usually write xy rather than $x \cdot y$. Because of (B1) and (B4), we can write sums and products without the need for brackets. We shall now describe three examples of Boolean algebras with, ironically, the last and simplest being the most important when it comes to circuit design.

Example 2.2.1. *The Lindenbaum algebra.* This is the Boolean algebra associated with PL. I have described it, in a rather cavalier fashion, as being obtained from PL by replacing $\equiv$ by $=$. How to do this in a mathematically legitimate way is quite involved so I shall only sketch out the idea here. Denote by **Prop** the set of all wff in propositional logic. The key is the behaviour of $\equiv$ on the set **Prop** as described by Proposition 1.4.13. The first three properties tell us that $\equiv$ is an *equivalence relation*. I shall not define these here but I shall describe what this means. For each wff A, we define the set

$$[A] = \{B \colon B \in \mathbf{Prop} \text{ and } B \equiv A\}.$$

These sets have an interesting property either $[A] \cap [B] = \emptyset$ or $[A] = [B]$. Define $\mathbb{L}$ to be the set of all these sets. We now make the following definitions

$$[A] \cdot [B] = [A \wedge B], \quad [A] + [B] = [A \vee B] \text{ and } \overline{[A]} = [\neg A].$$

These definitions do make sense. For example, if $[A] = [A']$ and $[B] = [B']$ then $[A \vee B] = [A' \vee B']$. If A is any tautology define $1 = [A]$ and if B is any contradiction define $0 = [B]$. Using Proposition 1.4.13, it is then possible to show (with some work) that with these definitions $\mathbb{L}$ really is a Boolean algebra. The Lindenbaum algebra is used in more advanced work to prove results in PL. This algebra is named after Adolf Lindenbaum (1834–1923).

Example 2.2.2. *The Power set Boolean algebra.* This is an important class of examples of Boolean algebras which arise naturally from set theory. Let X be a set and recall that $\mathsf{P}(X)$ is the set of all subsets of X. We now define the following operations on $\mathsf{P}(X)$.

BA	Power sets
$+$	$\cup$
$\cdot$	$\cap$
$^-$	$^-$
1	X
0	$\emptyset$

With the help of Theorem 2.1.6, it is straightforward to prove that

$$(\mathsf{P}(X), \cup, \cap, ^-, \emptyset, X)$$

is a Boolean algebra. In fact, it can be proved that the finite Boolean algebras always look like power set Boolean algebras.

Example 2.2.3. *The 2-element Boolean algebra* $\mathbb{B}$. This is the Boolean algebra we shall use in circuit design. It is defined as follows. Put $\mathbb{B} = \{0, 1\}$. We define operations $^-$, $\cdot$, and $+$ by means of the following tables. They are the same as the truth table 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

You can check that $\mathbb{B}$ is essentially the same as the Boolean algebra $\mathsf{P}(\{0, 1\})$.

2.2.2 Algebra in a Boolean algebra

Boolean algebra is similar to but also different from the kind of algebra you learnt at school. Just how different is illustrated by the following results. The results we prove will be true in all Boolean algebras because they are proved from the Boolean algebra axioms.

Proposition 2.2.4. *Let B be a Boolean algebra and let $a, b \in B$.*

1. $a^2 = a \cdot a = a$. *Idempotence.*
2. $a + a = a$. *Idempotence.*
3. $a \cdot 0 = 0$. *The element 0 is the zero for multiplication.*
4. $1 + a = 1$. *The element 1 is the zero for addition.*
5. $a = a + a \cdot b$. *Absorption law.*
6. $a + b = a + \bar{a} \cdot b$. *Absorption law.*

Proof. (1)

$$\begin{aligned} a &= a \cdot 1 \text{ by (B6)} \\ &= a \cdot (a + \bar{a}) \text{ by (B9)} \\ &= a \cdot a + a \cdot \bar{a} \text{ by (B7)} \\ &= a^2 + 0 \text{ by (B10)} \\ &= a^2 \text{ by (B3).} \end{aligned}$$

(2) is proved in a very similar way to (1). Use the fact that the Boolean algebra axioms come in pairs where $\cdot$ and $+$ are interchanged and 0 and 1 are interchanged. This is an aspect of what is called *duality*.

(3)

$$\begin{aligned} a \cdot 0 &= a \cdot (a \cdot \bar{a}) \text{ by (B10)} \\ &= (a \cdot a) \cdot \bar{a} \text{ by (B4)} \\ &= a \cdot \bar{a} \text{ by (1) above} \\ &= 0 \text{ by (B10).} \end{aligned}$$

(4) The dual proof to (3).

(5)

$$\begin{aligned} a + a \cdot b &= a \cdot 1 + a \cdot b \text{ by (B6)} \\ &= a \cdot (1 + b) \text{ by (B7)} \\ &= a \cdot 1 \text{ by (4) above} \\ &= a \text{ by (B6).} \end{aligned}$$

(6)

$$\begin{aligned} a + b &= a + 1b \text{ by (B6)} \\ &= a + (a + \bar{a})b \text{ by (B9)} \\ &= a + ab + \bar{a}b \text{ by (B7) and (B5)} \\ &= a + \bar{a}b \text{ by (5) above} \end{aligned}$$

□

These properties can be used to simplify Boolean expressions.

Example 2.2.5. Simplify

$$x + yz + \bar{x}y + x\bar{y}z.$$

At each stage in our argument we are explicit about what properties we are using. I shall use associativity of addition throughout to avoid too many

brackets.

$$\begin{aligned} x + yz + \bar{x}y + x\bar{y}z &= (x + \bar{x}y) + yz + x\bar{y}z \text{ by commutativity} \\ &= (x + y) + yz + x\bar{y}z \text{ by absorption} \\ &= x + (y + yz) + x\bar{y}z \text{ by associativity} \\ &= x + y + x\bar{y}z \text{ by absorption} \\ &= x + (y + \bar{y}(xz)) \text{ by commutativity and associativity} \\ &= x + (y + xz) \text{ by absorption} \\ &= (x + xz) + y \text{ by commutativity and associativity} \\ &= x + y \text{ by absorption.} \end{aligned}$$

We may now use Venn diagrams to illustrate certain Boolean expressions and so help us in simplify them.

Example 2.2.6. We return to the Boolean expression

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

that we simplified above. We interpret this in set theory where the Boolean variables x, y, z are replaced by sets X, Y, Z. Observe that $\bar{x}y$ translates into $Y \setminus X$. Thus the above Boolean expression is the set

$$X \cup (Y \cap Z) \cup (Y \setminus X) \cup ((X \cap Z) \setminus Y).$$

If you draw the Venn diagram of this set you get exactly $X \cup Y$. This translates into the Boolean expression $x + y$ which is what we obtained when we simplified the Boolean expression.

A more complex example of proving results about Boolean algebras is included below. It is important to at least know the result, if not the proof. It shows that the analogue of De Morgan's laws hold in any Boolean algebra.

Proposition 2.2.7. *Let B be a Boolean algebra.*

1. *If $a + b = 1$ and $ab = 0$ then $b = \bar{a}$.*
2. *$\bar{\bar{a}} = a$.*
3. *$\overline{(a + b)} = \bar{a}\bar{b}$.*

4. $\overline{ab} = \bar{a} + \bar{b}$.

Proof. (1) We are given that $a + b = 1$ and $ab = 0$. Our goal is to show that $b = \bar{a}$. We know that $a + \bar{a} = 1$ by (B9). Thus $a + b = a + \bar{a}$. Multiply both sides of the above equation on the left by b to get $b(a + b) = b(a + \bar{a})$. This gives $ba + b^2 = ba + b\bar{a}$ by (B7). Now $ba = ab$ by the commutativity of multiplication and $b^2 = b$. It follows that $ab + b = ab + \bar{a}b$. But $ab = 0$ by assumption. It follows that

$$b = \bar{a}b.$$

Now $a + b = 1$ by assumption. Multiply both sides of this equation on the left by $\bar{a}$ to get $\bar{a}(a+b) = \bar{a}1 = \bar{a}$ by (B6). By (B7), we have that $\bar{a}a+\bar{a}b = \bar{a}$. By (B10), we have that $\bar{a}a = 0$. It follows that

$$\bar{a} = \bar{a}b.$$

We have therefore proved that $b = \bar{a}$.

(2) By (B9), we have that $\bar{a} + \bar{\bar{a}} = 1$ and by (B10), we have that $\bar{a}\bar{\bar{a}} = 0$. But we also have by (B9) and (B10) that $a+\bar{a} = 1 = \bar{a}+a$ and $a\bar{a} = 0 = \bar{a}a$. By part (1), we deduce that $\bar{\bar{a}} = a$.

(3) By (B9) and (B10), we have that $(a + b) + \overline{(a + b)} = 1$ and $(a + b)\overline{(a + b)} = 0$. We now calculate

$$(a + b) + \bar{a}\bar{b} = (a + \bar{a}\bar{b}) + b = a + (\bar{b} + b) = a + 1 = 1$$

and

$$\bar{a}\bar{b}(a + b) = \bar{a}\bar{b}a + \bar{a}\bar{b}b = \bar{b}\bar{a}a + \bar{a}\bar{b}b = 0 + 0 = 0.$$

It follows by uniqueness that $\overline{(a + b)} = \bar{a}\bar{b}$.

(4) The proof of this case is similar to that in part (3). □

Exercises 2.2

1. Prove that $b + b = b$ for all $b \in B$ in a Boolean algebra.
2. Prove that $0b = 0$ for all $b \in B$ in a Boolean algebra.
3. Prove that $1 + b = 1$ for all $b \in B$ in a Boolean algebra.

4. Prove the following absorption laws in any Boolean algebra.

   (a) $a(a + b) = a$.

   (b) $a(\bar{a} + b) = ab$.

5. Simplify each of the following Boolean algebra expressions as much as possible. You might find it useful to draw Venn diagrams first.

   (a) $xy + x\bar{y}$.

   (b) $x\bar{y}\bar{x} + xy\bar{x}$.

   (c) $xy + x\bar{y} + \bar{x}y$.

   (d) $xyz + xy\bar{z} + x\bar{y}z + x\bar{y}\bar{z}$.

   (e) $\bar{x}\bar{y}z + x\bar{y}\bar{z} + x\bar{y}z$.

   (f) $x + yz + \bar{x}y + \bar{y}xz$.

6. Let $B = \{1, 2, 3, 5, 6, 10, 15, 30\}$. Let $m, n \in B$. Define $m \cdot n$ to be the greatest common divisor of m and n. Define $m + n$ to be the lowest common multiple of m and n. Define $\mathbf{0} = 1$ and define $\mathbf{1} = 30$.

   (a) Show that with these definitions B is a Boolean algebra.

   (b) How is this Boolean algebra related to the Boolean algebra of all subsets of the set $\{2, 3, 5\}$?

   (c) How do you compute complements of elements?

7. Let $X = \{0, 1\}$ be *any* two-element Boolean algebra. Prove that $+$ must behave like $\vee$, that $\cdot$ must behave like $\wedge$ and that complementation must behave like negation.

2.3 Combinational circuits

In this section, we shall describe how Boolean algebra can be used to help design the kinds of circuits important in building computers. The discovery that Boolean algebras could be used in this way is due to to Claude Shannon (1916–2001) in his MSc thesis of 1937. Shannon went on to lay the foundations of information theory; his work underpins the role of cryptography used in protecting sensitive information.

A computer circuit is a physical, not a mathematical, object. It is constructed out of transistors, resistors and capacitors and the like and powered by electricity. We shall not be dealing with such real circuits in this course. Instead, we shall describe some simple mathematical models which are idealizations of such circuits. Nevertheless, the models we describe are the basis of designing real circuits and their theory is described in all books on digital electronics. Currently, all computers are constructed using binary logic, meaning that their circuits operate using two values of some physical property, such as voltage. This feature will be modelled by the two elements of the Boolean algebra $\mathbb{B}$. We shall work with mathematical expressions involving letters such as x, y, z; these are *Boolean variables* meaning that $x, y, z \in \mathbb{B}$. Circuits come in two types: *combinational circuits*, which have no internal memory, and *sequential circuits*, which do. In this section, we describe combinational circuits which are, to a first mathematical approximation, nothing other than *Boolean functions* $f\colon \mathbb{B}^m \to \mathbb{B}^n$.[3]

2.3.1 How gates build circuits

Let C be a combinational circuit with m input wires and n output wires. We can think of C as consisting of n combinatorial circuits $C_1, \ldots, C_n$ each having m input wires but only one output wire each. The combinational circuit C_i tells us about how the ith output of the circuit C behaves with respect to inputs. Thus it is enough to describe combinational circuits with m inputs and 1 output. Such a circuit is said to describe a Boolean function $f\colon \mathbb{B}^m \to \mathbb{B}$ where $\mathbb{B}^m$ represents all the possible 2^m inputs. Boolean functions with one output are described by means of an *input/output table*. Here is an example

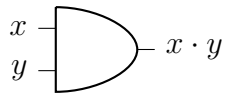
[3]Real combinational circuits have features that are not captured by our simple model. These are discussed in books on digital electronics.

of such a table.

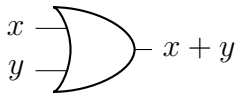
x	y	z	$f(x,y,z)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Our goal is to show that any such Boolean function can be constructed from certain simpler Boolean functions called *gates*. There are a number of different kinds of gates but we begin with the three basic ones.

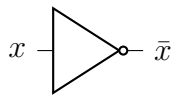
The *and-gate* is the function $\mathbb{B}^2 \to \mathbb{B}$ defined by $(x, y) \mapsto x \cdot y$. We use the following symbol to represent this function.



The *or-gate* is the function $\mathbb{B}^2 \to \mathbb{B}$ defined by $(x, y) \mapsto x + y$. We use the following symbol to represent this function.

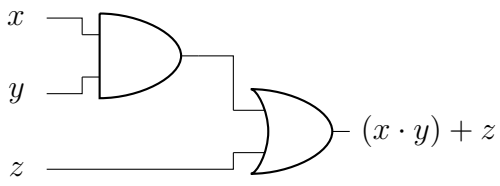


Finally, the *not-gate* is the function $\mathbb{B} \to \mathbb{B}$ defined by $x \mapsto \bar{x}$. We use the following symbol to represent this function.

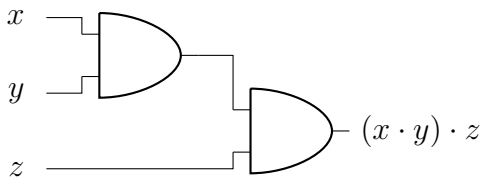


Diagrams constructed using gates are called *circuits* and show how Boolean functions can be computed as we shall see. Such mathematical circuits can be converted into physical circuits with gates being constructed from simpler circuit elements called transistors which operate like electronic switches.

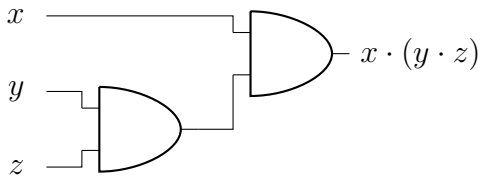
Example 2.3.1. Here is a simple circuit.



Example 2.3.2. Because of the associativity of $\cdot$, the circuit



and the circuit



compute the same function. Similar comments apply to the operation $+$.

The main theorem in circuit design is the following. It is nothing other than the Boolean algebra version of Theorem 1.7.2, the result that says that every truth function arises as the truth table of a wff.

Theorem 2.3.3 (Fundamental theorem of circuit design)**.** *Every Boolean function $f\colon \mathbb{B}^m \to \mathbb{B}$ can be constructed from and-gates, or-gates and not-gates.*

Proof. Assume that f is described by means of an input/output table. We deal first with the case where f is the constant function to 0. In this case,

$$f(x_1, \ldots, x_m) = (x_1 \cdot \overline{x_1}) \cdot x_2 \cdot \ldots \cdot x_m.$$

Next we deal with the case where the function f takes the value 1 exactly once. Let $\mathbf{a} = (a_1, \ldots, a_m) \in \mathbb{B}^m$ be such that $f(a_1, \ldots, a_m) = 1$. Define $\mathbf{m} = y_1 \cdot \ldots \cdot y_m$, called the *minterm* associated with $\mathbf{a}$, as follows:

$$y_i = \begin{cases} x_i & \text{if } a_i = 1 \\ \overline{x_i} & \text{if } a_i = 0. \end{cases}$$

Then $f(\mathbf{x}) = y_1 \cdot \ldots \cdot y_m$. Finally, we deal with the case where the function f is none of the above. Let the inputs where f takes the value 1 be $\mathbf{a}_1, \ldots, \mathbf{a}_r$, respectively. Construct the corresponding minterms $\mathbf{m}_1, \ldots, \mathbf{m}_r$, respectively. Then

$$f(\mathbf{x}) = \mathbf{m}_1 + \ldots + \mathbf{m}_r.$$

□

Example 2.3.4. We illustrate the proof of Theorem 2.3.3 by means of the following input/output table.

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

The three elements of $\mathbb{B}^3$ where f takes the value 1 are $(0,0,1)$, $(0,1,0)$ and $(1,0,0)$. The minterms corresponding to each of these inputs are $\bar{x} \cdot \bar{y} \cdot z$, $\bar{x} \cdot y \cdot \bar{z}$ and $x \cdot \bar{y} \cdot \bar{z}$, respectively. It follows that

$$f(x,y,z) = \bar{x} \cdot \bar{y} \cdot z + \bar{x} \cdot y \cdot \bar{z} + x \cdot \bar{y} \cdot \bar{z}.$$

We could if we wished attempt to simplify this Boolean expression. This becomes important when we wish to convert it into a circuit.

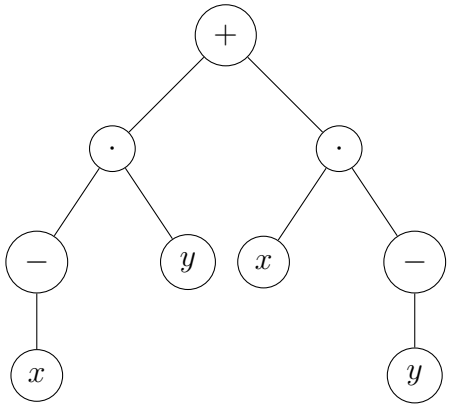
Example 2.3.5. The input/output table below

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

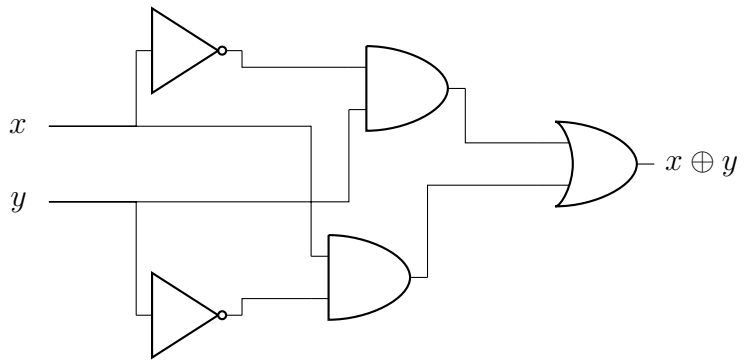
defines *exclusive or* or *xor*. By Theorem 2.3.3, we have that

$$x \oplus y = \bar{x} \cdot y + x \cdot \bar{y}.$$

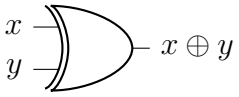
We may describe this by means of a parse tree just as we did in the case of wff.



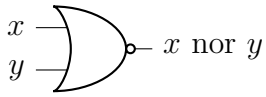
This parse tree may be converted into a circuit in a series of stages but requires two further circuit elements. First, we require two input wires: one labelled x and one labelled y. But in the parse tree x occurs twice and y occurs twice. We therefore need a new circuit element called *fanout*. This has one input and then branches with each branch carrying a copy of the input. In this case, we need one fanout with input x and two outward branches and another with input y and two outward branches. In addition, we need to allow wires to cross but not to otherwise interact. This is called *interchange* and forms the second additional circuit element we need. Finally, we replace the Boolean symbols by the corresponding gates and rotate the diagram ninety degrees clockwise so that the inputs come in from the left and the output emerges from the right. We therefore obtain the following circuit diagram.



For our subsequent examples, it is convenient to abbreviate the circuit for xor by means of a single circuit symbol called an *xor-gate*.



Example 2.3.6. There are two further gates that you are likely to encounter. The first is the *nor-gate*

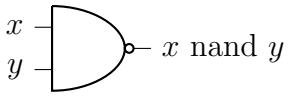


This has the following input/output table

x	y	x nor y
0	0	0
0	1	0
1	0	0
1	1	1

and is just the Boolean algebra version of the PL binary connective **nor**.

The second is the *nand-gate*



This has the following input/output table

x	y	x nor y
0	0	0
0	1	1
1	0	1
1	1	1

and is just the Boolean algebra version of the PL binary connective **nand**.

The following theorem is nothing other than the Boolean algebra version of Proposition 1.6.4.

Theorem 2.3.7 (Nor-gates and nand-gates)**.**

1. *Every Boolean function* $f\colon \mathbb{B}^m \to \mathbb{B}$ *can be constructed from only nor-gates.*
2. *Every Boolean function* $f\colon \mathbb{B}^m \to \mathbb{B}$ *can be constructed from only nand-gates.*

2.3.2 A simple calculator

The goal of this section is to show that we can actually build something useful with the theory we have developed so far: we construct, in stages, a circuit that will add two 4-bit binary numbers together. In everyday life, we write numbers down using base 10 but in computers, it is more natural to treat numbers as being in base 2 or *binary*. We begin by explaining what this means.

Recall that a string is simply an ordered sequence of symbols. If the symbols used in the string are taken only from $\{0, 1\}$, the set of *binary digits*, then we have a *binary string*. All kinds of data can be represented by binary strings but here we are only interested in the way they can be used to encode natural numbers. The key idea is that every natural number can be represented as a sum of powers of two. How to do this is illustrated by the following example. Recall that $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, ...

Example 2.3.8. Write 316 as a sum of powers of 2.

- We first find the highest power of 2 that is less than or equal to our number. We see that $2^8 < 316$ but $2^9 > 316$. We can therefore write $316 = 2^8 + 60$.

- We now repeat this procedure with 60. We find that $2^5 < 60$ but $2^6 > 60$. We can therefore write $60 = 2^5 + 28$.

- We now repeat this procedure with 28. We find that $2^4 < 28$ but $2^5 > 28$. We can therefore write $28 = 2^4 + 12$.

- We now repeat this procedure with 12. We find that $2^3 < 12$ but $2^4 > 12$. We can therefore write $12 = 2^8 + 4$. Of course $4 = 2^2$.

It follows that

$$316 = 2^8 + 2^5 + 2^4 + 2^3 + 2^2,$$

a sum of powers of two.

Once we have written a number as a sum of powers of two we can encode that information as a binary string. How to do so is illustrated by the following example that continues the one above.

Example 2.3.9. We have that

$$316 = 2^8 + 2^5 + 2^4 + 2^3 + 2^2.$$

We now set up the following table

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	1	1	1	1	0	0

which includes all powers of two up to and including the largest one used. We say that the binary string

$$100111100$$

is the *binary representation* of the number 316.

Given a number written in binary it is a simple matter to convert it back into base ten.

Example 2.3.10. We show how to convert the number 110010 written in base 2 into a number written in base 10. The first step is to draw up a table of powers of 2.

2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	0	1	0

We now add together the powers of 2 which correspond to a '1'. This is just $2^5 + 2^4 + 2^1 = 50$ which is the corresponding number in decimal.

We have shown how every natural number can be written in base 2. In fact, all of arithmetic can be carried out working solely in base 2 although we shall only need to describe how to do addition. The starting point is to consider how to add two one-bit binary numbers together. The table below shows how this is done.

		carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

However, this is not quite enough to enable us to add two arbitrary binary numbers together. The algorithm for doing this is the same as in base 10

except that you carry 2 rather than carry 10. Because of the carries you actually need to know how to add three binary digits rather than just two. The table below shows how.

			carry	sum
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

How we use the above table to add two binary numbers together is illustrated in the following example.

Example 2.3.11. We calculate $11 + 1101$ in binary using the second of the two tables above. We first pad out the first string with 0s to get 0011. We now write the two binary numbers one above the other.

0	0	1	1
1	1	0	1
			0

Now work from right-to-left a column at a time adding up the digits you see and making any necessary carries. Observe that in the first column on the right there is no carry but it is more helpful, as we shall see, to think of this as a 0 carry. Here is the sequence of calculations.

	0	0	1	1
	1	1	0	1
				0
			1	0

	0	0	1	1
	1	1	0	1
			0	**0**
		1	1	0

	0	0	1	1
	1	1	0	1
		0	**0**	**0**
	1	1	1	0

	0	0	1	1
	1	1	0	1
1	**0**	**0**	**0**	**0**
1	1	1	1	0

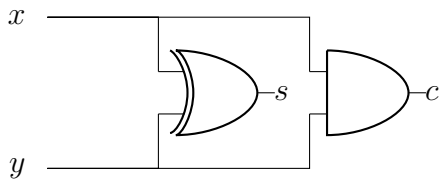
We find that the sum of these two numbers is 10000.

We shall describe in stages how to build a circuit that will add up two 4-bit numbers in binary.

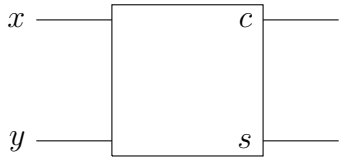
Example 2.3.12. Our first circuit is known as a *half-adder* which has two inputs and two outputs and is defined by the following input/output table.

x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

This treats the input x and y as numbers in binary and then outputs their sum. Observe that $s = x \oplus y$ and $c = x \cdot y$. Thus we may easily construct a circuit that implements this function.



In what follows, it will be useful to have a single symbol representing a half-adder. We use the following.

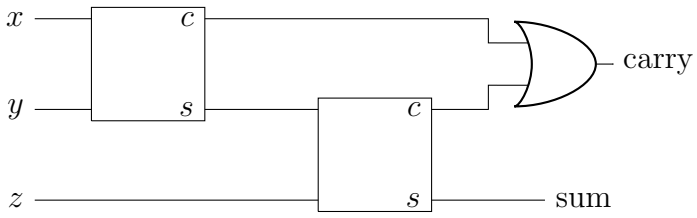


But as we have seen, the half-adder will not be quite enough for what we need.

Example 2.3.13. Our next circuit is known as a *full-adder* which has three inputs and two outputs and is defined by the following input/output table.

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

This treats the three inputs as numbers in binary and adds them together. The following circuit realizes this behaviour using two half-adders completed with an or-gate.



To understand why this circuit works, I need to introduce some notation. I shall write $\text{sum}(u, v)$ to mean the sum digit obtained when the two 1-bit binary numbers u and v are added. I shall write $\text{carry}(u, v)$ to mean the carry digit obtained when the two 1-bit binary numbers u and v are added. The sum-digit that results from adding x, y and z is

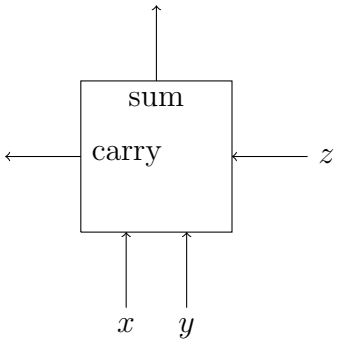
$$\text{sum}(\text{sum}(x, y), z).$$

The carry-digit that results from adding x, y and z is

$$\text{carry}(x, y) + \text{carry}(\text{sum}(x, y), z)$$

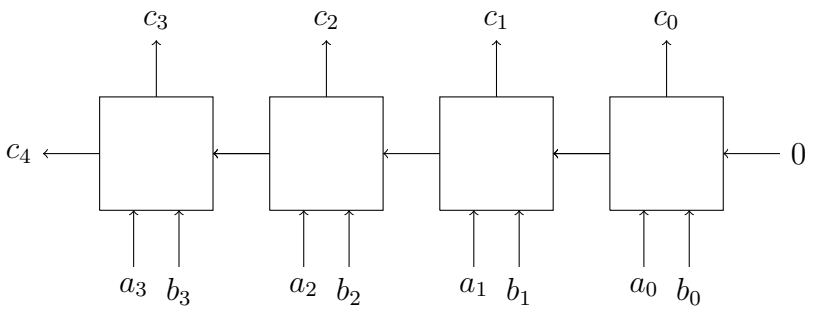
since $\text{carry}(x, y)$ and $\text{carry}(\text{sum}(x, y), z)$ are never both equal to 1.

We shall represent the full-adder by means of the following box-diagram.



We now have the ingredients we need to build a circuit that will add together two 4-bit binary numbers.

Example 2.3.14. Full-adders are the building blocks from which all arithmetic computer circuits can be built. Specifically, suppose that we want to add two four bit binary numbers together where we pad the numbers out by adding 0 at the front if necessary. Denote them by $m = a_3a_2a_1a_0$ and $n = b_3b_2b_1b_0$. The sum $m + n$ in base 2, which will be denoted by $m+n = c_4c_3c_2c_1c_0$, is computed in a similar way to calculating a sum in base 10. Thus first calculate $a_0 + b_0$, write down the sum bit, c_0, and pass any carry to be added to $a_1 + b_1$ and so on. Although $a_0 + b_0$ can be computed by a half-adder subsequent additions may require the addition of three bits because of the presence of a carry bit. For this reason, we actually use four full-adders joined in series with the rightmost full-adder having one of its inputs set to 0.



One point to observe in all this is that we did not use Theorem 2.3.3 directly. That guarantees that we can build any combinational circuit we want but to do so using the method employed in its proof would be prohibitively complicated except in small cases. Instead, we designed the circuit in stages taking what we constructed at one stage as a building block to be used in the next.

2.3.3 Transistors

The hardware of a computer is the part that we tend to notice first. It is also the part that most obviously reflects advances in technology. The single most important component of the hardware of a computer is called a *transistor*. Computers contain billions of transistors. In 2017, for example, Wikipedia claimed that there are commercially available chips containing $7 \cdot 2$ billion transistors. There is even an empirical result, known as *Moore's law*, which

says the number of transistors in integrated circuits doubles roughly every 2 years.

Transistors were invented in the 1940s and are constructed from semiconducting materials, such as silicon. The way they work is quite complex and depends on the quantum-mechanical properties of semiconductors, but what they *do* is very simple. The basic function of a transistor is to amplify a signal. A transistor is constructed in such a way that a weak input signal is used to control a large output signal and so achieve amplification of the weak input. An extreme case of this behaviour is where the input is used to turn the large input on or off. That is, where the transistor is used as an electronic switch. It is this function which is the most important use of transistors in computers. From a mathematical point of view, a transistor can be regarded as a device with one input, one control and one output.

- If 0 is input to the control then the switch is closed and the output is equal to the input.
- If 1 is input to the control then the switch is open and the output is 0 irrespective of the input.

A transistor on its own doesn't appear to accomplish very much, but more complicated behaviour can be achieved by connecting transistors together into circuits. Because of De Morgan's laws (for Boolean algebras), every Boolean expression is equal to one in which only $\cdot$ and $^-$ appear. We shall prove that and-gates and not-gates can be constructed from transistors and so we will have proved that every combinational circuit can be constructed from transistors.

I shall regard the transistor as a new Boolean operation that I shall write as $x \,\square\, y$. This is certainly not standard notation (there is none) but we only need this notation in this section. Its input/output behaviour is described by the following table.

x	y	$x \,\square\, y$
0	0	0
1	0	1
0	1	0
1	1	0

Observe that $x \,\square\, y \neq y \,\square\, x$. Clearly, $x \,\square\, y = x \cdot \bar{y}$. We now carry out a couple of calculations.

1. $1 \square y = 1\bar{y} = \bar{y}$. Thus by fixing $x = 1$ we can negate y.
2. Observe that $x \cdot y = x \cdot \bar{\bar{y}}$. Thus

$$x \cdot y = x \square \bar{y} = x \square (1 \square y).$$

We have therefore proved the following.

Theorem 2.3.15 (The fundamental theorem of transistors)**.** *Every combinational circuit can be constructed from transistors.*

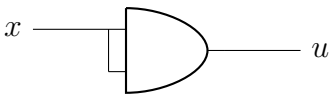
Exercises 2.3

1. Construct circuits using and- or- and not-gates for each of the following three input/output behaviours.

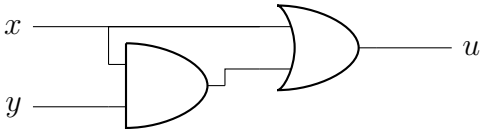
x	y	z	$f(x,y,z)$	$g(x,y,z)$	$h(x,y,z)$
0	0	0	0	1	0
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	0	0	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	1	0

2. Simplify each of the following combinational circuits as much as possible.

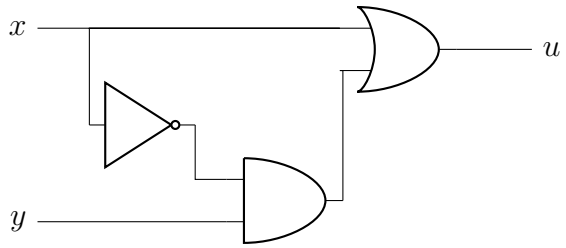
(a)



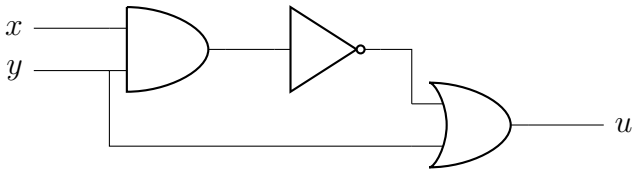
(b)



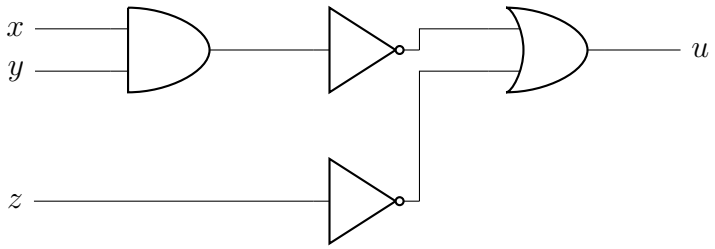
(c)



3. The following diagram shows a circuit with two inputs and one output. Write the output u as a Boolean expression in terms of the inputs x and y.



4. The following diagram shows a circuit with three inputs and one output. Write the output u as a Boolean expression in terms of the inputs x, y and z.



5. Convert the following numbers into binary.

   (a) 10.

   (b) 42.

   (c) 153.

   (d) 2001.

6. Convert the following binary numbers into decimal.

(a) 111.

(b) 1010101.

(c) 111000111.

7. Carry out the following additions in binary.

   (a) $11 + 11$.

   (b) $10110011 + 1100111$.

   (c) $11111 + 11011$.

8. Show how to construct a nor-gate from transistors.

9. A *Fredkin gate* has three inputs, a, b, c and three outputs, a', b', c'. It behaves as follows. If $c = 0$ then $a' = a$ and $b' = b$. If $c = 1$ then $a' = b$ and $b' = a$.

   (a) Draw up an input/output table for this gate.

   (b) What happens if you connect two such gates in series?

   (c) Write down Boolean expressions for a', b', c' if $a = 0$.

   (d) Write down Boolean expressions for a', b', c' if $a = 1$ and $b = 0$.

   (e) Write down Boolean expressions for a', b', c' if $c = 1$.

10. A *Toffoli gate* has three inputs a, b, c and three outputs a', b', c'. It behaves as follows: $a' = a$, $b' = b$ and $c' = ab \oplus c$.

    (a) Draw up an input/output table for this gate.

    (b) What happens if you connect two such gates in series?

    (c) Write down Boolean expressions for the output when $c = 1$.

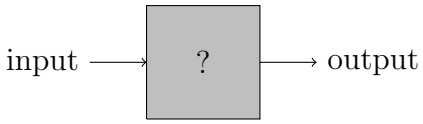
    (d) Write down the output when $a = 1$ and $c = 0$.

2.4 *Sequential circuits*

Suppose that we want to add three 4-bit binary numbers together. We can use the combinatorial circuit of the previous section to calculate the sum of two such binary numbers but then we would have to remember that result and add it to the third 4-bit binary number to calculate the total sum of the three

numbers. It follows that to build a real calculator and, more expansively, a real computer, we shall need memory. But now time will play a role since if there is no time there can be no memory.

Example 2.4.1. I have drawn a picture below of a 'black box circuit'. It has one input wire and one output wire and your job is to figure out what it is doing. You are allowed to input bits and observe what comes out.



You begin by inputting a 0 and a 0 is output. You then input a 1 and a 0 comes out again. If this were a combinational circuit then you would be able to say now that the output was always 0. But, just to be sure, you input 0 again but to your surprise a 1 is output. Thus, whatever this circuit is, it is not a combinational one. Rather than continuing with further examples, I shall tell you exactly what thus circuit is doing: it always outputs the previous input. It is our first example of a sequential circuit since it has a very limited form of memory: an extreme short-term memory, if you like. This particular circuit is called a *delay* and we shall use it as a building block in making more complex sequential circuits.

The concept of *state* plays an important role in understanding circuits with memory. A state is simply a memory configuration. Here is an example.

Example 2.4.2. The diagram below shows an array.

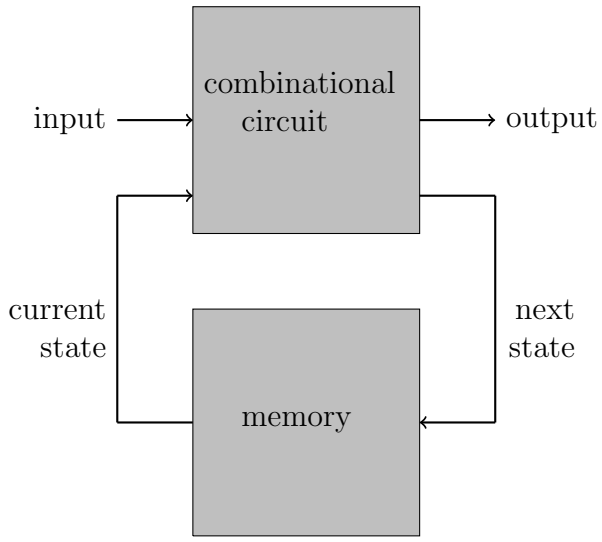
x_1	x_2
x_3	x_4

Each cell of the array can hold one bit. There are therefore $2^4 = 16$ such arrays

from $\begin{array}{|c|c|}\hline 0 & 0 \\ \hline 0 & 0 \\ \hline \end{array}$ to $\begin{array}{|c|c|}\hline 1 & 1 \\ \hline 1 & 1 \\ \hline \end{array}$.

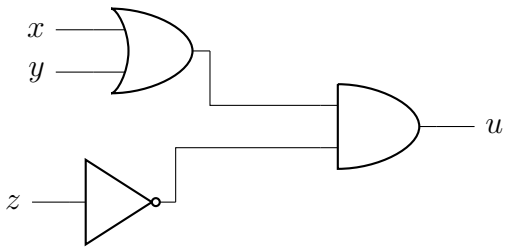
Each of these 16 possible arrays can be regarded as a particular memory or, as we shall say, a *state*. We encountered a concrete example of this situation where we used two coins to record 'memories'. In that case, there were four states: HH, HT, TH, TT.

A sequential circuit is a combinational circuit which has access to a memory. The following diagram is a schematic representation of how such circuits are to be regarded. We shall make this precise soon.



This diagram has one additional difference with a combinatorial circuit and that is *feedback*. This is represented by the loop that comes out of the combinational circuit, passes through the memory, and then goes back into the combinational circuit. Thus the current state and the input determine the next state and the output.

Example 2.4.3. We begin with an example to illustrate how the above schematic diagram is realized in practice. We start with the purely combinational circuit below.

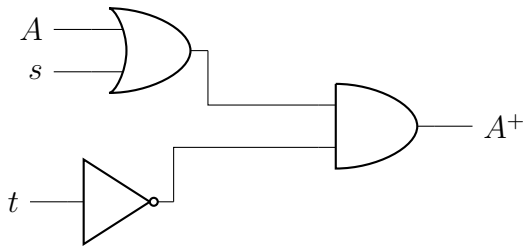


It is easy to check that $u = \bar{z} \cdot (x + y)$. Alternatively, we can construct an

input/output table to describe the behaviour of this combinational circuit.

x	y	z	u
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	0

I shall now relabel the above circuit and interpret its behaviour in a different way.



The Boolean variables s and t are now interpreted as the inputs whereas the Boolean variable A is interpreted as the *current state* and A^+ is interpreted as the *next state*. We can still use an input/output table to describe the behaviour of this circuit.

A	s	t	A^+
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	0

But now we can also use a *state transition diagram*. This is constructed directly from the above table. First, we describe the input alphabet. We

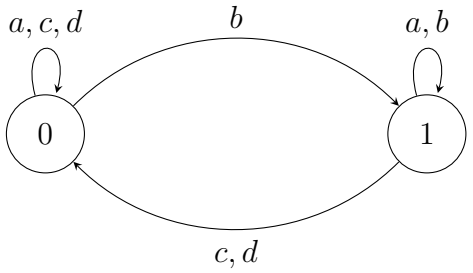
write the two inputs as a single vector

$$\begin{pmatrix} s \\ t \end{pmatrix}.$$

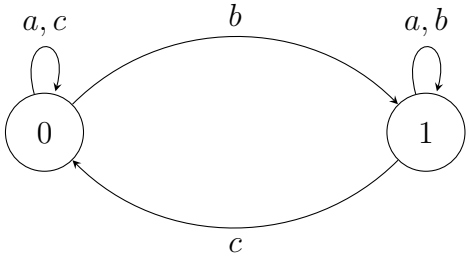
There are four possible values of this vector which we assign letters to for convenience

$$a = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad c = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad d = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Thus our input alphabet is $\{a, b, c, d\}$. There are two states corresponding to the two possible values of A. We now use the above table to construct the following finite-state automaton.

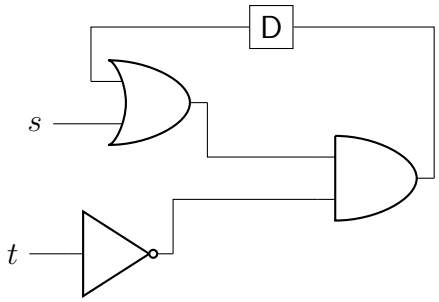


It is important to observe that the input/output table can be reconstituted from this diagram. If we remove the input d then our finite-state automaton becomes more symmetric.



Observe that a has no effect on the state but when c is applied, the automaton always goes into state 0 whereas when b is applied it always goes into state 1. Thus this automaton is capable of remembering one bit. Because of the way it functions, it is called a *flip-flop*.

I have used A and A^+ as *memory variables* where A^+ can be read as *next* A. However, we still don't quite have a real circuit. Given what we are doing, we could redraw it by connecting A^+ to A. The drawback of this is that you have to know how to handle the feedback correctly. The approach I prefer is to introduce an explicit delay gate.



This serves to act as a partition between the current state on the left hand side and the next state on the right hand side.

I shall mathematically analyse the above example but an understanding of this analysis is not needed to carry out computations or solve problems. It is included for completeness. Our starting point was a Boolean function $F\colon \mathbb{B}^3 \to \mathbb{B}$ which was described by means of a combinational circuit. In this case, $u = F(x, y, z) = \bar{z} \cdot (x + y)$. Essentially, such functions are timeless. In our sequential machine, y and z are replaced by s and t. But now, $s\colon \mathbb{N} \to \mathbb{B}$ and $t\colon \mathbb{N} \to \mathbb{B}$. The idea is that we have discretized time into $0, 1, 2, \ldots$. Thus $s(t)$ is the value of the s-input at time t, and similarly for $r(t)$. These are known functions. We now define a new function $A\colon \mathbb{N} \to \mathbb{B}$ by

$$A(t+1) = \overline{r(t)} \cdot (A(t) + s(t)).$$

This looks problematic because A appears on both sides of the above equation. But there is a difference. On the right hand side, we evaluate A at t whereas on the left hand side we evaluate A at $t + 1$. If we know the value of A at $t = 0$ then we can calculate the value of A at any subsequent time using the above equation. This is an example of defining a function by *recursion*. However, in our case recursion simplifies. This is because A, r, s can each only assume a finite number of values. This means that we can, in fact, forget about the time variable t and work instead with the operator $A \mapsto A^+$. Thus our equation above can be written simply

$$A^+ = \bar{r} \cdot (A + s)$$

which in turn can be encoded by means of a finite-state automaton. Let me now describe the general case of a sequential circuit. We are given input variables $\mathbf{x}$, output variables $\mathbf{y}$ and state variables $\mathbf{A}$ together with two Boolean equations

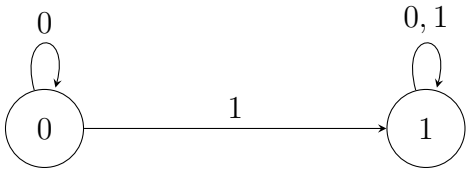
$$\mathbf{A}^+ = \mathsf{D}(\mathbf{x}, \mathbf{A}) \text{ and } \mathbf{y} = \mathsf{O}(\mathbf{x}, \mathbf{A})$$

where the first equation described the dynamics, that is how the states change with inputs, and the second equation describes the outputs, that is how the output is calculated on the basis of input and state.

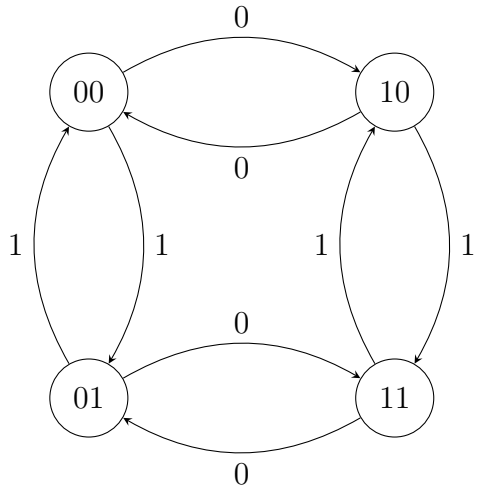
We shall now work towards a theorem that will show that sequential circuits and finite state automata are different ways of thinking about the same thing. We begin with a finite state automaton $\mathbf{A} = (Q, A, \delta)$ where $Q = \{q_1, \ldots, q_m\}$ is the set of states, $A = \{a_1, \ldots, a_n\}$ is the input alphabet and $\delta \colon Q \times A \to Q$ is the state transition function. At this stage, we do not assume that there is an initial state or terminal state. The first step is to label the states by means of binary strings. We shall do this in the most straightforward way and use *one-hot encoding*. This simply means that we use binary strings of length m and endode state q_i by the binary string of length m which consists solely of zeros except in position i where there is a 1. We now have to encode the input alphabet. Let r be the smallest natural number such that $n \leq 2^r$.

Exercises 2.4

1. In this question, denote the input by x and the memory variable by A. Construct (a) an input/output table for the following finite-state automaton and then (b) a Boolean equation expressing A^+ in terms of x and A and finally (c) a sequential circuit.



2. In this question, denote the input by x and the memory variables by A and B. Construct (a) an input/output table for the following finite-state automaton and then (b) Boolean equations expressing A^+ and B^+ in terms of x, A and B and finally (c) a sequential circuit.



Chapter 3

First-order logic

> *7. What we cannot speak about we must pass over in silence.* — Ludwig Wittgenstein.

PL is useful, as we have seen, but also very limited. The goal of this final part of the course is to add features to PL that will make it more powerful and more useful although there will be a price to pay in that the resulting system will be intrinsically harder to work with. We shall study what is known as *first-order logic* (FOL) and sometimes *predicate logic*. Essentially

$$\text{FOL} = \text{PL} + \text{predicates} + \text{quantifiers}.$$

This logic is the basis of applications of logic to CS, such as PROLOG. For mathematics, we have the following

$$\boxed{\text{Mathematics} = \text{Set theory} + \text{FOL}}$$

3.1 First steps

First-order logic is a more complex system than propositional logic so we shall proceed in a way that combines the informal with the formal.

3.1.1 Splitting the atom

Recall that a statement is a sentence which is capable of being either true or false. In PL, statements can only be analysed further in terms of the usual PL connectives until we get to atoms. The atoms cannot then be further analysed. We shall show how, in fact, atoms can be split by using some new ideas.

A *name* is a word that picks out a specific individual. For example, 2, π, Darth Vader are all names. Once something has been named, we can refer to it by that name.

At its simplest, a sentence can be analysed into a *subject* and a *predicate*. For example, the sentence 'grass is green' has 'grass' as its subject and 'is green' as its predicate. To make the nature of the predicate clearer, we might write '— is green' to indicate the slot into which a name could be fitted. Or, more formally, we could use a *variable* and write 'x is green'. We could symbolize this predicate thus '$G(x)$ = 'x is green'. We may replace the variable x by names to get honest statements that may or may not be true. Thus G(grass) is the statement 'grass is green' whereas G(cheese) is the statement 'cheese is green'. This is an example of a *1-place predicate* because there is exactly one slot into which a name can be placed to yield a statement. Other examples of 1-place predicates are: 'x is a prime' and 'x likes honey'.

There are also *2-place predicates*. For example,

$$P(x, y) = \text{ '}x \text{ is the father of } y\text{'}$$

is such a predicate because there are two slots that can be replaced by names to yield statements. Thus (spoiler alert) P(Darth Vader, Luke Skywalker) is true but P(Winnie-the-Pooh, Darth Vader) is false.

More generally, $P(x_1, \ldots, x_n)$ denotes an *n-place predicate* or an *n-ary* predicate. We say that the predicate has *arity* n. Here $x_1, \ldots, x_n$ are n variables that mark the n positions into which names can be slotted.

Most of the predicates we shall meet will have arities 1 or 2. But in theory there is no limit. Thus $F(x_1, x_2, x_3)$ is the 3-place predicate 'x_1 fights x_2 with a x_3'. By inserting names, we get the statement

$$F(\text{Luke Skywalker}, \text{Darth Vader}, \text{banana})$$

(deleted scene).

In FOL, names are called *constants* and are usually denoted by the rather more mundane $a, b, c, \ldots$ or $a_1, a_2, a_3 \ldots$. *Variables* are denoted by $x, y, z \ldots$ or $x_1, x_2, x_3, \ldots$. They have no fixed meaning but serve as place-holders into which names can be slotted.

An *atomic formula* is a predicate whose slots are filled with either variables or constants.

We now turn to the question of what predicates do. 1-place predicates describe sets[1]. Thus the 1-place predicate $P(x)$ describes the set

$$P = \{a\colon P(a) \text{ is true } \}$$

although this is usually written simply as

$$P = \{a\colon P(a)\}.$$

For example, if $P(x)$ is the 1-place predicate 'x is a prime' then the set it describes is the set of prime numbers. To deal with what 2-place predicates describe we need some new notation. An *ordered pair* is written (a, b) where a is the *first component* and b is the *second component*. Observe that we use round brackets. As the name suggests: order matters and so $(a, b) \neq (b, a)$, unlike in set notation. Let $P(x, y)$ be a 2-place predicate. Then it describes the set

$$P = \{(a, b)\colon P(a, b) \text{ is true } \}$$

which is usually just written

$$P = \{(a, b)\colon P(a, b)\}.$$

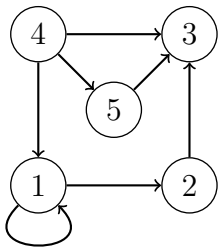
A set of ordered pairs where the elements are taken from some set X is called a *binary relation* on the set X. Thus 2-place predicates describe binary relations. We often denote binary relations by Greek letters. There is a nice graphical way to represent binary relations at least when the set they are defined on is not too big. Let ρ be a binary relation defined on the set X. We draw a *directed graph* or *digraph* of ρ. This consists of *vertices* labelled by the elements of X and *arrows* where an arrow is drawn from a to b precisely when $(a, b) \in \rho$.

[1]At least informally. There is the problem of Russell's paradox. How that can be dealt with, if indeed it can be, is left to a more advanced course.

Example 3.1.1. The binary relation

$$\rho = \{(1,1),(1,2),(2,3),(4,1),(4,3),(4,5),(5,3)\}$$

is defined on the set $X = \{1,2,3,4,5\}$. Its corresponding directed graph is



Example 3.1.2. Binary relations are very common in mathematics. Here are some examples.

1. The relation $x \mid y$ is defined on the set of natural numbers $\mathbb{N} = \{0,1,2,3,\ldots\}$ if x exactly divides y.

2. The relations $\leq$ (less than or equal to) and $<$ (strictly less than) are defined on $\mathbb{Z} = \{\ldots,-3,-2,-1,0,1,2,3,\ldots\}$.

3. The relation $\subseteq$ (is a subset of) is defined between sets.

4. the relation $\in$ (is an element of) is defined between sets.

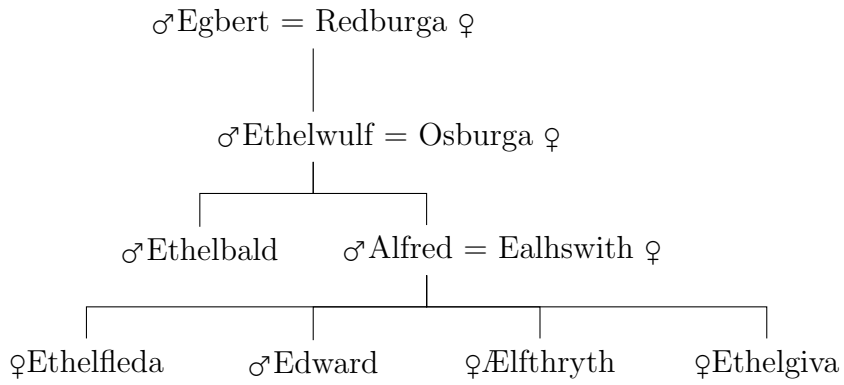
5. The relation $\equiv$ (logical equivalence) is defined on the set of wff.

For convenience, I shall only use 1-place predicates and 2-place predicates (and so also only sets and binary relations), but, in principle, there is no limit on the arities of predicates and we can study *ordered n-tuples* just as well as ordered pairs.

3.1.2 Structures

We begin with two examples.

Example 3.1.3. Here is part of the family tree of some Anglo-Saxon kings and queens.



We shall analyse the mathematics behind this family tree. First, we have a set D of kings and queens called the domain. This consists of eleven Tolkienesque elements

$$D = \left\{ \begin{array}{cccccc} \text{Egbert}, & \text{Redburga}, & \text{Ethelwulf}, & \text{Osburga}, & \text{Ethelbald}, & \text{Alfred}, \\ \text{Eahlswith}, & \text{Ethelfleda}, & \text{Edward}, & \text{Ælfthryth}, & \text{Ethelgiva} & \end{array} \right\}.$$

But we have additional information. Beside each name is a symbl ♂ or ♀ which means that that person is respectively male or female. This is just a way of defining two subsets of D:

$$M = \{\text{Egbert}, \text{Ethelwulf}, \text{Ethelbald}, \text{Alfred}, \text{Edward}\}$$

and

$$F = \{\text{Redburga}, \text{Osburga}, \text{Eahlswith}, \text{Ethelfleda}, \text{Ælfthryth}, \text{Ethelgiva}\}.$$

There are also two other peices of information. The most obvious are the lines linking one generation to the next. This is the binary relation defined by the 2-place predicate 'x is the parent of y'. It is the set of ordered pairs π. For example,

$$(\text{Ethelwulf}, \text{Alfred}), (\text{Osburga}, \text{Alfred}) \in \pi.$$

A little less obvious is the notation $=$ which stands for the binary relation defined by the 2-place predicate 'x is married to y'. It is the set of ordered pairs μ. For example,

$$(\text{Egbert}, \text{Redburga}), (\text{Redburga}, \text{Egbert}), (\text{Ethelwulf}, \text{Osburga}) \in \mu.$$

It follows that the information contained in the family tree is also contained in the following package

$$(D, M, F, \pi, \mu).$$

Example 3.1.4. This looks quite different at first from the previous example but is mathematically very closely related to it. Define

$$(\mathbb{N}, \mathbb{E}, \mathbb{O}, \leq, |)$$

where $\mathbb{E}$ and $\mathbb{O}$ are, respectively, the sets of odd and even natural numbers and $\leq$ and $|$ are binary relations.

We define a *structure* to consist of a non-empty set D, called the *domain*, together with a finite selection of subsets, binary relations, etc.

FOL is a language that will enable us to talk about structures.

3.1.3 Quantification: $\forall$, $\exists$

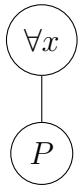
We begin with an example. Let $A(x)$ be the 1-place predicate 'x is made of atoms'. I want to say that 'everything is made from atoms'. In PL, I have no choice but to use infinitely many conjunctions

$$A(\text{jelly}) \wedge A(\text{ice-cream}) \wedge A(\text{blancmange}) \wedge \ldots$$

This is inconvenient. We get around this by using what is called the *universal quantifier* $\forall$. We write

$$(\forall x)A(x)$$

which should be read 'for all x, x is made from atoms' or 'for each x, x is made from atoms'. The variable does not have to be x, we have all of these $(\forall y)$, $(\forall z)$ and so on. As we shall see, you should think of $(\forall x)$ and its ilk as being a new unary connective. Thus



is the parse tree for $(\forall x)P$.

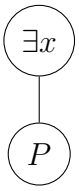
The corresponding infinite disjunction

$$P(a) \vee P(b) \vee P(c) \vee \ldots$$

is true when at least one of the terms of true. There is a corresponding *existential quantifier* $\exists$. We write

$$(\exists x)A(x)$$

which should be read 'there exists an x, such that $P(x)$ is true' or 'there is at least one x, such that $P(x)$ is true'. The variable does not have to be x, we have all of these $(\exists y)$, $(\exists z)$ and so on. You should also think of $(\exists x)$ and its ilk as being a new unary connective. Thus



is the parse tree for $(\exists x)P$.

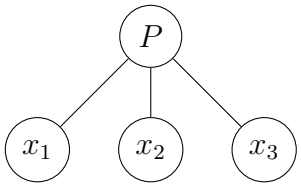
3.1.4 Syntax

A *first-order language* consists of a choice of predicate letters with given arities. Which ones you choose depends on what problems you are interested in. In addition to our choice of predicate letters, we also have, for free, our logical symboms carried forward from PL: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus$; we also have variables $x, y, \ldots, x_1, x_2, \ldots$; constants (names) $a, b \ldots a_1, a_2, \ldots$; and quantifiers $(\forall x)$, $(\forall y), \ldots (\forall x_1)$, $(\forall x_2)$, ... Recall that an *atomic formula* is a predicate letter with variables or constants inserted in all the available slots. We can now define a *formula* or *wff* in FOL.

1. All atomic formulae are wff.
2. If A and B are wff so too are $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$, $(A \leftrightarrow B)$, $(A \oplus B)$, and $(\forall x)A$, for any variable x and $(\exists x)A$ for any variable x.
3. All wff arise by repeated application of steps (1) or (2) a finite number of times.

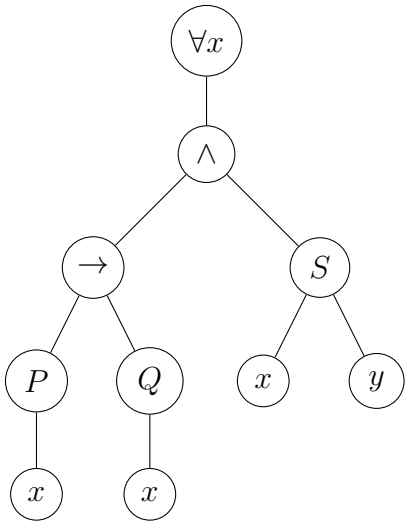
I should add that I will carry forward to FOL from PL the same conventions concerning the use of brackets without further comment.

We may adapt parse trees to FOL. An expression such as $P(x_1, x_2, x_3)$, for example, gives rise to the parse tree



The quantifiers $(\forall x)$ and $(\exists x)$ are treated as unary operators as we have seen. The leaves of the parse tree are either variables or constants.

Example 3.1.5. The formula $(\forall x)[(P(x) \to Q(x)) \wedge S(x, y)]$ has the parse tree



We now come to a fundamental, and quite difficult, definition. Choose any vertex v in a tree. The part of the tree, including v itself, that lies below v is clearly also a tree. It is called the *subtree determined by* v. In a parse tree, the subtree determined by an occurrence of $(\forall x)$ is called the *scope* of this occurrence of the quantifier. Similarly for $(\exists x)$. An occurrence of a variable x is called *bound* if it occurs within the scope of an occurrence of either $(\forall x)$ and $(\exists x)$. Otherwise, this occurrence is called *free*. Observe that the occurrence of x in both $(\forall x)$ and $(\exists x)$ is always bound.

Example 3.1.6. Consider the formula $(P(x) \to Q(y)) \to (\exists y)R(x, y)$. The scope of $(\exists y)$ is $R(x, y)$. Thus the x occurring in $R(x, y)$ is free and the y occurring in $R(x, y)$ is bound.

A formula that has a free occurrence of some variable is called *open* otherwise it is called *closed*. A closed wff is called a *sentence*. FOL is about sentences.

Example 3.1.7. Consider the wff

$$(\exists x)(F(x) \wedge G(x)) \rightarrow ((\exists x)F(x) \wedge (\exists x)G(x)).$$

I claim this is a sentence. There are three occurrences of $(\exists x)$. The first occurrence binds the x in $F(x) \wedge G(x)$. The second occurrence binds the occurrence of x in the second occurrence of $F(x)$. The third occurrence binds the occurrence of x in the second occurrence of $G(x)$. It follows that every occurrence of x in this wff is bound and there are no other variables. Thus the wff is closed as claimed.

I shall explain why sentences are the natural things to study once I have defined the semantics of FOL.

3.1.5 Semantics

Let L be a first-order language. For concreteness, suppose that it consists of two predicate letters where P is a 1-place predicate letter and Q is a 2-place predicate letter. An *interpretation* I of L is any structure (D, A, ρ) where D, the domain, is a non-empty set, $A \subseteq D$ is a subset and ρ is a binary relation on D. We interpret P as A and Q as ρ. Thus the wff $(\exists x)(P(x) \wedge Q(x, x))$ *is interpreted as* $(\exists x \in D)((x \in A) \wedge ((x, x) \in \rho))$. Under this interpretation, every sentence S in the language makes an assertion about the elements of D using A and ρ. We say that I *is a model of* S, written $I \vDash S$, if S is true when interpreted in this way in I. A sentence S is said to be *universally (or logically) valid*, written $\vDash S$, if it is true in **all** interpretations.

> Whereas in PL we studied tautologies, in FOL we study logically valid formulae.

We write $S_1 \equiv S_2$ to mean that the sentences S_1 and S_2 are true in exactly the same interpretations. It is not hard to prove that this is equivalent to showing that $\vDash S_1 \leftrightarrow S_2$.

The following example should help to clarify these definitions. Specifically, why it is sentences that we are interested in.

Example 3.1.8. Interpret the 2-place predicate symbol $P(x, y)$ as the binary relation 'x is the father of y' where the domain is the set of people. Observe that the phrase 'x is the father of y' is neither true nor false since we know nothing about x and y. Consider now the wff $(\exists y)P(x, y)$. This says 'x is a father'. This is still neither true nor false since x is not specified. Finally, consider the wff $S_1 = (\forall x)(\exists y)P(x, y)$. This says 'everyone is a father'. Observe that it is a sentence and that it is also a statement. In this case, it is false. I should add that in reading a sequence of quantifiers work your way in from left to right. The new sentence $S_2 = (\exists x)(\forall y)P(x, y)$ is a different statement. It says that there is someone who is the father of everyone. This is also false. The new sentence $S_3 = (\forall y)(\exists x)P(x, y)$ says that 'everyone has a father' which is true.

We now choose a new interpretation. This time we interpret $P(x, y)$ as '$x \leq y$' and the domain as the set $\mathbb{N} = \{0, 1, 2, \ldots\}$ of natural numbers. The sentence S_1 says that for each natural number there is a natural number at least as big. This is true. The sentence S_2 says that there exists a natural number that is less than or equal to any natural number. This is true because 0 has exactly that property. Finally, sentence S_3 says that for each natural number there is a natural number that is no bigger which is true.

As we see in the above examples, sentences say something, or rather can be interpreted as saying something. Thus a sentence interpreted in some structure is a statement about that structure and is therefore true or false in that interpretation.

I should add that sentences also arise (and this is a consequence of the definition but perhaps not obvious), when constants, that is names, are substituted for variables. Thus 'Darth Vader is the father of Winnie-the-Pooh' is a sentence, that happens to be false (and I believe has never been uttered by anyone before).

3.1.6 De Morgan's laws for $\forall$ and $\exists$

The universal quantifier $\forall$ is a sort of infinite version of $\wedge$ and the existential quantifier is a sort of infinite version of $\vee$. The following result is therefore not surprising.

Theorem 3.1.9 (De Morgan for quantifiers)**.**

1. $\neg(\forall x)A \equiv (\exists x)\neg A$.

2. $\neg(\exists x)A \equiv (\forall x)\neg A$.

Exercises 3.1

The following were adapted from Chapter 1 of Volume II of [45]. There are some liberties taken in the solutions. Read [45] for more background.

In the following exercises, use this transcription guide:

a: Athelstan (name)

e: Ethelgiva (name)

c: Cenric (name)

$M(x)$: x is marmalade — the colour, not what you put on your toast (1-place predicate)

$C(x)$: x is a cat (1-place predicate)

$L(x, y)$: x likes y (2-place predicate)

$T(x, y)$: x is taller than y (2-place predicate)

1. Transcribe the following formulae into English.
   (a) $\neg L(a, a)$.
   (b) $L(a, a) \to \neg T(a, a)$.
   (c) $\neg(M(c) \vee L(c, e))$.
   (d) $C(a) \leftrightarrow (M(a) \vee L(a, e))$.
   (e) $(\exists x)T(x, c)$.
   (f) $(\forall x)L(a, x) \wedge (\forall x)L(c, x)$.
   (g) $(\forall x)(L(a, x) \wedge L(c, x))$.
   (h) $(\exists x)T(x, a) \vee (\exists x)T(x, c)$.
   (i) $(\exists x)(T(x, a) \vee T(x, c))$.

(j) $(\forall x)(C(x) \rightarrow L(x, e))$.

(k) $(\exists x)(C(x) \wedge \neg L(e, x))$.

(l) $\neg(\forall x)(C(x) \rightarrow L(e, x))$.

(m) $(\forall x)[C(x) \rightarrow (L(c, x) \vee L(e, x))]$.

(n) $(\exists x)[C(x) \wedge (M(x) \wedge T(x, c))]$.

2. For each formula in Question 1, draw its parse tree.

3. Transcribe the following English sentences into formulae.

   (a) Everyone likes Ethelgiva.

   (b) Everyone is liked by either Cenric or Athelstan.

   (c) Either everyone is liked by Athelstan or everyone is liked by Cenric.

   (d) Someone is taller than both Athelstan and Cenric.

   (e) Someone is taller than Athelstan and someone is taller than Cenric.

   (f) Ethelgiva likes all cats.

   (g) All cats like Ethelgiva.

   (h) Ethelgiva likes some cats.

   (i) Ethelgiva likes no cats.

   (j) Anyone who likes Ethelgiva is not a cat.

   (k) No one who likes Ethelgiva is a cat.

   (l) Somebody who likes Athelstan likes Cenric.

   (m) No one likes both Athelstan and Cenric.

4. This question is about interpreting sequences of quantifiers. Remember that $\forall x$ can be read both as *for all* x as well as *for each* x. Consider the following 8 sentences involving the 2-place predicate P.

   (a) $(\forall x)(\forall y)P(x, y)$.

   (b) $(\forall x)(\exists y)P(x, y)$.

   (c) $(\exists x)(\forall y)P(x, y)$.

(d) $(\exists x)(\exists y)P(x, y)$.

(e) $(\forall y)(\forall x)P(x, y)$.

(f) $(\exists y)(\forall x)P(x, y)$.

(g) $(\forall y)(\exists x)P(x, y)$.

(h) $(\exists y)(\exists x)P(x, y)$.

Consider two interpretations. The first has domain $\mathbb{N}$, the set of natural numbers, and P is interpreted as $\leq$. The second has domain 'all people', and P is interpreted by the binary relation 'is the father of'. Write down what each of the sentences means in each interpretation and state whether the interpretation is a model of the sentence or not. You may use the fact that (a)≡(e) and (d)≡(h).

3.2 Truth trees for FOL

We begin with a motivating example.

Example 3.2.1. Consider the following famous argument.

1. All man are mortal.
2. Socrates is a man.
3. Therefore socrates is mortal.

Here (1) and (2) are the assumptions and (3) is the conclusion. If you agree to the truth of (1) and (2) then you are obliged to accept the truth of (3). This cannot be verified using PL but we can use what we have introduced so far to analyse this argument and prove that it is valid. We introduce some predicate symbols. We interpret $M(x)$ to be 'x is mortal', and $H(x)$ to be 'x is a man'. Our argument above has the following form.

1. $(\forall x)(H(x) \to M(x))$.
2. H(Socrates).
3. Therefore M(Socrates).

We prove that this argument is valid. If (1) is true, then it is true for every named individual a and so $H(a) \rightarrow M(a)$. Thus for the particular individual Socrates we have that $H(\text{Socrates}) \rightarrow M(\text{Socrates})$. But we are told in (2) that $H(\text{Socrates})$ is true. We are now in the world of PL and we have that

$$H(\text{Socrates}) \rightarrow M(\text{Socrates}), H(\text{Socrates}) \vDash M(\text{Socrates}).$$

Thus $M(\text{Socrates})$ is true.

Truth tree rules for FOL

Given a sentence $(\forall x)A(x)$, where $A(x)$ here means some wff containing x, then if we replace all free occurrences of x in $A(x)$ by a constant a, we have *instantiated the universal quantifier at* a. There is a similar procedure for existential quantification.

The leading idea in what follows is this: *convert FOL sentences into PL wff by means of instantiation.*

- All PL truth tree rules are carried forward.
- De Morgan's rules for quantifiers

$$\begin{array}{ll} \neg(\forall x)A & \neg(\exists x)A \\ \quad | & \quad | \\ (\exists x)\neg A & (\forall x)\neg A \end{array}$$

- New name rule.

$$\begin{array}{c} (1)\ (\exists x)A(x)\checkmark \\ \quad | \\ A(a) \end{array}$$

  where we add $A(a)$ at the bottom of all branches containing (1) and where a is a constant that does not already appear in the branch containing (1).

- Never ending rule.

$$\begin{array}{c} (2)\ (\forall x)A(x)\ * \\ \quad | \\ A(a) \end{array}$$

  where we add $A(a)$ at the bottom of a branch containing (2) and a is any constant appearing in the branch containing (2) or a is a new constant if no constants have yet been introduced. [The rationale for the latter is that all domains are non-empty]. We have used the $*$ to mean that the wff is never used up.

We use truth trees for FOL in the same way as in PL.

- To prove that $\vDash X$ we show that some truth tree with root $\neg X$ closes.
- To prove that $X_1, \ldots, X_n \vDash X$ we show that some truth tree with root $X_1, \ldots, X_n, \neg X$ closes.
- To prove that $X \equiv Y$ prove that $\vDash X \leftrightarrow Y$.

We shall describe how these rules are applied by means of examples.

Examples 3.2.2.

1. Show that the following is a valid argument

$$(\forall x)(H(x) \to M(x)), H(a) \vDash M(a).$$

Here is the truth tree.

$$\begin{array}{c} (\forall x)(H(x) \to M(x)) * \\ H(a) \\ \neg M(a) \\ | \\ H(a) \to M(a) \checkmark \\ \swarrow \quad \searrow \\ \times \neg H(a) \qquad M(a) \times \end{array}$$

2. Show that the following argument is valid

$$(\exists x)(\forall y)F(x,y) \vDash (\forall y)(\exists x)F(x,y).$$

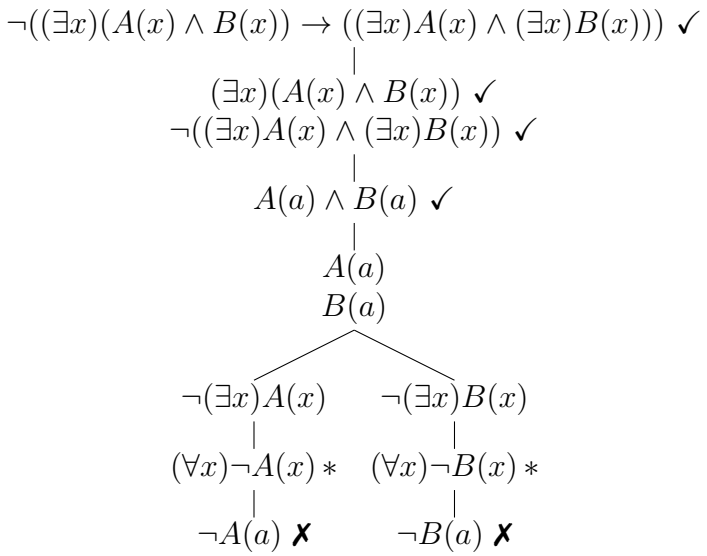
Here is the truth tree.

$$\begin{array}{c} (\exists x)(\forall y)F(x,y) \checkmark \\ \neg(\forall y)(\exists x)F(x,y) \checkmark \\ | \\ (\exists y)\neg(\exists x)F(x,y) \checkmark \\ | \\ (\exists y)(\forall x)\neg F(x,y) \checkmark \\ | \\ (\forall y)F(a,y) * \\ | \\ (\forall x)\neg F(x,b) * \\ | \\ F(a,b) \\ | \\ \neg F(a,b) \times \end{array}$$

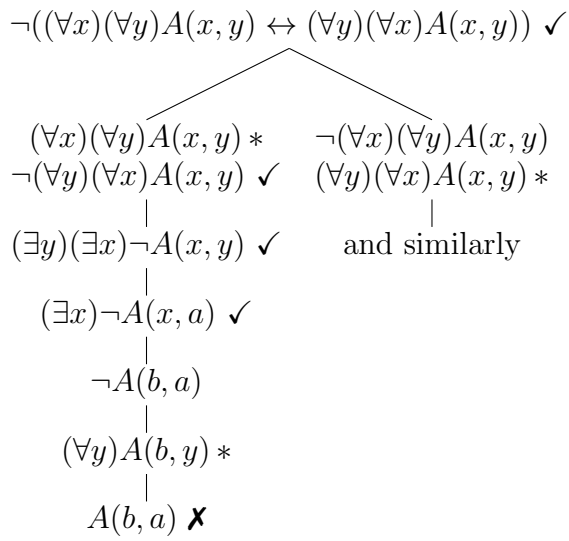
3. Prove that

$$\vDash (\exists x)(A(x) \wedge B(x)) \rightarrow ((\exists x)A(x) \wedge (\exists x)B(x)).$$

Here is the truth tree.



4. Prove that $(\forall x)(\forall y)A(x,y) \equiv (\forall y)(\forall x)A(x,y)$. Here is the truth tree.



There are, however, some major differences between PL and FOL when it comes to the behaviour of truth trees.

Examples 3.2.3.

1. Truth trees can have infinite branches. Here is an example.

$$\begin{array}{c} (\forall x)(\exists y)R(x,y)\ * \\ | \\ (\exists y)R(a_1,y)\ \checkmark \\ | \\ R(a_1,a_2) \\ | \\ (\exists y)R(a_2,y)\ \checkmark \\ | \\ R(a_2,a_3) \\ | \\ \text{and so on} \end{array}$$

2. There are sentences that have no finite models. See Question 10 of Exercises 3.2.

3. The order in which the rules for truth trees are applied in FOL does matter. If we place at the root the wff $(\forall x)(\exists y)P(x,y)$ and $P(a)\wedge\neg P(a)$ then we can get an infinite tree if we repeatedly apply the tree rules to the first wff but an immediate contradiction, and so a closed finite truth tree, if we start with the second wff instead.

Exercises 3.2

---

1. Prove that
$$(\forall x)R(x) \models (\exists x)R(x).$$
Explain informally why it is a valid argument.

2. Prove that
$$(\forall x)(\forall y)R(x,y) \models (\forall x)R(x,x).$$

3. Prove that
$$(\exists x)(\exists y)F(x,y) \equiv (\exists y)(\exists x)F(x,y).$$

4. Prove that
$$(\forall x)(P(x)\wedge Q(x)) \equiv ((\forall x)P(x)\wedge(\forall x)Q(x)).$$

5. Prove that

$$(\exists x)(P(x) \vee Q(x)) \equiv ((\exists x)P(x) \vee (\exists x)Q(x)).$$

6. Prove that

$$((\forall x)P(x) \vee (\forall x)Q(x)) \rightarrow (\forall x)(P(x) \vee Q(x))$$

is logically valid. On the other hand, show that the following is not logically valid by constructing a counterexample

$$(\forall x)(P(x) \vee Q(x)) \rightarrow ((\forall x)P(x) \vee (\forall x)Q(x)).$$

7. Prove that

$$(\exists x)[D(x) \rightarrow (\forall y)D(y)]$$

is universally valid. Interpret $D(x)$ as 'x drinks' with domain people. What does the above wff say in this interpretation? Does this seem plausible? Resolve the issue. (This is a famous example of Raymond Smullyan).

8. For each of the following formulae draw a parse tree and demonstrate that the formula is closed and therefore a sentence. Then show that each sentence is logically valid.

   (a) $(\forall x)P(x) \rightarrow (\exists x)P(x)$.

   (b) $(\exists x)P(x) \rightarrow (\exists y)P(y)$.

   (c) $(\forall y)((\forall x)P(x) \rightarrow P(y))$.

   (d) $(\exists y)(P(y) \rightarrow (\forall x)P(x))$.

   (e) $\neg(\exists y)P(y) \rightarrow [(\forall y)((\exists x)P(x) \rightarrow P(y))]$.

9. This question marks the beginning of using our logic in mathematical proof. Let R be a 2-place predicate symbol. We say that an interpretation of R has the respective property (in italics) if it is a model of the corresponding sentence.

   - *Reflexive:* $(\forall x)R(x, x)$.
   - *Irreflexive:* $(\forall x)\neg R(x, x)$.

- *Symmetric:* $(\forall x)(\forall y)(R(x,y) \rightarrow R(y,x))$.
- *Asymmetric:* $(\forall x)(\forall y)(R(x,y) \rightarrow \neg R(y,x))$.
- *Transitive:* $(\forall x)(\forall y)(\forall z)(R(x,y) \wedge R(y,z) \rightarrow R(x,z))$.

Illustrate these definitions by using directed graphs.

Prove the following using truth trees.

(a) If R is asymmetric then it is irreflexive.

(b) If R is transitive and irreflexive then it is asymmetric.

10. Put $S = F_1 \wedge F_2 \wedge F_3$ where

- $F_1 = (\forall x)(\exists y)R(x,y)$.
- $F_2 = (\forall x)\neg R(x,x)$.
- $F_3 = (\forall x)(\forall y)(\forall z)[R(x,y) \wedge R(y,z) \rightarrow R(x,z)]$.

Prove first that the following is a model of S: the domain is $\mathbb{N}$ and $R(x,y)$ is interpreted as $x < y$. Next prove that S has *no finite models*. That is, no models in which the domain is finite.

3.3 The *Entscheidungsproblem*

Let's start with PL. If I give you a wff, you can decide, using a truth table for example, in a finite amount of time whether that wff is a tautology or not. The decision you make does not require any intelligence; it uses an algorithm. We say that the problem of deciding whether a wff in PL is a tautology is decidable.

Let's turn now to FOL. The analogous question of whether a formula in FOL is universally valid or not is usually known by its German form, the *Entscheidungsproblem*, because it was formulated by the great German mathematician David Hilbert in 1928. Unlike the case of PL, there is no algorithm for deciding this question. This was proved independently in 1936 by Alonzo Church in the US and by Alan Turing in the UK. Turing's resolution to this question was far-reaching since it involved him in formulating, mathematically, what we would now call a computer.

Bibliography

[1] S. Aaronson, *Quantum computing since Democritus*, CUP, 2013.

[2] Marcus Aurelius, *The meditations*, translated by Robin Hard, OUP, 2011.

[3] M. Ben-Ari, *Mathematical logic for computer science*, Third Edition, Springer, 2012.

[4] A. Bierce, *The enlarged Devil's dictionary*, Penguin Books, 1989.

[5] G. Boole, *The laws of thought*, 1854 (modern editions available).

[6] G. S. Boolos, J. P. Burgess, R. C. Jeffrey, *Computability and logic*, Fifth Edition, CUP, 2010.

[7] R. Cori, D. Lascar, *Mathematical logic*, OUP, 2000.

[8] W. J. Dally, R. C. Harting, T. M. Aamodt, *Digital design using VHDL*, CUP, 2016.

[9] H. Delong, *A profile of mathematical logic*, Dover, 2004.

[10] A. Doxiadis, C. H. Papadimitriou, *Logicomix*, Bloomsbury, 2009.

[11] R. M. Exner, M. F. Rosskopf, *Logic in elementary mathematics*, Dover, 2011.

[12] T. L. Floyd, *Digital fundamentals*, Ninth Edition, Pearson Education International, 2006.

[13] R. W. Floyd, R. Beigel, *The language of machines*, Computer Science Press, 1994.

[14] J. H. Gallier, *Logic for computer science. Foundations of automatic theorem proving*, John Wiley & Sons, 1987.

[15] M. R. Garey, D. S. Johnson, *Computers and intractability. A guide to the theory of NP-completeness*, W. H. Freeman and Company, 1997.

[16] G. Gentzen, Untersuchungen über das logische Schließen. I. *Mathematische Zeitschrift* **39** (1933), 176–210.

[17] S. Givant, P. Halmos, *Introduction to Boolean algebras*, Springer, 2009.

[18] T. Hailperin, Boole's algebra isn't Boolean algebra, *Mathematics Magazine* **54** (1981), 172–184.

[19] R. Hammack, *Book of proof*, VCU Mathematics Textbook Series, 2009. This book can be downloaded for free from `http://www.people.vcu.edu/~rhammack/BookOfProof/index.html`.

[20] D. Harel, *Computers Ltd. What they really can't do*, OUP, 2012.

[21] D. Hilbert, W. Ackermann, *Principles of mathematical logic*, Chelsea Publishing Company, New York, 1950.

[22] D. R. Hofstadter, *Gödel, Escher, Bach: an eternal golden braid*, Basic Books, 1999.

[23] M. R. A. Huth, M. D. Ryan, *Logic in computer science: modelling and reasoning about systems*, CUP, 2000.

[24] W. Kneale, M. Neale, *The development of logic*, Clarendon Press, Oxford, 1986.

[25] D. C. Kozen, *Automata and computability*, Springer, 1997.

[26] G. Labrecque, Truth tree generator, `http://gablem.com/logic`.

[27] M. V. Lawson, *Finite automata*, Chapman & Hall/CRC, 2004.

[28] M. V. Lawson, *Algebra & geometry: an introduction to university mathematics*, CRC Press, 2016.

[29] S. Lipschutz, M. Lipson, *Discrete mathematics*, Second Edition, McGraw-Hill, 1997.

[30] M. M. Mano, *Digital design*, Prentcie-Hall International, 1984.

[31] G. H. Mealy, A method for synthesizing sequential circuits, *Bell System Technical Journal* **34** (1955), 1045-1079.

[32] A. A. Milne, *Winnie-the-Pooh*, Methuen, 1995.

[33] A. A. Milne, *The house at Pooh corner*, Methuen, 1995.

[34] M. A. Nielsen, I. L. Chuang, *Quantum computation and quantum information*, CUP, 2002.

[35] C. H. Papadimitriou, *Computational complexity*, Addison Wesley Longman, 1994.

[36] C. Petzold, *Code: the hidden language of computer hardware and software*, Microsoft Press, 2000.

[37] C. Petzold, *The annotated Turing*, John Wiley & Sons, 2008.

[38] U. Schöning, *Logic for computer scientists*, Birkhäuser, 2008.

[39] D. Scott et al, *Notes on the formalization of logic: Parts I and II*, Oxford, July, 1981.

[40] R. M. Smullyan, *First-order logic*, Dover, 1995.

[41] R. M. Smullyan, *Logical labyrinths*, A. K. Peters, Ltd. 2009.

[42] R. M. Smullyan, *A beginners guide to mathematical logic*, Dover, 2014.

[43] P. Smith, *An introduction to formal logic*, CUP, 2011.

[44] R. R. Stoll, *Set theory and logic*, W. H. Freeman and Company, 1961.

[45] P. Teller, *A modern formal logic primer*, Prentice Hall, 1989. This book can be downloaded for free from `http://www.ma.hw.ac.uk/~markl/teaching/LOGIC/Teller.html`.

[46] N. Tennant, *Introductory philosophy: god, mind, world and logic*, Routledge, 2015.

[47] A. M. Turing, On computable numbers, with an application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society*, **42** (1937), 230–265.

[48] L. E. Turner, Truth table generator, `http://turner.faculty.swau.edu/mathematics/materialslibrary/truth/`.

[49] D. van Dalen, *Logic and structure*, Third Edition, Springer, 1997.

[50] L. Wittgenstein, *Tractatus Logico-Philosophicus*, Routledge, 2009.

[51] M. Zegarelli, *Logic for dummies*, Wiley Publishing, 2007.