Mathematical Structures

Second Edition

Version 2.0

Elizabeth Scott

José Fiadeiro

Department of Computer Science
Royal Holloway University of London

September 2017

Preface

Quoting Keith Devlin, a famous British mathematician and popular science writer,

> *Once you realise that computing is all about constructing, manipulating, and reasoning about abstractions, it becomes clear that an important prerequisite for writing (good) computer programs is the ability to handle abstractions in a precise manner.*
>
> *As it happens, that is something we humans have been doing successfully for more than three thousand years. We call it mathematics.*[1]

These notes cover some of the elements of mathematics that contribute to one of the essential skills that employers look for in computer scientists: *problem solving*. This means the ability to, given a problem, use mathematical structures to model the domain and use that model to develop a software system to solve the problem.

Changes in this second edition

This is a revised and extended edition of the notes written by Elizabeth Scott in 2009 to accompany the first-year module CS1860.

The major modifications that have been brought to the original version reflect the change to the syllabus operated in 2017/18: vector spaces have moved to a new second-year module dedicated to multi-dimensional data, and have been replaced by a more extended coverage of orderings (which now have a dedicated chapter).

[1] "Why universities require computer science students to take maths", Communications of the ACM 46(9), 2003

Contents

Chapter 1

Introduction

1.1 Why do computer science students need to learn maths?

What sorts of problems can we expect a computer to be able to solve? How do we represent and reason about programs in a way that allows us to be confident that they are correct?

Some students may ask why a computer science degree should include a mathematics component. For some students, mathematics is interesting and/or enjoyable and this is reason enough. But why is mathematics compulsory for all computer science students at Royal Holloway?

Mathematics is intimately involved with computer science in two basic ways:

- Ultimately, computers perform mathematical calculations — this is historically why computing was developed (by mathematicians such as Alan Turing). In order to understand and be able to ensure that a program does what it is meant to do, computer scientists need to be 'mathematically literate'.

- Computer programs (software) are essentially developed in response to a client's need to solve a problem. To develop such a program, it helps that the developer conceptualises the problem through a mathematical model and uses that model to find a solution that can be programmed.

This is similar to what engineers or physicists do. In order to predict the weather, one needs a mathematical model of climate; that model keeps being updated by confronting the predictions with what actually happens. In order to build a bridge, engineers need a mathematical model of the physical location of where the bridge is needed, of the bed of the river where the pillars will go, of the winds that the bridge will be subject to, and so on; many disasters are due to the poor quality of the mathematical model that was used.

In summary, computer scientists need to be able to abstract models from real-world situations that can be processed by computers to answer relevant questions.

1.1.1 Mathematical literacy

In order to be a competent historian it is necessary to read, write and comprehend English (or the native language of the country whose history is being studied). It is necessary to be able to write critical essays, and to use the power of the language to convey the points being made.

Mathematics plays a similar role in computer science to that played by English in History. Just as historians don't need to be able to write novels or poetry, computer scientists don't need to be able to prove new mathematical theorems. However, one of the best ways to acquire skill in expression is to study novels and poetry, and one of the best ways to acquire mathematical literacy is to study mathematical results and applications. A historian does not need to be a poet but does need to be literate; a computer scientist does not need to be a mathematician, but does need to be mathematically literate.

It is possible to view any task we wish to perform in terms of tools, techniques, and methodologies. The tools are the equipment available, the techniques are the (standard) ways in which the tools are operated, and the methodologies are the ways in which the techniques are combined to successfully perform the task.

Tools	**Techniques**	**Methodologies**
tennis shoes tennis racket tennis balls tennis court	forehand backhand serve volley lob	singles strategy doubles strategy specific opponents
terminal keyboard computer compiler	typing skills using the program constructs using the compiler using the operating system	produce an algorithm
pen+paper calculator symbols	use of notation proof by contradiction generalisation proof by induction ...	mathematical modelling axiomatic deduction ...

In general, the tools are provided (created) by someone else. The techniques for using the tools are taught by an instructor. The skill of finding a methodology for performing a given task can only be developed with practice and experience.

The process of deciding how to (correctly) perform a task is referred to as *problem solving*. The task may be anything from beating a particular person at tennis to writing a program to calculate a specific result.

1.1.2 Mathematical modelling

Many methodologies for problem solving are types of *mathematical modelling*. Mathematical modelling consists of taking a 'real' situation and reformulating it in mathematical terms, so that we can use mathematical techniques to find a solution. The reformulation involves selecting some properties, ignoring others, and using more concise notation. This is called *abstraction*, and models which involve generalisation are called *abstract models* because we have abstracted away some unnecessary details to make the model simpler.

For example, suppose we have to solve the following problem:

> *Mary has 35 pencils, which she bought for 12p each. How many pencils must she sell at 14p before she makes a profit?*

We break the problem down into small steps. The expenses are the amount Mary had to spend, the income is what she receives from her sales, and her profit is the income minus the expenses:

$$\text{profit} = \text{income} - \text{expenses}$$

We can calculate the values of income and expenses:

$$\begin{aligned}\text{expenses} &= (\text{cost})\times(\text{number bought}) = 12\times 35 = 420\\ \text{income} &= (\text{price})\times(\text{number sold}) = 14\times(\text{number sold})\end{aligned}$$

Mary makes a profit when (income − expenses) > 0, i.e. when

$$14\times(\text{number sold}) > 420$$

Using simple algebra we have that

$$(\text{number sold}) > 30$$

So Mary must sell more than 30 pencils before she makes a profit.

We have done more than solve the pencils problem. We have produced an algorithm that can be used to write a general purpose computer program. Suppose that we are given the following task:

> *Write a computer program which accepts the cost, the selling price, and the number of items bought, and outputs the number that must be sold before a profit is made.*

All we have to do is substitute variable names for the prices and quantities. Then the mathematical model above will provide us with an algorithm for solving the problem.

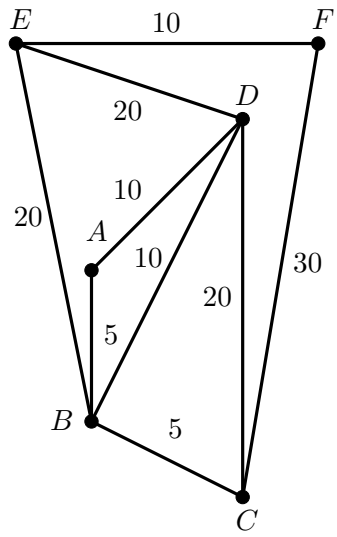
```
write('enter number of items bought')
read(number_bought)
write('enter cost per item')
read(cost)
write('enter selling price')
read(price)
    need_to_sell = (cost * number_bought) / price
write('for a profit sell more than ', need_to_sell)
```

This can be translated into the chosen programming language.

Next we look at another example of mathematical modelling. Suppose that there are six villages scattered over a region of Exmoor and that, to bring super-fast broadband to the villages, a company decides to layout fibre optics along the roads that connect them. In order to minimise the overall cost, the company wants to find the shortest amount of cable needed to connect all the villages. Thus we have the following task.

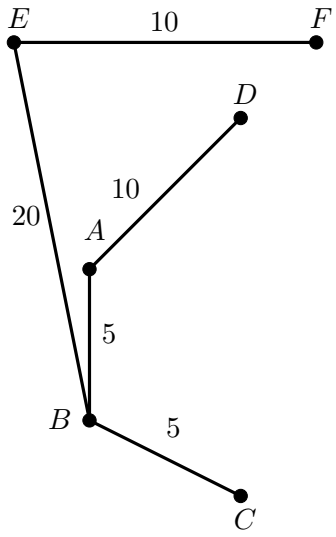
> *Find the smallest length of (existing) road needed to ensure that all the villages are connected.*

We begin by formulating the problem in mathematical terms, using weighted graphs. A *weighted graph* consists of a set of vertices, a set of edges joining selected vertices and, for each edge a real number (called weight). We can use a weighted graph to represent the Exmoor village road system: the villages are vertices, the roads between them are edges and the weights are the length of the roads.



This representation is the result of a process of abstraction: in order to solve the problem, we don't need to know many aspects of the villages (for example, what they are called) or of the roads (for example, how wide they are), or indeed how they are geographically located (the graph does not need to be a faithful map). All the information that is needed to solve the problem is represented by the graph and none of it is redundant.

Why is this useful? Having modelled the problem domain as a graph, a computer scientist can then use properties of graphs to develop algorithms that, once implemented in a programming language, can be used to solve the problem. In fact, as explained in Chapter 7, a computer scientist who has taken this module will know that such an algorithm already exists and can be programmed. A possible solution returned by that algorithm is:



You can check that 50 miles is indeed the minimum length of fibre optics that is required to connect all villages.

In summary, one of the main skills that a computer scientist needs is to be able to determine how to solve a problem in a way that allows a program to be written that does it. Ultimately, a computer has no ability to 'think', it only 'knows' what it has been told and it can only execute actions that it has been given.

1.2 Aims

The aim of this module is to provide insights and skills in rigour and formal reasoning in a way that allows reasoning about behaviour, correctness and performance in a programming environment. This module contributes to one of the essential skills that employers look for in computer scientists: *problem solving*. This means the ability to use mathematical

structures to model the problem domain and use that model to develop a software system to solve the problem.

By the end of this module, students should be able to:

- reason about sets, relations, functions and cardinality
- reason about recursive definitions and prove results by induction
- represent and reason about problems using graphs
- have an understanding of basic probability and statistics suitable for use in studying artificial intelligence and information security

1.3 Course outline

Sets: defining sets, logic notation, proofs by construction, counterexample and contradiction

Relations: relations, orderings, functions

Recursion: recursive definitions and induction, cardinality of infinite sets

Graph theory: graphs, trees and spanning trees, directed graphs

Probability: elementary and conditional probability, Bayes theorem, random variables, statistical distributions (uniform, binomial and geometric)

1.4 Recommended reading

Most of the topics in this course are covered in the book by Rosen, which is also one of the recommended books for CS1870. All references in these notes to Rosen are for the 7th Edition. Earlier editions are also acceptable but the numbering of sections may be different.

1. Kenneth Rosen, *Discrete Mathematics And Its Applications*, 7th Edition, McGraw-Hill, 2012. ISBN 978-0-07-338309-5. (Library code 512.23ROS.)

Another recommended book is:

2. Kenneth A. Ross and Charles R.B. Wright, *Discrete Mathematics*, Prentice Hall, 2003. ISBN 0-13-065247-4. (Library code 510ROS.)

Other reading material will be suggested throughout the notes for particular topics, and kept up to date on the Moodle page of the module.

In this module, familiarity with prime numbers and division properties of integers is assumed. If you have not had much practice with these topics you should read Chapter 4 of Rosen, or the beginning of any elementary book on number theory. A summary of some of the basic properties is given at the end of these notes.

Chapter 2

Set theory

Set theory forms a basis of all mathematics. It provides us with a powerful notation for expressing properties. In this chapter we will review the basic aspects of set theory.

(*Most of the material in this chapter can be found in Sections 1.7, 1.8, 2.1 and 2.2 of Rosen.*)

2.1 Defining and comparing sets

2.1.1 Sets and elements

A *set* can be informally defined as a collection of objects. The objects are called *elements* of the set. For example, all the people in this class form a set, and each person here is an element of that set.

Some sets are used so often that they have special symbols to denote them.

- $\emptyset$ denotes the *empty set*, the set that has no elements.
- $\mathbb{N}$ (or $\mathbb{N}_0$) denotes the set of all 'natural' numbers, i.e. $0, 1, 2, 3, 4$, etc
- $\mathbb{P}$ (or $\mathbb{N}_1$) denotes the set of all positive natural numbers, i.e. $1, 2, 3, 4$, etc.
- $\mathbb{Z}$ denotes the set of all integers: zero, positive and negative natural numbers
- $\mathbb{Q}$ denotes the set of all rational numbers: numbers that can be expressed as a fraction n/m where n is an integer and m is a positive integer.
- $\mathbb{S}_n$ denotes the set of the first n positive integers, i.e. $1, 2, \ldots, n$.

Sets can be queried in relation to their elements through the expression $a \in S$, which is true if S is a set and a is one of its elements. Its negation is $a \notin S$, which is true if either S is not a set or it is a set but a is not one of its elements. So $a \notin \emptyset$ is true no matter what a is, for example $\emptyset \notin \emptyset$ – if you think of sets as boxes that store objects, $\emptyset$ is an empty box so it cannot have anything inside, not even an empty box.

Sets can be defined in several ways.

Descriptively, as in 'the set of all black swans'.

This is perhaps the simplest method, but it is also the least precise: for many years, that set was believed to be empty! In addition, one cannot tell precisely how many elements it contains.

So we use other methods.

Enumeration, by listing all the members of a set between braces. For example:

- $\{1, 2, 4, 6, 8\}$
- {red, blue, the tower of London}
- {} (an alternative notation for $\emptyset$)

In order to query a set A defined in this way through $a \in A$, we just check whether a is listed as a member of A.

The order in which the members are listed is unimportant. For example,

$$\{1, 2, 3\},\ \{3, 1, 2\},\ \{3, 2, 1\}$$

all represent the same set. Repeated elements do not count either. For example,

$$\{1, 2\},\ \{1, 1, 2\},\ \{1, 2, 1\}$$

all represent the same set.

This method is fine, but there is a problem if there are lots of elements. If the elements follow a (very) simple pattern, we can use 'dots' to indicate missing elements, thus making the representation shorter, as in

$$\{1, 2, ..., 201\}$$

However, this is not precise enough, especially for a computer (at least, not at the time of writing these notes): for a computer to answer a query of the form $a \in A$, it would need the algorithm behind what we mean by the dots.

Predicates: We often need to consider sets of objects all of which share a common property. Our third way of defining sets essentially says 'the set of all objects x with property $P(x)$'. For example,

- the set of all odd integers — $P(x)$ is 'x is an odd integer'

- the set of all roots of the function f — $P(x)$ is 'x is a root of the function f'
- the set of all Kings and Queens of England since 1066 — $P(x)$ is 'x is a King or Queen of England since 1066'

In order to query a set A defined in this way through $a \in A$, we just check if a satisfies $P(x)$, i.e. if $P(a)$ is true.

We write the set of all objects for which $P(x)$ is true as $\{x \mid P(x)\}$. For example, the set of odd integers is written

$$\{x \mid x = 2n + 1, \text{ for some } n \in \mathbb{Z}\}.$$

Notice how this definition effectively provides an algorithm for answering a query: to check whether x belongs to the set, you check if the equation $x = 2n + 1$ (note that x is given) has a solution n in $\mathbb{Z}$. For example, $5 = 2n + 1$ has a solution ($n = 2$), but $6 = 2n + 1$ does not, nor does $\pi = 2n + 1$.

Other examples are:

- $\mathbb{S}_n = \{x \mid x \in \mathbb{N} \text{ and } 1 \leq x \leq n\}$
- $\mathbb{P} = \{x \mid x \in \mathbb{N}, x \neq 0\}$
- $\mathbb{Z} = \{x \mid x \in \mathbb{N} \text{ or } -x \in \mathbb{N}\}$
- $\mathbb{Q} = \{p/q \mid p \in \mathbb{Z},\ q \in \mathbb{P}\}$

Programs operate on data that are grouped together in *types*. These types are sets. For example, the Java type *int* is the set of integers that can be used in a program. A C++ enumerated type is the set of objects whose elements are listed explicitly. For example,

```
enum days_of_week {Mon, Tue, Wed, Thur, Fri, Sat, Sun}
```

declares the set {Mon, Tue, Wed, Thur, Fri, Sat, Sun}.

2.1.2 Equality of sets

Throughout this module we shall have to consider the question of what we mean by saying two things are equal. In programming we are concerned about whether two objects have the same value, and in this case whether changing the value of one also changes the value of the other (aliasing). More generally, in computer science we are interested in knowing whether two objects that are different are actually equivalent, and thus can be used interchangeably. In Section 3.4, we will return to this question.

Definition 2.1.1 *Two sets are* equal *if they each contain exactly the same elements. We write $A = B$ if the sets A and B are equal.*

For example:

- $\{2, 4, \{\}, \theta\} = \{\{\}, 2, \theta, 4\}$
- $\{x \mid x = 2n + 1 \text{ for some } n \in \mathbb{N} \text{ and } x \leq 11\} = \{1, 3, 5, 7, 9, 11\}$

The most general way of proving that two sets A, B are equal is by showing that all the elements of A are also elements of B and the other way round, i.e. each set contains the other.

Definition 2.1.2 *A set A is a* subset *of a set B if $a \in A$ implies $a \in B$, in which case we write $A \subseteq B$ (or, equivalently, $B \supseteq A$).*

For example,

- $\{1, 2, 4, 77\} \subseteq \mathbb{N}$
- $\mathbb{N} \subseteq \mathbb{Z}$
- $\emptyset \subseteq A$, for any set A

Set inclusion satisfies three important properties that we will further explore in Section 4:

Proposition 2.1.3 *Given any sets A, B and C*

- *$A \subseteq A$, i.e $\subseteq$ is reflexive*
- *If $A \subseteq B$ and $B \subseteq A$ then $A = B$, i.e. $\subseteq$ is antisymmetric*
- *If $A \subseteq B$ and $B \subseteq C$ then $A \subseteq C$, i.e. $\subseteq$ is transitive.*

How can we check if one set is included in another? For small sets we can write out the elements and check by examining each element. However, we can't do this when reasoning about large sets.

If the sets A and B are defined by a predicate (property) then we can show that an element of A is also an element of B by showing that it has the property required for B.

Example 2.1.4 *Show that for the following two sets $A \subseteq B$:*

$A = \{5, 8, 23\}$
$B = \{x \mid x = 3n + 2 \text{ for some } n \in \mathbb{Z}\}$

Proof *We take the elements of A in turn and check if they belong to B:*

$5 = 3n + 2$ *implies* $3 = 3n$, *which implies* $n = 1$; *because* $1 \in \mathbb{Z}$, $5 \in B$.

$8 = 3n + 2$ *implies* $6 = 3n$, *which implies* $n = 2$; *because* $2 \in \mathbb{Z}$, $8 \in B$.

$23 = 3n + 2$ *implies* $21 = 3n$*, which implies* $n = 7$*; because* $7 \in \mathbb{Z}$*,* $23 \in B$*.*

Thus every element of A *belongs to* B *and so* $A \subseteq B$*.*

Example 2.1.5 *Show that the following two sets are equal.*

$A_1 = \{x \mid x = 2n + 1 \text{ for some } n \in \mathbb{Z}\}$
$A_2 = \{x \mid x = 2n - 1 \text{ for some } n \in \mathbb{Z}\}$

Proof *We show that* $A_1 \subseteq A_2$ *and* $A_2 \subseteq A_1$*:*

$\boxed{A_1 \subseteq A_2}$ *If* $x \in A_1$ *then, by definition of* A_1*, there must be an integer* $n \in \mathbb{Z}$ *such that* $x = 2n + 1$*. Since* n *is an integer then so is* $m = n + 1$*. We have* $m \in \mathbb{Z}$ *and* $x = 2m - 1$*, so* $x \in A_2$*. Thus we have* proved *that every element of* A_1 *also belongs to* A_2 *(even though there are infinitely many of these elements).*

$\boxed{A_2 \subseteq A_1}$ *If* $y \in A_2$ *then, by definition of* A_2*, then there must be an integer* $n \in \mathbb{Z}$ *such that* $y = 2n - 1$*. Since* n *is an integer then so is* $m = n - 1$*. We have* $m \in \mathbb{Z}$ *and* $y = 2m + 1$*, so* $y \in A_1$*.*

So $A_1 = A_2$ *as required.*

It is important to note that the use of the variable n is local to the definition of A_1 and to that of A_2. Therefore, in order to show that every element of A_1 is an element of A_2 one should NOT try to solve the equation $2n + 1 = 2n - 1$. This is just as in programming languages: n is like a local variable of both a `for` loop that generates the elements of A_1 and also of a separate `for` loop that generates the elements of A_2.

This method of proof can also allow us to prove two sets are equal even if we don't know what the sets are! More precisely, it can allow us to show that sets constructed from other sets in different ways are equal. We shall return to this later.

2.1.3 Cardinality

The number of elements in a set S is called its *cardinality*, and is written $|S|$. For example,

$$|\{1, 2\}| = 2 = |\{a, b\}| \qquad |\emptyset| = 0 \qquad |\mathbb{S}_n| = n$$

Some sets are infinite: for example, $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}$. This is often indicated by writing

$$|\mathbb{N}| = \infty \qquad |\mathbb{Z}| = \infty \qquad |\mathbb{R}| = \infty.$$

However, ∞ is not a number, so from the fact that two sets are infinite one cannot conclude that they have the same cardinality: for example, the statement $|\mathbb{N}| = |\mathbb{R}|$ is false. We will see in Section 5.4 how the cardinality of infinite sets can be compared, and show that

$$|\mathbb{N}| = |\mathbb{Z}| < |\mathbb{R}|$$

i.e. there are as many natural numbers as integers, but there are more real numbers than natural numbers.

2.1.4 Power sets

One way of building sets is through the powerset construction. This construction produces sets of sets.

Definition 2.1.6 *The* power set of A *is defined by*

$$\mathcal{P}(A) = \{B \mid B \subseteq A\}$$

That is, $\mathcal{P}(A)$ *the set of all subsets of* A*.*

For example,

- $\mathcal{P}(\{a, b\}) = \{\{a, b\},\ \{a\},\ \{b\},\ \emptyset\}$
- $\mathcal{P}(\{1, 2, 3\}) = \{\{1, 2, 3\},\ \{1, 2\},\ \{1, 3\},\ \{2, 3\},\ \{1\},\ \{2\},\ \{3\},\ \emptyset\}$
- $\mathcal{P}(\emptyset) = \{\emptyset\}$

It follows that $B \in \mathcal{P}(A)$ if and only if $B \subseteq A$. Therefore, the query $B \in \mathcal{P}(A)$ can be reduced to the query $B \subseteq A$, for which we can use the techniques covered in Section 2.1.2.

A useful result is that, for any finite set A, $|\mathcal{P}(A)| = 2^{|A|}$. That is, if $|A| = n \in \mathbb{N}$ then $|\mathcal{P}(A)| = 2^n$. This can be proved by induction, as we shall see later.

2.1.5 Product sets

Most programming languages have arrays as one of their basic data types. Arrays allow several pieces of information to be stored together.

For example, we may hold the roots of quadratic equations in arrays of size 2. So [1,-3] holds the roots of $x^2 + 2x - 3$.

In mathematics the equivalent of an array $[a, b]$ of size 2, is an *ordered pair*, or *vector*, (a, b). As you might expect, in an ordered pair the order of the elements is important. So $(a, b) = (c, d)$ if and only if $a = c$ and $b = d$. Therefore, $(1, 3) \neq (3, 1)$.

We can form ordered pairs (a, b) with elements from any two sets A and B. For example,

(1,a), (1,b), (1,c), ... , (1,z),
(2,a), (2,b), (2,c), ... , (2,z),
(3,a), (3,b), (3,c), ... , (3,z)

are ordered pairs formed by taking the first element from $\mathbb{S}_3$ and the second element from $\{a, b, c, ..., z\}$.

Definition 2.1.7 *Given any sets* A *and* B*, we define the* (Cartesian) product *of* A *and* B *through*

$$A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}$$

That is $A \times B$ *is the the set of ordered pairs whose first element comes from* A *and whose second element comes from* B*.*

For example,

- $\mathbb{S}_2 \times \mathbb{S}_4 = \{(1,1),(1,2),(1,3),(1,4),(2,1),(2,2),(2,3),(2,4)\}$
- $\mathbb{S}_4 \times \mathbb{S}_2 = \{(1,1),(1,2),(2,1),(2,2),(3,1),(3,2),(4,1),(4,2)\}$

This illustrates that $A \times B = B \times A$ does not necessarily hold.

As you might expect, we can form the Cartesian product of more than two sets: (a_1, a_2, a_3) is called an *ordered triple*, or a *3-dimensional vector*, and corresponds to an array $[a_1, a_2, a_3]$ of size 3.

More generally, we call $(a_1, a_2, \ldots, a_n)$ an *ordered n-tuple*, or an *n-dimensional vector*.

Definition 2.1.8 *Given $n > 0$ sets $A_1, A_2, \ldots, A_n$, we define their* (Cartesian) product *through*

$$A_1 \times A_2 \times \ldots \times A_n = \{(a_1, a_2, \ldots, a_n) \mid a_1 \in A_1, a_2 \in A_2, \ldots, a_n \in A_n\}.$$

We have that $(a_1, a_2, \ldots, a_n) = (b_1, b_2, \ldots, b_n)$ if and only if $a_1 = b_1, a_2 = b_2, \ldots, a_n = b_n$.

For example,

$$\mathbb{S}_2 \times \{a, b\} \times \mathbb{S}_1 = \{(1,a,1),(2,a,1),(1,b,1),(2,b,1)\}$$

It is easy to see that:

$$|\mathbb{S}_2 \times \mathbb{S}_4| = 8 = 2 \times 4$$
$$|\mathbb{S}_2 \times \{a, b\} \times \mathbb{S}_1| = 4 = 2 \times 2 \times 1$$

It is possible to prove that $|A \times B| = |A| \times |B|$, and generally that

$$|A_1 \times A_2 \times \ldots \times A_n| \quad = \quad |A_1| \times |A_2| \times \ldots \times |A_n|$$

2.1.6 Strings

When all the sets in a Cartesian product are the same, we write A^n rather than $A \times A \times \ldots \times A$ (A repeated $n > 0$ times) and we call their elements *strings* of length n (or n-strings). That is, $A^2 = A \times A$, $A^3 = A \times A \times A$, etc.

For example, if $B = \{0, 1\}$ (the elements of which we call 'bits') then

$$B^4 = \left\{ \begin{array}{llll} (0,0,0,0), & (1,0,0,0), & (0,1,0,0), & (0,0,1,0), \\ (0,0,0,1), & (1,1,0,0), & (1,0,1,0), & (1,0,0,1), \\ (0,1,1,0), & (0,1,0,1), & (0,0,1,1), & (1,1,1,0), \\ (1,1,0,1), & (1,0,1,1), & (0,1,1,1), & (1,1,1,1) \end{array} \right\}$$

Sometimes we just write sequences instead of tuples of elements, for example 0000 instead of $(0, 0, 0, 0)$.

We are often interested in the set of strings over A that are of arbitrary positive length, which we denote by A^+. For example, B^+ will contain all finite non-empty sequences of bits, which are called 'bit strings'; this includes 0, 00, 000, etc. You will know that we call a 'byte' a bit string of length 8.

A string of length 0 is an empty sequence, which we usually denote by ϵ; we also define $A^0 = \{\epsilon\}$. Given a set A, A^* is the set of all strings over A including the empty string ϵ.

2.2 Operations on sets

2.2.1 Definitions and examples

Typical operations on sets are *union*, *intersection*, *complement* or *difference*, and *symmetric difference*. The mathematical notation for these operations is $\cup$, $\cap$, $\setminus$ and $\oplus$.

These operations are defined as follows:

Union, $\cup$

The union of two sets A and B is obtained by taking all the elements of A and all the elements of B.

Definition 2.2.1 *Let A and B be sets. We define:* $A \cup B = \{x \mid (x \in A) \text{ or } (x \in B)\}$.

So $\{red, blue\} \cup \{1, 4, -5\} = \{red, blue, 1, 4, -5\}$, and $\mathbb{N} = \mathbb{P} \cup \{0\}$.

Notice that the formal definition of union tells us how to prove that a given set C is equal to $A \cup B$ (i.e. $C = (A \cup B)$): we need to show that for an arbitrary c, $c \in C$ if and only if $c \in A$ or $c \in B$. As usual, this can be done in two steps: we show that if $c \in C$ then $c \in A$ or $c \in B$ (i.e. $C \subseteq (A \cup B)$), and we show that if $c \in A$ or $c \in B$ then $c \in C$ (i.e. $(A \cup B) \subseteq C$).

Example 2.2.2 *Show that the union of the following two sets is* $\mathbb{N}$.

$A = \{x \mid x = 2n \text{ for some } n \in \mathbb{N}\}$
$B = \{x \mid x = 2n + 1 \text{ for some } n \in \mathbb{N}\}$

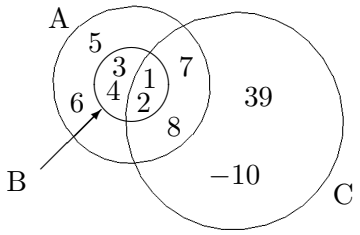
Proof

$\boxed{\mathbb{N} \subseteq (A \cup B)}$ *Let $c \in \mathbb{N}$. We know that c is either even or odd. If c is even then there is $n \in \mathbb{N}$ such that $c = 2n$, and therefore $c \in A$. If c is odd then there is $n \in \mathbb{N}$ such that $c = 2n + 1$, and therefore $c \in B$. Therefore, $c \in A \cup B$.*

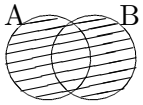
$\boxed{(A \cup B) \subseteq \mathbb{N}}$ *Let $c \in (A \cup B)$. From the definition, we know that either $c \in A$ or $c \in B$. If $c \in A$ we know that there is $n \in \mathbb{N}$ such that $c = 2n$; because $2n \in \mathbb{N}$, it follows that $c \in \mathbb{N}$. If on the other hand $c \in B$ we know that there is $n \in \mathbb{N}$ such that $c = 2n + 1$; because $2n + 1 \in \mathbb{N}$, it follows that $c \in \mathbb{N}$. Therefore, if $c \in (A \cup B)$ then $c \in \mathbb{N}$.*

It is important to explain in a bit more detail what 'an arbitrary c' means. 'Arbitrary' does not mean 'random': we cannot do the proof by picking an element at random as that will prove the property only for the particular element that we pick, even if it is picked at random. Rather, we use a variable such as c above. However, in order for c to be arbitrary, it cannot be used elsewhere in the formulation of the problem. So, for example, we cannot pick n and say that if $n \in A$ then there is $n \in \mathbb{N}$ such that $n = 2n$: this is because the n used in the definition of A is internal to that definition. We show other examples throughout these notes.

A *Venn diagram* is a pictorial representation of sets: sets are drawn as circles, and sets that have elements in common are drawn to overlap. For example, if $A = \{1, 2, 3, 4, 5, 6, 7, 8\}$, $B = \{1, 2, 3, 4\}$, and $C = \{1, 2, 7, 8, 39, -10\}$ we can draw



We often use shading to denote the result of an operation. In the case of union, this would be:



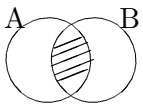
The shaded part is the union of A and B.

Note: Venn diagrams are only used for *illustration*. They are merely pictures and cannot be used to *prove* properties. For example, they cannot be used to show that two sets are equal: having the same 'picture' is not a proof.

Intersection, $\cap$

The intersection of two sets A and B is obtained by taking the elements which are in both A and B.

Definition 2.2.3 *Let A and B be sets. We define: $A \cap B = \{x \mid (x \in A) \text{ and } (x \in B)\}$.*



So $\{1, 3, 4, 17\} \cap \{1, 4, -5\} = \{1, 4\}$, and $\emptyset = \mathbb{P} \cap \{-1, 0, 4.567\}$.

To prove that a given set C is equal to $A \cap B$ we follow the definition: we show that for an arbitrary c, $c \in C$ if and only if $c \in A$ and $c \in B$. We do it in two steps: we show that if

$c \in C$ then $c \in A$ and $c \in B$ (i.e. $C \subseteq (A \cap B)$), and we show that if $c \in A$ and $c \in B$ then $c \in C$ (i.e. $(A \cap B) \subseteq C$).

Example 2.2.4 *Let A, B and C be the sets defined below. Show that $C = (A \cap B)$.*

$A = \{x \mid x = 2n \text{ for some } n \in \mathbb{P}\}$
$B = \{x \mid x = 3m \text{ for some } m \in \mathbb{P}\}$
$C = \{x \mid x = 6k \text{ for some } k \in \mathbb{P}\}$

Proof

$\boxed{C \subseteq (A \cap B)}$ *Let $c \in C$. We know there is $k \in \mathbb{P}$ such that $c = 6k$. Because $6k = 2 \times (3k)$ and $3k \in \mathbb{P}$, this implies that there is $n \in \mathbb{P}$ such that $c = 2n$ ($n = 3k$) and therefore $c \in A$. On the other hand, because $6k = 3 \times (2k)$ and $2k \in \mathbb{P}$, this implies that there is $m \in \mathbb{P}$ such that $c = 3m$ ($m = 2k$) and therefore $c \in B$. Therefore, $c \in A$ and $c \in B$, i.e. $c \in A \cap B$.*

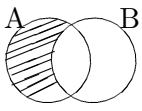
$\boxed{(A \cap B) \subseteq C}$ *Let $c \in (A \cap B)$. From the definition, we know that $c \in A$ and $c \in B$. Because $c \in A$ we know that there is $n \in \mathbb{P}$ such that $c = 2n$, and because $c \in B$ we know that there is $m \in \mathbb{P}$ such that $c = 3m$. This means that $c = 2n = 3m$, i.e. c is a multiple of both 2 and 3. Because the minimum multiple of both 2 and 3 is 6 (they are prime numbers), if follows that c is itself a multiple of 6, i.e. there is $k \in \mathbb{P}$ such that $c = 6k$, which means that $c \in C$. Therefore, if $c \in (A \cap B)$ then $c \in C$.*

Notice that we have chosen a different internal variable for each of the definitions of the sets A, B and C. This was not necessary, but it facilitates the proof. Should the same internal variable be used in more than one definition, we would have to rename them to construct the proof.

Complement, $\setminus$

The complement of B in A is obtained by removing the elements of B from the set A.

Definition 2.2.5 *Let A and B be sets. We define: $A \setminus B = \{x \mid (x \in A) \text{ and } (x \notin B)\}$.*



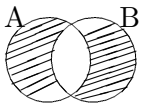
So $\{1, 3, 4, 17\} \setminus \{1, 4, -5\} = \{3, 17\}$ and $\mathbb{P} = \mathbb{N} \setminus \{0\}$.

Exercise 2.2.6 *Given the three sets defined in Example 2.2.4, show that $C \setminus A = B$.*

Symmetric difference, $\oplus$

The symmetric difference of two sets A and B is obtained by taking the elements which are in A or in B, but not in both.

Definition 2.2.7 *Let A and B be sets. We define: $A \oplus B = \{x \mid (x \in A \text{ or } x \in B) \text{ and } x \notin A \cap B\}$.*



So $\{1, 3, 4, 17\} \oplus \{1, 4, -5\} = \{3, 17, -5\}$

Exercise 2.2.8 *Given the sets defined in Example 2.2.4, show that $A \oplus B = (A \cup B) \backslash C$.*

2.2.2 Implementation over bit strings

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set with finitely many elements. We pick an order, $s_1, s_2, \ldots, s_n$ say, for these elements. Then we can represent subsets of S using bit strings (see Section 2.1.6). We define $b_A = (b_1, b_2, \ldots, b_n)$ by putting $b_i = 1$ if $s_i \in A$ and $b_i = 0$ if $s_i \notin A$. The bit string b_A is called the *characteristic vector* of the subset A.

For example, if $S = \{1, 3, 5, 7, 9, 11\}$ then

- if $A = \{3, 7\}$ then $b_A = (0, 1, 0, 1, 0, 0)$
- if $B = \{1, 7, 9\}$ then $b_B = (1, 0, 0, 1, 1, 0)$
- $A \oplus B = \{3, 1, 9\}$ and so $b_{A \oplus B} = (1, 1, 0, 0, 1, 0)$

Notice that $b_{A \oplus B} = (1, 1, 0, 0, 1, 0) = b_A + b_B$ (using binary addition, i.e. $1 + 1 = 0$).

In fact, for all subsets A and B of a set S we have $b_A + b_B = b_{A \oplus B}$. We can use this to give a simple algorithm for finding the symmetric difference of two sets and program it.

We can also give an algorithm using characteristic vectors to find the union of two subsets. We find b_C where $C = A \cup B$ through

```
[for i=1 to n do
if b_A[i]==1 or b_B[i]==1 then b_C[i]=1
else b_C[i]=0
]
```

(Here b[i] is the value held at index i in b.)

$(A \cup B) = (B \cup A)$	commutativy
$(A \cap B) = (B \cap A)$	commutativy
$(A \cup B) \cup C = A \cup (B \cup C)$	associativity
$(A \cap B) \cap C = A \cap (B \cap C)$	associativity
$(A \cup A) = A$	idempotence
$(A \cap A) = A$	idempotence
$A \cup (A \cap B) = A$	absorption
$A \cap (A \cup B) = A$	absorption
$A \subseteq (A \cup B)$	upper bound
$(A \cap B) \subseteq A$	lower bound
$A \subseteq C$ and $B \subseteq C$ imply $(A \cup B) \subseteq C$	least upper bound
$C \subseteq A$ and $C \subseteq B$ imply $C \subseteq (A \cap B)$	greatest lower bound
$A \cup \emptyset = A$	identity
$A \cap \emptyset = \emptyset$	least element
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	distributivity
$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$	distributivity

Figure 2.1: Properties of set operations

2.2.3 Properties of operations on sets

The operations of set union and set intersection that we have just defined satisfy a number of useful properties, some of which are generalised in Chapter 4. Figure 2.1 summarises those properties.

We now prove that some of these properties hold; we recommend that you read those proofs as a way of understanding how a logical argument should be presented (more follows in Section 2.4). For those properties where a proof is not given, their proof is recommended as an exercise.

Proposition 2.2.9 *Given any sets A and B, $(A \cup B) = (B \cup A)$. That is, set union is* commutative.

Proof

$\boxed{(A \cup B) \subseteq (B \cup A)}$ *If $x \in (A \cup B)$ then, by definition of $\cup$, either $x \in A$ or $x \in B$. If $x \in A$ then, by definition of $\cup$, $x \in B \cup A$. If on the other hand $x \in B$ then, by definition of $\cup$, $x \in B \cup A$. So $x \in (B \cup A)$.*

$\boxed{(B \cup A) \subseteq (A \cup B)}$ *If $x \in (B \cup A)$ then, by definition of $\cup$, either $x \in B$ or $x \in A$. If $x \in B$ then, by definition of $\cup$, $x \in A \cup B$. If on the other hand $x \in A$ then, by definition of $\cup$, $x \in A \cup B$. So $x \in (A \cup B)$.*

You will have noticed that the proof of $(B \cup A) \subseteq (A \cup B)$ is essentially the same as that of $(A \cup B) \subseteq (B \cup A)$ in the sense that we just swapped A and B. This is the only time we shall fully spell a proof out like this. We usually say something of the sort 'the argument that shows $(B \cup A) \subseteq (A \cup B)$ is similar'.

Proposition 2.2.10 *Given any sets A and B, $(A \cap B) = (B \cap A)$. That is, set intersection is* commutative.

Proof *If $x \in A \cap B$ then, by definition of $\cap$, $x \in A$ and $x \in B$. So $x \in B \cap A$ and $(A \cap B) \subseteq (B \cap A)$. The reverse inclusion follows similarly.*

Proposition 2.2.11 *Given any sets A, B and C, $(A \cup B) \cup C = A \cup (B \cup C)$. That is, set union is* associative.

Proof *First we prove that the set on the left of the equality is included in that on the right, i.e. that if $x \in (A \cup B) \cup C$ then either $x \in (A \cup B)$ or $x \in C$. We analyse each case separately to conclude that $x \in A \cup (B \cup C)$.*

$\boxed{x \in (A \cup B)}$ *By the definition of $\cup$, we know that either $x \in A$ or $x \in B$. If $x \in A$ then, by the definition of $\cup$, $x \in A \cup (B \cup C)$. If $x \in B$ then, by the definition of $\cup$, $x \in (B \cup C)$ and, again by the definition of $\cup$, $x \in A \cup (B \cup C)$.*

$\boxed{x \in C}$ *By the definition of $\cup$, $x \in (B \cup C)$ and, again by the definition of $\cup$, $x \in A \cup (B \cup C)$.*

The proof that $x \in A \cup (B \cup C)$ implies $x \in (A \cup B) \cup C$ is similar.

Transitivity implies can we can omit brackets and write $A \cup B \cup C$ (and $A \cap B \cap C$) because both possible interpretations — $(A \cup B) \cup C$ and $A \cup (B \cup C)$ — give the same set. Furthermore, because of commutativity the order of the sets is not important either, i.e. $A \cup B \cup C = B \cup C \cup A$. This follows the similar situation in arithmetic: $a + b + c = b + c + a$, etc.

However, $A \cap B \cup C$ is ambiguous as $(A \cap B) \cup C$ and $A \cap (B \cup C)$ are not necessarily the same set. Again, this is similar to arithmetic: $(2 - 1) + 2 \neq 2 - (1 + 2)$ and so $2 - 1 + 2$ is ambiguous.

Proposition 2.2.12 *Given any sets A and B, $A \subseteq (A \cup B)$. That is, $(A \cup B)$ is an* upper bound *of A.*

Proof *If $x \in A$ then $x \in A$ or $x \in B$, which by definition of $\cup$ means that $x \in (A \cup B)$.*

Notice that, by Proposition 2.2.9 (commutativity), it follows that $B \subseteq (A \cup B)$.

Proposition 2.2.13 *Given any sets A and B, $(A \cap B) \subseteq A$. That is, $(A \cap B)$ is a* lower bound *of A.*

Proof *If $x \in (A \cap B)$ then, by definition of $\cap$, $x \in A$ and $x \in B$, which implies that $x \in A$.*

Notice that, by Proposition 2.2.10 (commutativity), it follows that $(A \cap B) \subseteq B$.

Proposition 2.2.14 *Given any sets A, B and C, if $A \subseteq C$ and $B \subseteq C$ then $(A \cup B) \subseteq C$. That is, $(A \cup B)$ is the* least upper bound *of A and B.*

Proof *Assume that $A \subseteq C$ and $B \subseteq C$. To prove that $(A \cup B) \subseteq C$ let $x \in (A \cup B)$. By definition of $\cup$, $x \in A$ or $x \in B$. We analyse each case separately:*

$\boxed{x \in A}$ *Because $A \subseteq C$, we can conclude that $x \in C$.*

$\boxed{x \in B}$ *Because $B \subseteq C$, we can conclude that $x \in C$.*

Therefore, $x \in C$.

Proposition 2.2.15 *Given any sets A, B and C, if $C \subseteq A$ and $C \subseteq B$ then $C \subseteq (A \cap B)$. That is, $(A \cap B)$ is the* greatest lower bound *of A and B.*

Proof *Assume that $C \subseteq A$ and $C \subseteq B$, and let $x \in C$. It follows from $C \subseteq A$ that $x \in A$ and from $C \subseteq B$ that $x \in B$. Therefore, by definition of $\cap$, $x \in (A \cap B)$.*

Proposition 2.2.16 *Given any set A, $A \cup \emptyset = A$.*

Proof *If $x \in A \cup \emptyset$ then, by definition of $\cup$, $x \in A$ or $x \in \emptyset$. By definition of $\emptyset$ we have $x \notin \emptyset$, so we must have $x \in A$.*

Proposition 2.2.17 *Given any set A, $A \cap \emptyset = \emptyset$.*

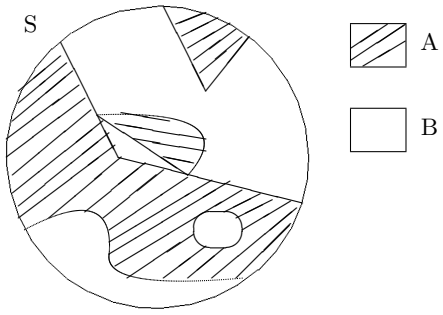
Proof *If $x \in A \cap \emptyset$ then, by definition of $\cap$, $x \in A$ and $x \in \emptyset$, which is impossible because, by definition of $\emptyset$, we have $x \notin \emptyset$.*

2.3 Partitions of sets

We often have to split things up into different cases. For example, when testing programs we divide the input into subsets of 'normal', 'boundary', and 'illegal' cases. Every test case is either normal, boundary or illegal, and no case is both normal and boundary or normal and illegal, etc. This is an example of a partition.

We say that two subsets $A, B \subseteq S$ are a *binary partition* of S if

$$A \cup B = S \text{ and } A \cap B = \emptyset.$$



That is, a binary partition splits the elements of S into two separate cases. If $x \in S$ then either $x \in A$ or $x \in B$ but not both.

The example of program testing required a set to be split into three parts. In fact we can form a partition with any number of subsets.

Definition 2.3.1 *The subsets* $A_1, A_2, \ldots, A_n \subseteq S$ partition *(or are a partition of)* S *if*

- $S = A_1 \cup A_2 \cup \ldots \cup A_n$, *i.e. the* A_i *cover* S*; and*
- *for all* $1 \leq i, j \leq n$, *if* $i \neq j$ *then* $A_i \cap A_j = \emptyset$, *i.e. the* A_i *are pairwise disjoint.*

Example 2.3.2 *Let* $S = \{1, 2, a, X, -3, \pi\}$ *and let* $A = \{1, 2\}$, $B = \{a, -3\}$, $C = \{\pi\}$, $D = \{X\}$. *Then A,B,C, and D partition* S.

Note that the condition 'for all $1 \leq i, j \leq n$, if $i \neq j$ then $A_i \cap A_j = \emptyset$' (i.e., being pairwise disjoint) implies but is NOT equivalent to $A_1 \cap A_2 \cap \ldots \cap A_n = \emptyset$ unless $n = 2$. As an exercise, find three sets A_1, A_2, A_3 such that $A_1 \cap A_2 \cap A_3 = \emptyset$ but $A_1 \cap A_2 \neq \emptyset$.

Partitions will be used in Chapters 3 and 8.

2.4 Proofs

A proposition is a statement that is either true or false. We may not know if is true or false but it must be one or the other. So

- $6 < 10$
- King Henry VIII of England had six wives
- $4 - 1 = 7$
- On 8th December 2019 it will rain in Egham
- On 10th May 1004 there was a Viking named Swen in Winchester

are all propositions, but the statement

This statement is false

is not a proposition (try to determine if it's true or false).

It is important that computer scientists understand what constitutes a proof. Any program is based on an algorithm; for the program to work correctly the algorithm must be correct. Most program implementations are too complex to be able to *prove* them correct (although producing provably correct software is now a major area of research and development). However, an implementation is more likely to be correct if its writer has a good understanding of how proofs are constructed.

Proving something means showing that an implication is true,

(the assumptions) imply (the required proposition).

A proof is a logical argument which ends with the conclusion that some proposition is true. We state the assumptions, or hypotheses, and these form the axioms on which the proof is based.

Logical arguments can assume several forms. In this section, we review some of them.

2.4.1 Direct proofs

Most of the proofs given in Section 2.2.3 are said to be direct proofs: one starts from the assumptions and arrives at the conclusion through a process called natural deduction. At any step, one can use definitions or results already proved. However, if at some point in this process you find yourself using the property you want to prove, something will have gone wrong: start again!

It is important to remember one particular rule. If an assumption is of the form (P or Q) and one wants to derive R, then one should provide two derivations: one of R assuming P, and another of R assuming Q. That is the technique used, for example, in the proof of Proposition 2.2.11: to show that $x \in (A \cup B)$ or $x \in C$ implies $x \in A \cup (B \cup C)$, we analysed each case separately to conclude that $x \in A \cup (B \cup C)$, i.e. we showed how to derive $x \in A \cup (B \cup C)$ from $x \in (A \cup B)$, and then how to derive $x \in A \cup (B \cup C)$ from $x \in C$. The proof was structured as follows

$\boxed{x \in (A \cup B)}$ logical argument to arrive at $x \in A \cup (B \cup C)$ assuming $x \in (A \cup B)$.

$\boxed{x \in C}$ logical argument to arrive at $x \in A \cup (B \cup C)$ assuming $x \in C$.

The same technique was used in the proof of Proposition 2.2.14.

2.4.2 Contrapositive proofs

These are based on the fact that $P \Rightarrow Q$ is equivalent to $\neg Q \Rightarrow \neg P$. Instead of deriving Q from P, we derive $\neg P$ from $\neg Q$. That is, we assume that $\neg Q$ is true, i.e. that Q is false, and prove that $\neg P$ is true, i.e. that P is false. The reason for doing this is that the second derivation may be easier than the first.

For example, we can give a contrapositive proof that if n is a prime number which is not equal to 2 then n is odd.

P: (n is prime) $\wedge$ $\neg$(n=2)
Q: n is odd

We prove that if n is even, then $n = 2$ or n is not prime.

P1: $\neg$Q: n is even
Q1: $\neg$P: $\neg$((n is prime) $\wedge$ $\neg$(n=2)) = (n is not prime) $\vee$ ($n = 2$).

Since n is even we have $n = 2m$, for some m. We know that either $m \leq 0$, or $m = 1$, or $m > 1$, so we analyse each case separately as before:

$\boxed{m \leq 0}$ In this case, we know that also $2m \leq 0$, i.e. $n \leq 0$, so n is not prime, which proves the result.

$\boxed{m = 1}$ In this case, $n = 2$, which proves the result.

$\boxed{m > 1}$ In this case, $n > 2$. But 2 divides n, so n is not prime, which proves the result.

2.4.3 Proofs by contradiction

Another way of proving that $P \Rightarrow Q$ is to assume both that P is true and that Q is false, and derive a contradiction (a proposition which is always false, usually of the form $R \wedge \neg R$). This technique is based on the fact that $P \Rightarrow Q$ is equivalent to $P \wedge \neg Q \Rightarrow$ false.

For example, we can use proof by contradiction to prove the result that, for all integers n, if n is prime and not equal to 2, then n is odd.

We have that $P \wedge \neg Q =$ (n is prime) $\wedge$ $n \neq 2$ $\wedge$ (n is even). Therefore, we assume that n is prime, that $n \neq 2$, and n is even. The fact that n is even means that $n = 2m$, i.e. that 2 divides n. Given that $n \neq 2$, this contradicts the assumption that n is prime.

2.4.4 Existence proofs

Some mathematical propositions involve quantifiers of the form 'for every' or 'for some'. For example, the properties listed in Section 2.2.3 are all of the form 'Given any sets...', which is the same as 'For all sets...': these are called *universal quantifiers*.

An example of a definition that involves 'for some' (i.e, an *existential quantifier*) is the definition of even integer: an integer n is even if there exists another integer m such that $n = 2m$. We have used this definition before. To prove that, for example, that 6 is even, one than needs to find an integer m such that $6 = 2m$. This can be done by exhibiting one such m, in this case $m = 3$.

In summary, to prove that a predicate $P(n)$ (such as 'n is even') is true *for some value* of n we just have to exhibit one value of n for which $P(n)$ is true. (We sometimes write '$\exists n.P(n)$ is true' if there exists some value of n for which $P(n)$ is true.)

Existence proofs are often *constructive* – they involve producing an algorithm that constructs an element with the property. Another example of an existence proof can be found in the proof of Proposition 2.2.4. At some point, we had to prove that there was $n \in \mathbb{P}$ such that $c = 2n$, which we did by exhibiting $n = 3k$ for a k that we had already picked satisfying $c = 6k$. This is a simple example of such an algorithm.

Existence proofs are also used to produce *counter-examples* for propositions of the form 'for all x, $P(x)$', i.e. to prove that such universal statements are *false*: we just need to exhibit one value of x for which $P(x)$ is false — this value is called a *counter-example*. (We sometimes write $\forall n.P(n)$ if $P(n)$ is true for all values of n, and $\exists n \neg P(n)$ if $P(n)$ has a counter-example.)

For example, consider the proposition 'for all integers n, if n is prime then $2n + 1$ prime'. To disprove it, let $n = 7$. It follows that $2n + 1 = 15$, which is not prime. So 7 is a counter-example for that proposition.

Another example is the proposition: for all sets A, B, and C, $(A \cap B) \cup C = A \cap (B \cup C)$. If we choose $A = \emptyset$, $B = C = \mathbb{P}$, we obtain $\mathbb{P} = \emptyset$, which is false.

2.4.5 Proofs of universal statements

These are proofs that statements of the form 'for all x, $P(x)$' are true. This usually done by picking an arbitrary x, and showing that $P(x)$ is true. The subtlety here is on how to choose such an arbitrary x. We discussed this already after Definition 2.2.1 and Example 2.2.2, and we will see more examples further on. We have also discussed in relation to Examples 2.1.5 and 2.2.2 the use of internal variables in definitions and how they cannot be confused with each other when making proofs using those definitions; more examples will follow in later sections.

2.4.6 Further reading

Sections 1.7 and 1.8 of Rosen contain much more detailed material, including very useful examples of alleged proofs that are not real proofs: these expose you to the pitfalls of reasoning when rules are not correctly applied!

Other recommended reading, especially if the topic interests you, are the books:

Ethan D. Bloch. *Proofs and Fundamentals.* (Library code 511.2BLO) — This textbook covers the topics addressed in this section, and its Chapter 2 is especially good at explaining what a proof is, and how to write one.

Daniel J. Velleman. *How to Prove It: A Structured Approach.* (Library code 511.2VEL) — This textbook is very useful when learning what it means to prove something. Again, it covers the topics addressed in this section, but goes into much more detail on what a proof is, using an analogy between proving and programming.

2.5 Why isn't everything a set? – Russell's paradox

The elements of a set don't need to have anything in common, and their elements can themselves be sets.

$$\{4,\ \text{the tower of London},\ \{-31,\ \text{my car},\ \{\}\},\ \text{apollo 11},\ \{\},\ 9012\}$$

So why do we have sets, isn't every collection of objects a set?

It turns out that if we just say that a set is any collection of objects then we end up with a *paradox*, a statement which is both true and false.

Mathematical paradoxes play a very important role in Science. Until the beginning of the 20th century people did not realise that such paradoxes exist. Then in 1931 a Mathematician called Kurt Gödel proved that in any formal mathematical system specified by a set of axioms and rules either there is a true proposition which cannot be proved true using the rules of the system, or the system contains a proposition which can be proved both true and false using the rules of the system!

This has implications for what we try to do as computer scientists. There is no point in trying to build a computer that can prove everything that's true about the complete system that it can represent. For example, we cannot expect a computer to be able to derive all the true statements in arithmetic.

In 1937 Alan Turing put the problem at the heart of computing when he showed that the halting problem is undecidable. When we write a program we would like to be able to tell if it will terminate, even if we have to wait a very long time for it to do so. Turing showed that there cannot exist an algorithm which, given any program, will be able to decide whether or not that program will terminate. Of course, there are lots of programs that we can decide will terminate, or will not terminate. But there is no hope of writing a computer program which will decide whether or not any given program will terminate.

Proving, or even understanding the basic arguments for, the halting problem and Gödel's incompleteness theorem requires some mathematical skill. But one of the earliest mathematical paradoxes, discovered by the mathematician Russell in 1901, can be formulated using only the concepts of set membership.

Some sets are clearly not elements of themselves. For example, the set $\mathbb{S}_4$ of integers from 1 to 4 is not itself an integer between 1 and 4, but it is an element of the set $\{3,\ \text{desert orchid},\ \mathbb{S}_4,\ \text{Emma}\}$, and of the set of subsets of $\mathbb{S}_6$.

There are a lot of sets which do not have $\mathbb{S}_4$ as an element. In fact we can form the set $\mathcal{S}$ of all sets which do not have $\mathbb{S}_4$ as an element, and $\mathbb{S}_4$ is an element of this set. Since $\mathbb{S}_4$ is an element of $\mathcal{S}$, we have that $\mathcal{S}$ is not an element of itself.

OK, so what about the collection Θ of all those sets which are not elements of themselves. Is Θ a set?

If Θ is a set then we can ask is Θ an element of itself? If the answer is no then it is also yes, and if it's yes then it is also no!

Thus we have that the collection

$$\Theta = \{S \mid S \text{ is a set}, S \notin S\}$$

cannot be a set. This is known as Russell's paradox and it shows that not every collection of objects which can be defined by a given property is a set.

If you want to know more about the formal rules which determine when a collection of objects is a set, you should look up the work of Georg Cantor and then the work of Bertrand Russell. For this module, you do not need to know the formal definition of a set and it is safe to proceed with the intuitive understanding used so far. In particular, all the operations that we present in the rest of these notes produce sets when applied to sets.

2.6 Look up on the Web

Georg Cantor Born in Russia in 1845, while Cantor was still a child his family moved to Germany. As well as his work on set theory, Cantor produced the formal definition of the 'real' numbers in terms of Dedekind cuts, and proved that the rational numbers are countable and that the real numbers are not. Cantor suffered periodic bouts of mental ill health, which grew worse as he got older. He never met Russell, because his health forced him to cancel a planned visit. But Cantor's work took the study of orderings and set theory from their naive setting in the mid 1800's to the modern approach, and forms the foundations of the work continued by Russell, Gödel and Turing.

Bertrand Russell Born in England in 1872, Russell worked at Trinity College Cambridge. He was a pacifist who was imprisoned in 1918 and again in 1961 (when he was 89!) for his pacifist beliefs. He won the Nobel Prize for literature in 1950, but his most important work is the Principia Mathematica which he wrote with Alfred Whitehead. Whitehead's nephew Henry Whitehead was Professor of Pure Mathematics at Oxford, where he was the D.Phil supervisor of Graham Higman. Higman was also Professor of Pure Mathematics at Oxford, where he was my D.Phil supervisor!

Kurt Gödel Born in 1906 in what was then Austria but is now the Czech Republic, Gödel did his work on his incompleteness theorem at the University of Vienna in the 1920's. All his life Gödel was worried about his health and at the start of the second World War he moved to America, probably to avoid conscription which he felt his health would not survive. In America Gödel he became a close friend of Albert Einstein.

2.7 Exercises

1. List the elements of the set $\{x \mid x \in \mathbb{N}, -2 \leq x < 4\}$.
2. Write the set $\{1, 2, ..., n\}$ in predicate form
   (i) using $\mathbb{S}_n$
   (ii) without using $\mathbb{S}_n$.
3. For $A = \{1, 2, 3, 4, 5\}$, $B = \{2, 5, a, b\}$, and $C = \{a, 2\}$ write down
   $A \cup B, \quad B \cup C, \quad A \cap B, \quad B \cap C, \quad A \backslash C, \quad A \oplus C.$
4. Draw Venn diagrams for $A \oplus (B \oplus C)$ and $(A \oplus B) \oplus C$.
5. Write out the power set of $\{1, 2, 3, 4\}$.
6. Let $S = \{a, b, c, d, e, f, g\}$ and suppose that the elements of S are ordered alphabetically. If $A = \{c, g, a, c\}$, write down b_A, the characteristic vector of A.
   Which subset has characteristic vector $(0, 1, 1, 1, 0, 1, 0)$?
7. Calculate the number of subsets of cardinality 4 in the set $\{1, 2, 3, ..., 23\}$.
8. Let $A = \{a, b, c\}$, $B = \{\alpha, \beta\}$. Write out the elements of $A \times B$ and $B \times A$.
9. How many elements are there in $\mathbb{S}_{14} \times \{x, y, z\}$?
10. Let $S = \{1, 2, 3, a, b, c\}$. Write down three sets A, B, C such that A, B, C partition S.
11. Prove that $A \oplus B = (A \backslash B) \cup (B \backslash A)$.
12. Prove the following identities, which are called the *distributive laws*.
    (i) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.
    (ii) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.
13. Prove that $A \cap (B \cap C) = (A \cap B) \cap C$.
14. Prove that $(A \oplus B) \backslash B = A \backslash B$.

15. Prove that $A \backslash B = A \backslash (A \cap B)$ and $A \backslash B = (A \cup B) \backslash B$.

16. Let $|S| = n$. Write out an algorithm for calculating the intersection of two subsets which are represented as bit strings.

17. Calculate the number of subsets of cardinality 3 in the set $\{x \mid x \in \mathbb{N}, -1 \leq x \leq 17\}$.

18. Find all possible partitions of the set $\{1, 2, 3\}$.

19. If $|A| = n$ and $|B| = m$, what is $|\mathcal{P}(A \times B)|$?

Chapter 3

Relations

It is natural to compare things – your pudding is larger than mine, she is taller than her brother. We often want to divide a collection of objects into groups, each group having some distinguishing property that makes it relevant for the problem at hand. For example, we could divide this class up into groups according to the degree course for which each person is registered. Then two students are related if they are registered for the same course and one can query about all students that take the same course of any given student. A large amount of the world's computing power is being used to classify data into classes.

Relations form the basis one of the mostly widely used types of database – relational databases. Relations are also the basis of the Semantic Web: these are so-called 'triples' that relate subjects, predicates and objects, which together establish ontologies.

Another typical example are inheritance hierarchies in object-oriented programming languages: two classes are related if one inherits from another. Inheritance allows for the features of a class to be used by the classes that inherit from it, which is a powerful mechanism for structuring object-oriented programs.

In this chapter we study relations, in particular equivalence relations. In the next two chapters, we study two particular kinds of relations: orderings and functions.

(*Most of the material in this chapter can be found in Sections 9.1 to 9.5 of Rosen.*)

3.1 Definition and examples

Consider the set of all undergraduate students in the College, and the set of all courses offered by departments in the College. The students and the courses are related in an obvious way: a student is related to a course if he/she is currently registered for the course. This is an example of a relation. Most students are related to several courses, and most courses are related to several students.

Relations are just sets of ordered pairs. If S is the set of all students in the College, and if C is the set of all courses then we form the ordered pair (s, c) if student s is registered for course c. So, for example, we may have (*John Brown*, CS1860).

If R is the relation 'is registered for' then we have

$$R = \{(s, c) \mid s \in S, c \in C, s \text{ is registered for } c\}.$$

So $R \subseteq S \times C$. We could also write

$$R = \{(s, c) \in S \times C \mid s \text{ is registered for } c\}.$$

Definition 3.1.1 *Let S and T be sets. A* (binary) relation *between S and T is a set of ordered pairs, i.e. a subset of $S \times T$.*

Example 3.1.2 *Let $S = \{\text{int}, \text{float}, \text{char}, \text{double}\}$ be a set of types of a programming language and $T = \{\text{term}, \text{amount}, \text{total\_cost}, \text{finished}, \text{answer}\}$ be variable names from a program in that language. Then we can define a relation R between S and T by saying that s is related to t if $t \in T$ is of type s. For example:*

$$R = \{(\text{int}, \text{term}), (\text{int}, \text{amount}), (\text{int}, \text{finished}), (\text{float}, \text{total\_cost}), (\text{char}, \text{answer})\}$$

This is a very useful relation that compilers may store in order to report errors when a value is assigned to a variable of a different type.

It follows from the definition that the set of all relations between S and T is the powerset $\mathcal{P}(S \times T)$ of the cartesian product of S and T. Therefore, if $|S| = n$ and $|T| = m$ then there are $2^{n \times m}$ different relations that can be defined between S and T.

Example 3.1.3 *Write down all the relations between $\{a, b\}$ and $\{1, 2\}$.*

$$\begin{array}{ll}
R_1 = \{(a,1),\ (a,2),\ \ (b,1),\ (b,2)\} & R_2 = \{(a,2),\ (b,1),\ (b,2)\} \\
R_3 = \{(a,1),\ (b,1),\ (b,2)\} & R_4 = \{(a,1),\ (a,2),\ (b,2)\} \\
R_5 = \{(a,1),\ (a,2),\ (b,1)\} & R_6 = \{(b,1),\ (b,2)\} \\
R_7 = \{(a,2),\ (b,2)\} & R_8 = \{(a,2),\ (b,1)\} \\
R_9 = \{(a,1),\ (b,2)\} & R_{10} = \{(a,1),\ (b,1)\} \\
R_{11} = \{(a,1),\ (a,2)\} & R_{12} = \{(b,2)\} \\
R_{13} = \{(b,1)\} & R_{14} = \{(a,2)\} \\
R_{15} = \{(a,1)\} & R_{16} = \emptyset
\end{array}$$

There are indeed $2^{2\times 2} = 2^4 = 16$ relations between $\{a, b\}$ and $\{1, 2\}$. Notice that one of those relations is $\emptyset$, i.e. the relation in which no element of $\{a, b\}$ is related to any element of $\{1, 2\}$.

Sometimes, a relation is defined over elements of a single set.

Definition 3.1.4 *We call a relation between a set A and itself a* relation on A.

Example 3.1.5 *Let $S = T = \mathbb{S}_4$ and say that s is related to t if s is bigger than or equal to t. Then the relation is* $\{(1,1),(2,1),(2,2),(3,1),(3,2),(3,3),(4,1),(4,2),(4,3),(4,4)\}$.

The symbol $\geq$ is normally used for this relation. It would be awkward (though correct) to write $(2,1) \in \geq$: everyone normally writes $2 \geq 1$ instead. That is, we often use *infix* notation and write $a\ R\ b$ instead of $(a,b) \in R$.

Example 3.1.6 *Given any set A, one of the possible relations on A is the* identity*, which we define as $id_A = \{(s,s) \in A \times A \mid s \in A\}$. As usual, we write $s = t$ instead of $(s,t) \in id_A$.*

Example 3.1.7 *Let $C = \{\text{Vehicle}, \text{Car}, \text{Bicycle}, \text{MountainBike}\}$ be a set of classes of a Java program. Then we can define a relation on C by saying that c_1 is related to c_2 if c_1 inherits from c_2. For example, car* inherits-from *vehicle, bike* inherits-from *vehicle, mountainbike* inherits-from *bike.*

The most general way of describing a relation is to use the predicate notation for sets, making the predicate the condition for two objects to be related.

Example 3.1.8 *Let p be a prime number, and let $n \in \mathbb{N}$. Then let $n\%p$ be the remainder when n is divided by p. So there is some integer k such that $n = kp + n\%p$. For example,* $5\%2 = 1,\ 4\%2 = 0,\ 8\%3 = 2,\ p\%p = 0,\ (p+1)\%p = 1$.

We define a relation R on $\mathbb{N}$ by

$$R = \{(n,m) \mid n\%p = m\%p\}$$

Notice that this is the same as the relation you may have met before:

$$R = \{(n,m) \mid n \equiv m \text{ (modulo } p)\}$$

We have $(n,m) \in R$ if and only if $n \equiv m$ (modulo p).

(If you have not met the relation 'modulo p' before, you can read more about it in Section 2.3 of Rosen.)

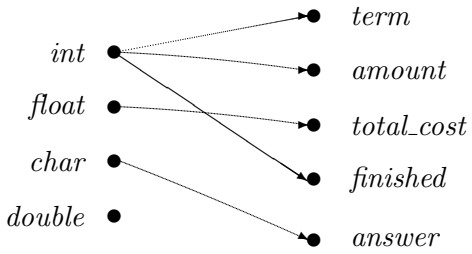
3.2 Representations of relations

3.2.1 Graphical representation

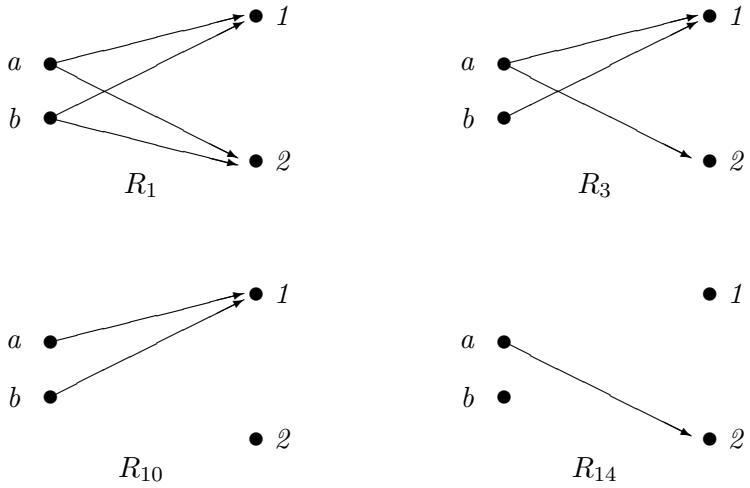
When there are only a few elements in each of the sets involved in the relation then we can use a graphical representation.

We have already mentioned that a graph is a set of vertices (nodes) and a set of edges between them. A *digraph* – di(rected) graph – is a graph whose edges are arrows, i.e. they have a source and a target node. We can use digraphs to represent relations. The nodes of the digraph are the elements of the two sets, and there is an arrow from one node to another if the two nodes are related.

Example 3.2.1 *A graphical representation of the relation defined in Example 3.1.2 between types and variables is*

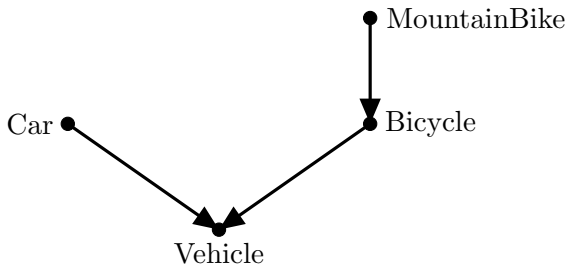


Example 3.2.2 *The following are graphical representations of the relations R_1, R_3, R_{10} and R_{14} defined in Example 3.1.3 between $\{a, b\}$ and $\{1, 2\}$.*



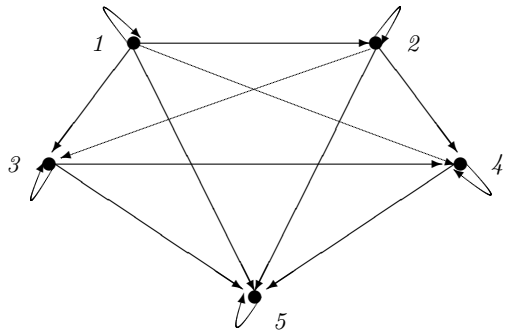
If a relation is defined on a set A then we only list the nodes of A once.

Example 3.2.3 *A graphical representation of the inheritance relation defined in Example 3.1.7 between classes is*



Notice that we may end up with graphs that look different because there can be edges in both directions between two nodes and there can be an edge from a node to itself.

Example 3.2.4 *The graphical representation of* $\leq$ *on* $\mathbb{S}_5$ *is*



3.2.2 Matrix representation

A matrix is a two-dimensional array. Matrices have rows and columns. We refer to an entry in the matrix by stating its row number and then its column number. For example:

$$M = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

is a matrix with three rows and four columns.

If M is a matrix with n rows and m columns, we say that M is an $n \times m$ matrix (said 'n by m'). We use $M(i, j)$ to denote the entry in the ith row and jth column of M. So, for the above matrix, $M(2, 4) = 1$.

We can use matrices whose entries are either 1 (true) or 0 (false) as above to describe relations. We use the elements of the first set to label the rows and the elements of the second set to label the columns. Then we put 1 in the (s, t) entry if $(s, t) \in R$, and 0 in the entry otherwise.

Example 3.2.5 *The following matrix represents the relation $\leq$ on $\mathbb{S}_3$.*

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

If R is a relation between two finite sets A and B then we pick an order for the elements of the sets, say $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_m$. We then build an $n \times m$ matrix M such that $M(i,j) = 1$ if $(a_i, b_j) \in R$, and is 0 otherwise.

Example 3.2.6 *Suppose that $A = \{u, w, y\}$ and $B = \{1, 3, 5, 7\}$ and*

$$R = \{(u,1), (w,1), (u,3), (u,5), (w,5), (y,5), (y,7)\}.$$

Then the matrix representation of R is

$$\begin{array}{c} \\ u \\ w \\ y \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 3 & 5 & 7 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \end{array}.$$

Of course, we can reconstruct the relation from its representation matrix.

Example 3.2.7 *Consider the relation between the set $E = \{e_1, e_2, e_3, e_4\}$ of text editors supported by a given software company and the set $F = \{f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8\}$ of features (packages, classes) that are used by those text editors. Whenever a change needs to be made to one of those features, it is important for the company to know which editors use which features, which it does through the following table/matrix where editors are in rows and features in columns:*

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

The relation is therefore the set

$$\left\{ \begin{array}{llllllll} (e_1,f_1), & (e_1,f_2), & (e_1,f_3), & (e_1,f_4), & & & & \\ (e_2,f_1), & (e_2,f_2), & (e_2,f_3), & (e_2,f_4), & (e_2,f_7), & (e_2,f_8), & & \\ (e_4,f_1), & (e_4,f_2), & (e_4,f_3), & (e_4,f_4), & (e_4,f_5), & (e_4,f_6), & (e_4,f_7), & (e_4,f_8) \end{array} \right\}$$

3.3 Inverse and complement relations

We have looked at two relations $<$ and $\leq$ on $\mathbb{Z}$. There are two corresponding relations $>$ and $\geq$ where: $n < m$ if and only if $m > n$, and $n \leq m$ if and only if $m \geq n$. These are examples of inverse relations.

Definition 3.3.1 *If we have a relation R on a set A we can form another relation R^{-1} on A using the rule*

$$a \; R^{-1} \; b \quad \textit{if and only if} \quad b \; R \; a.$$

The relation R^{-1} is called the inverse of R.

For example, $>$ is the inverse of $<$ and $\geq$ is the inverse of $\leq$.

Note that we have $a \; (R^{-1})^{-1} \; b$ if and only if $b \; R^{-1} \; a$, which is equivalent to $a \; R \; b$. So the inverse of R^{-1} is R, as we might hope.

Also, looking at the relations $<$ and $\geq$ on $\mathbb{Z}$ we see that $m \geq n$ if and only if $m \not< n$. We say that $\geq$ is the complement of $<$.

Definition 3.3.2 *We define the complement $\overline{R}$ of R by the rule*

$$a \; \overline{R} \; b \textit{ if and only if } (a, b) \notin R.$$

3.4 Equivalence relations

Some relations have a number of properties that make them 'special' in the sense of the role that they play in modelling real-world situations, which ultimately influences the way systems are programmed to be used in those situations. One such category of relations is characterised by the following three properties:

Definition 3.4.1 *Let R be a relation on a set A.*

- *R is* reflexive *if $(a, a) \in R$ for all $a \in A$.*
- *R is* symmetric *if, for all $a, b \in A$, we have $(a, b) \in R$ implies that $(b, a) \in R$.*
- *R is* transitive *if, for all $a, b, c \in A$, we have $(a, b), (b, c) \in R$ implies that $(a, c) \in R$.*

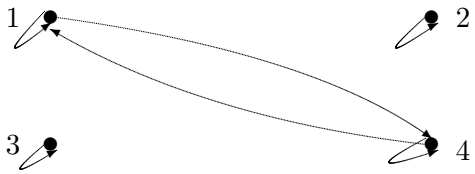
Example 3.4.2 *The relation $\leq$ on $\mathbb{S}_n$ is reflexive and transitive, but it is not symmetric if $n \geq 2$. For example, $1 \leq 2$ but $2 \not\leq 1$.*

Example 3.4.3 *The relation $\equiv_3$ defined on $\mathbb{N}$ by the rule*

$$n \; \equiv_3 \; m \quad \textit{if and only if} \quad n = m \text{ (modulo 3)},$$

is reflexive, symmetric and transitive.

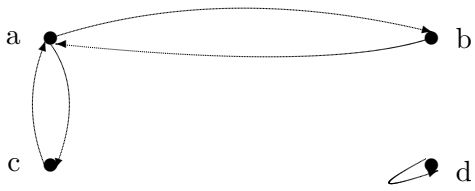
If you draw a graphical representation of a reflexive relation you will find that every node has an edge going to itself. For example, the graphical representation of $\equiv_3$ on $\mathbb{S}_4$ is



If you write out the matrix representation of a reflexive relation you will find that there are 1's down the leading diagonal. For example, the matrix representation of $\equiv_3$ on $\mathbb{S}_4$ is

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

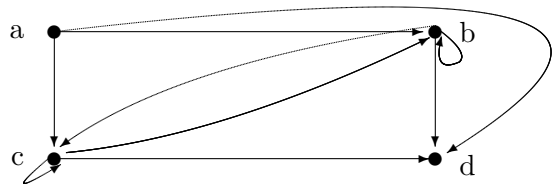
If you draw a graphical representation of a symmetric relation you will find that if there is an edge in one direction between two nodes, then there is an edge in the other direction. For example, The graphical representation of $R = \{(a, b), (b, a), (a, c), (c, a), (d, d)\}$ on $A = \{a, b, c, d\}$ is



If you write out the matrix representation of a symmetric relation you will find that the matrix is symmetric about the leading diagonal. For example, the matrix representation of $R = \{(a, b), (b, a), (a, c), (c, a), (d, d)\}$ on $\{a, b, c, d\}$ is

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

If you draw a graphical representation of a transitive relation you will find that if there is an edge from node A to node B, and if there is an edge from node B to node C, then there is an edge directly from node A to node C. For example, The graphical representation of $R = \{(a, b), (b, b), (b, c), (a, c), (c, b), (c, c), (b, d), (c, d), (a, d)\}$ on $\{a, b, c, d\}$ is



If you are using a graph or a matrix to represent a relation in a computer program, then you can exploit the special properties that these representations have when the relation is known to be reflexive or symmetric. For example, if the relation is symmetric then it is only necessary to store the 'upper triangular' half of the matrix representation.

It is easy to tell whether a relation is reflexive or symmetric from either the graphical or the matrix representations. In general it is hard to tell whether a relation is transitive.

Definition 3.4.4 *A relation on a set A that is reflexive, symmetric and transitive is called an* equivalence relation *on A.*

An equivalence relation is an abstraction (generalisation) of equality. We think of objects as being equivalent (or, in some way, the "same") if, for all purposes and intents, they are indistinguishable. As you can imagine, this can have huge effects in the way we classify and store data (for example, in databases) or process that data through computer programs.

For example, in day-to-day life, any two bank notes are the same as long as they have the same denomination (and are not fake, of course). No two (good) bank notes are really the same – they will have different serial numbers – but this is something that you tend to ignore unless, say, you are investigating money laundering. Therefore, a software system managing an ATM might well keep a record of how many ten-pound notes it currently has in order to decide if it can give you the amount you requested, but it would probably not record their serial numbers (i.e., which exact notes it has).

The same applies to cars. No two cars are the same: each car has a unique vehicle identification number (VIN), which is a series of numbers and letters that identify the model year, the engine code, and so on. However, when you are about to buy a new car, you will probably pay more attention to features such as the model, the colour, and so on, than the full VIN itself (if you ever get to know it). Therefore, all car search engines will allow you to filter according to some features, but the VIN will hardly ever be one of them – tough luck if what you really want is a car whose VIN ends with a prime number!

Equivalence relations arrange objects into classes ("pots") according to the way we want to distinguish them. For example, consider the relation *has-the-same-value-as* on the set $\mathbb{BN}$ of all bank notes defined by, for all $b_1, b_2 \in \mathbb{BN}$, $b1$ *has-the-same-value-as* b_2 if and only if they have the same denomination, i.e. if they are both five-pound, or ten-pound, or twenty-pound, or fifty-pound notes. This is an equivalence relation and it divides $\mathbb{BN}$ into four sets: the set of all five-pound notes, the set of all ten-pound notes, the set of

all twenty-pound notes, and the set of all fifty-pound notes. These four sets are pairwise disjoint – no bank note has more than one denomination – and their union is the whole of $\mathbb{BN}$ because there are no bank notes of other denominations (at least at the time these notes are being written); therefore, they form a partition of $\mathbb{BN}$ (see Section 2.3 on partitions).

Definition 3.4.5 *Let A be any non-empty set, let a be an element of A, and let R be an equivalence relation on A. The* equivalence class of a *is the subset of A that contains all the elements that are related to a under R.*

We use the notation $[a]$ (or $[a]_R$ if we want to make the equivalence relation explicit) for the equivalence class of a. Therefore,

$$[a]_R = \{b \in A \mid (a, b) \in R\}.$$

Example 3.4.6 *Define a relation R on $\mathbb{Z}$ by saying that nRm if the difference $(n - m)$ is divisible by 4. Show that R is an equivalence relation and find all the equivalence classes relative to R.*

Solution To prove that it is an equivalence relation, let us consider each property in turn:

reflexive $n - n = 0$, which is divisible by 4, so nRn for all $n \in \mathbb{Z}$, which means that R is reflexive.

symmetric $m - n = -(n - m)$, so if $n - m$ is divisible by 4 then so is $m - n$. Thus nRm implies mRn, and R is therefore symmetric.

transitive If $n - m$ and $m - r$ are divisible by 4, then $n - m = 4k$ and $m - r = 4h$ for some integers k, h. This implies that $n - r = n - m + (m - r) = 4(k + h)$ and, therefore, $n - r$ is divisible by 4, so nRr. Thus R is transitive

So R is an equivalence relation on $\mathbb{Z}$.

To find all the equivalence classes relative to R, notice that every integer can be written in one of the following four forms:

$$4h, \quad 4h + 1, \quad 4h + 2, \quad 4h + 3.$$

If n and m are both of the same form then $n - m$ is divisible by 4. That is, if $n = 4h + r$ and $m = 4k + r$ for some h, k and r, then $n - m = 4(h - k)$.

If n and m are of different forms then $n - m$ is not divisible by 4. For example, if $n = 4h + 1$ and $m = 4k + 2$ then $n - m = 4(h - k) - 1$.

So the equivalence classes are the subsets which contain elements of the same form:

$$\{\ldots, -8, -4, 0, 4, 8, 12, \ldots\} = \{4h \mid h \in \mathbb{Z}\} = [0]$$
$$\{\ldots, -7, -3, 1, 5, 9, 13, \ldots\} = \{4h + 1 \mid h \in \mathbb{Z}\} = [1]$$

$$\{\ldots,-6,-2,2,6,10,14,\ldots\} = \{4h+2 \mid h \in \mathbb{Z}\} = [2]$$
$$\{\ldots,-5,-1,3,7,11,15,\ldots\} = \{4h+3 \mid h \in \mathbb{Z}\} = [3]$$

Every element of $\mathbb{Z}$ is in one of the equivalence classes, and these classes are mutually disjoint. Again, we have a partition of the set.

The following theorem shows that there is a direct correspondence between equivalence relations and partitions of a set.

Theorem 3.4.7 *Let A be a set.*

(i) If R is an equivalence relation on A then the equivalence classes relative to R form a partition of A.

(ii) If $A_1, A_2, \ldots, A_n$ is a partition of A then the relation

$$R = \{(a,b) \mid (a \in A_i \text{ and } b \in A_i) \text{ for some } 1 \leq i \leq n\}$$

is an equivalence relation on A.

Proof

(i) We need to prove that the equivalence classes relative to R are mutually disjoint and that their union is A.

To prove that they are mutually disjoint, consider two equivalence classes that have a non-empty intersection, $[a] \cap [b] \neq \emptyset$. Choose $c \in [a] \cap [b]$. Because $c \in [a]$ and $c \in [b]$, we know that $a\ R\ c$ and $b\ R\ c$. Because R is symmetric, we know that cRb. Finally, because R is transitive, it results that $a\ R\ b$. Therefore, $b \in [a]$, which implies that $[a] = [b]$ (you may prove this more thoroughly as an exercise). In conclusion, we have proved that $[a] \neq [b]$ implies $[a] \cap [b] = \emptyset$.

Now we need to prove that every element of A belongs to an equivalence class. This is easy because, given $a \in A$, the fact that R is reflexive tell us that aRa and, therefore, $a \in [a]$.

(Notice that the proof used all the three properties of R: reflexivity, symmetry and transitivity.)

(ii) Let now $A_1, A_2, \ldots, A_n$ be a partition of the set A. We have to check that the relation R is reflexive, symmetric and transitive.

To show that it is reflexive, let $a \in A$. We know that $a \in A_i$ for some i, and therefore aRa (a is in the same A_i as itself).

To show that it is symmetric, let $a, b \in A$ such that $a\ R\ b$. This means that there is an i such that $a, b \in A_i$, i.e., a and b are in the same set of the partition. But then so are b and a, so $b\ R\ a$.

To show that it is transitive, let $a, b, c \in A$ such that $a\ R\ b$ and $b\ R\ c$. This means that there is an i such that $a, b \in A_i$, and there is a j such that $b, c \in A_j$. But then $b \in A_i$ and

$b \in A_j$, so $A_i \cap A_j \neq \emptyset$. Because the sets that form the partition are mutually disjoint, this can only happen if $A_i = A_j$ and, therefore, also $c \in A_i$. Therefore, $a\ R\ c$.

(Notice that the proof used all the properties of being a partition.)

3.5 Closure of relations

A relation that we often use, especially when travelling in big cities, is given by the map of the metro (as an example, London's tube map is shown in Figure 3.1) so that we can find out how to travel from one station to another. A directed graph of that relation is easily obtained from the map by noticing that one can travel in both directions between any two stations (i.e. the relation is symmetric) and, therefore, making each edge of the graph bi-directional. If *STATIONS* is the set of all London tube stations, the relation on *STATIONS* thus obtained could be named *is-one-stop-from*. For example, *Waterloo is-one-stop-from Embankment*.

One is often more interested in travelling from Waterloo to, say, Leicester Square. This is possible because we also get from the map that *Embankment is-one-stop-from Charing Cross* and *Charing Cross is-one-stop-from Leicester Square*. This means that, by looking at the tube map, one often thinks of the induced relation between any two stations that is obtained by iterating *is-one-stop-from*. This relation could be named *is-connected-to* and defined, for any two stations s and t, by s *is-connected-to* t if and only if there exists a sequence $s_0, \ldots, s_{n+1}$ of stations such that $s_0 = s$ (i.e. we start at s), $s_{n+1} = t$ (i.e. we finish at t) and, for every $0 \leq i \leq n$, s_i *is-connected-to* s_{i+1} (i.e. any station in the sequence except for the last one is *is-one-stop-from* the next. The relation *is-connected-to* is obviously transitive (which was the purpose), it contains *is-one-stop-from* (i.e. travelling just one stop is a particular case of a journey across the map), and is in fact the smallest transitive relation that contains *is-one-stop-from* (i.e. it does not add more journeys than those obtained by following the map).

This is an example of what is called a *transitive closure*:

Definition 3.5.1 *The transitive closure of a relation R on a set A, often denoted R^+, is defined by, for any $a, b \in A$, $a\ R^+\ b$ if and only if:*

- *there exist $a_0, \ldots, a_{n+1} \in A$ such that $a_0 = a$, $a_{n+1} = b$; and*
- *for every $0 \leq i \leq n$, $a_i\ R\ a_{i+1}$.*

That is, we have $a\ R\ a_1\ R \ldots a_n\ R\ b$. We call such a sequence $a_0, \ldots, a_{n+1}$ a *path* of length $n+1$ in R (a concept that we will meet again when we study graphs). Therefore, R^+ only adds to R elements connected by paths of length at least one.

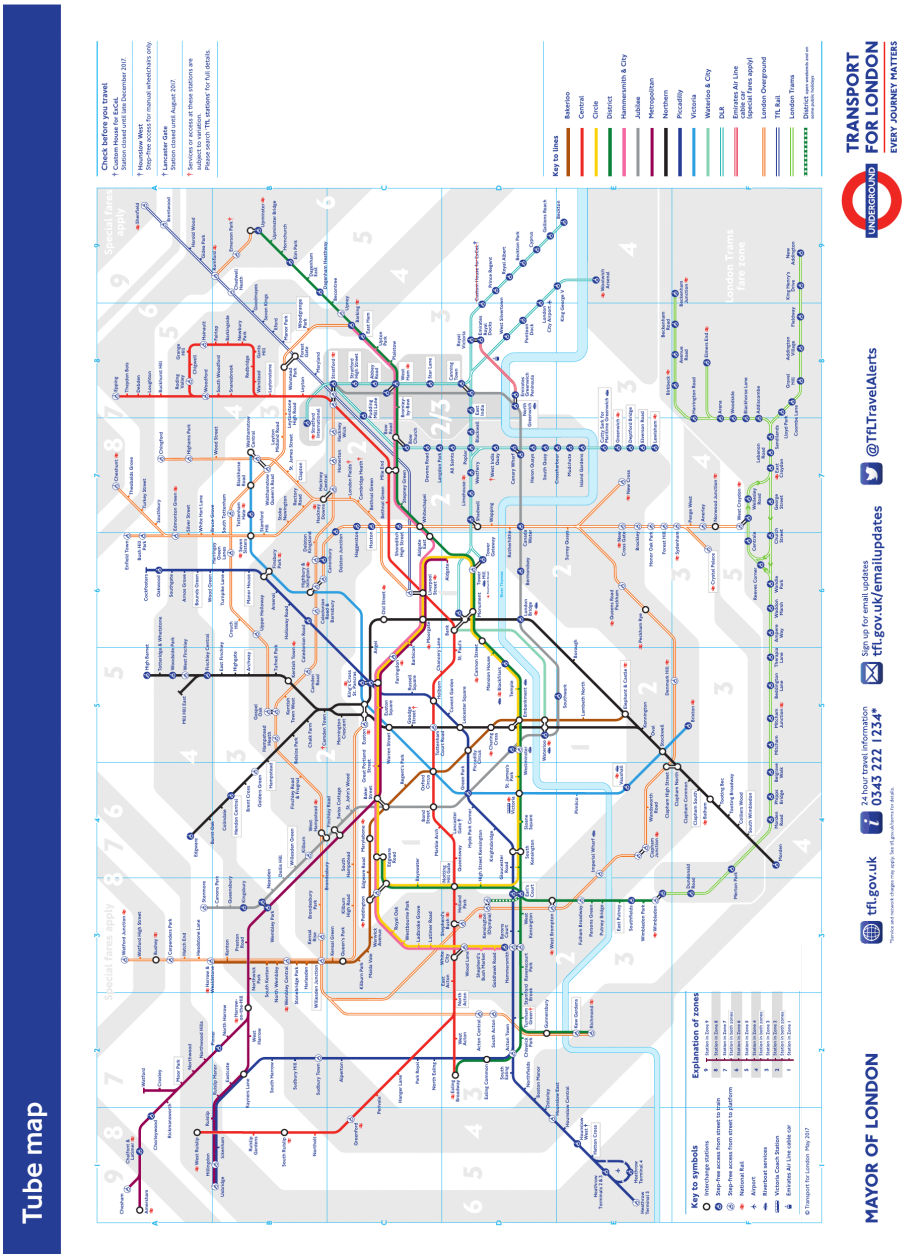


Figure 3.1: The London tube map in 2017

A path of length zero in R is a sequence a of an element $a \in A$, i.e. it connects an element to itself. If we add all such pairs to R, i.e. all pairs (a, a) where $a \in A$, we obtain what is called the reflexive closure of R — the smallest reflexive relation that contains R.

Definition 3.5.2 *The* reflexive closure *of a relation R is the union $R \cup id_A$ of R and the identity relation on A.*

Example 3.5.3 *The reflexive and transitive closure of the* inherits-from *relation between classes in Java (see Example 3.1.7) is usually called* descendant*. The descendants of a class are itself and all the descendants of the classes inherits from it.*

To define the symmetric closure of R, we need to add to R all pairs (b, a) such that $(a, b) \in R$.

Definition 3.5.4 *The* symmetric closure *of a relation R is the union $R^{-1} \cup R$.*

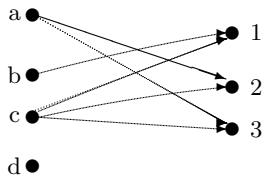
You can learn more about closure of relations in Section 9.4 of Rosen including Warshall's algorithm, which is an efficient method for computing the transitive closure of a relation.

3.6 Exercises

1. Let $A = \mathbb{S}_4$ and let $R = \{(1,2),(1,4),(2,1),(2,2),(2,4),(3,3),(4,2)\}$. Give the graphical and matrix representation of R. Write down all the elements which are related to 2 and all the elements to which 4 is related.

2. Write out all the possible relations between $\{a\}$ and $\{A, B, C\}$. How many relations are there between $\mathbb{S}_8$ and $\{a, b, c, d\}$?

3. The following is a graphical representation of a relation R. Write out all the ordered pairs in R.



4. Let $R = \{(s,t) \mid s \in \mathbb{S}_7, t \in \mathbb{S}_4, s\%3 = t\%3\}$. Give the graphical representation of R.

5. Draw the graphical representation of $\leq$ on $\mathbb{S}_6$.

6. Let R be the relation defined on $\mathbb{S}_4$ by the rule that nRm if $n \neq m$. Give the ordered pair, graphical and matrix representations of R and $\overline{R}$.

7. Let B^4 be the set of bit strings of length 4. Define a relation R on B^4 by setting $a\ R\ b$ if a and b both have the same number of 1's. Show that R is an equivalence relation, and write out the equivalence classes.

8. Let $S = \{a, b, c, d\}$ and suppose that subsets of S are represented as bit strings from B^4, so b_A represents the subset A. Let R be the relation on B^4 defined in Question 1, and define a relation R' on $\mathcal{P}(S)$ as follows:

$$R' = \{(A, B) \mid (b_A, b_B) \in R\}$$

   Show that R' is an equivalence relation on S, and describe in words the condition for two sets to be equivalent relative to R'.

9. Let $R = \{(s, t) \mid s, t \in \mathbb{S}_5, s\%3 = t\%3\}$. Write out the elements of the inverse relation R^{-1} and the complement relation $\overline{R}$.

   Give the graphical and matrix representations of R, R^{-1} and $\overline{R}$. What is the relationship between the three graphs, and what is the relationship between the three matrices?

10. Show that R^+ is transitive and that it is the smallest transitive relation on A that contains R. Show also that if R is transitive then $R^+ = R$.

11. Show that $R \cup id_A$ is reflexive and that it is the smallest reflexive relation that contains R. Show also that if R is reflexive then $R \cup id_A = R$.

12. Show that $R \cup R^{-1}$ is symmetric and that it is the smallest symmetric relation that contains R. Show also that if R is symmetric then $R \cup R^{-1} = R$.

Chapter 4

Orderings

As argued in the previous chapter, relations pay a fundamental role in Computer Science because the data that we need to represent often concerns relationships between entities or objects of the domain in which a software system needs to operate, which is why databases and spreadsheets are so commonly used. We also saw how equivalence relations give us a powerful mechanism of abstracting from one such domain the information that is relevant to represent in the system.

There is another class of relations that plays an essential role in Computer Science, this time from a more computational point of view, i.e. from the point of view of how we process data. Such relations capture orderings, which enable data to be searched in an efficient way. For example, a list of words sorted into alphabetical order is much easier to search; Google works very hard at ensuring that the results of a search are ordered in a way that the answer to our query is displayed, ideally, on the first page.

(*Most of the material in this chapter can be found in Section 9.6 of Rosen.*)

4.1 Definition and examples

Orderings are a special kind of relation on a set. One of their distinguishing properties is that they are antisymmetric:

Definition 4.1.1 *A relation R on A is* antisymmetric *if whenever both $(a, b) \in R$ and $(b, a) \in R$ then $a = b$.*

From a computational point of view, symmetric relations cause loops when searching. Requiring a relation to be antisymmetric ensures that, when searching, we always move forward, i.e. we do not go back to elements that we have already found.

Example 4.1.2 *The relation $\leq$ on $\mathbb{N}$ is antisymmetric: for any elements n, m of $\mathbb{N}$, if $n \leq m$ and $m \leq n$ then $n = m$.*

Note that being antisymmetric is not the same as not being symmetric. For example, the identity relation id_A on a set A — i.e., $(a, b) \in id_A$ if and only if $a = b$ — is both symmetric and antisymmetric. To prove that it is symmetric, let $(a, b) \in id_A$; this means that $a = b$ and, therefore, $b = a$, i.e. $(b, a) \in id_A$. Let now $(a, b), (b, a) \in id_A$; because $(a, b) \in id_A$ we have that $a = b$, and therefore id_A is antisymmetric.

An example of a relation that is neither symmetric nor antisymmetric is the relation on $\mathbb{Z}$ defined by $(x, y) \in R$ if and only if $x^2 \leq y^2$. This relation is not symmetric because, for example, $(0, 1) \in R$ but $(1, 0) \notin R$. On the other hand, the relation is not antisymmetric either because, for example, $(1, -1) \in R$ and $(-1, 1) \in R$ but $1 \neq -1$.

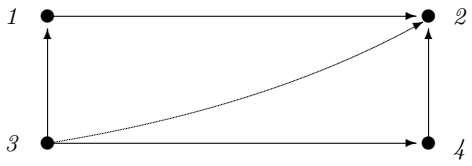
Definition 4.1.3 *A relation R on A is* a partial ordering *(or a* partial order*) if it is reflexive, antisymmetric and transitive. A set A together with a partial ordering R is called a* partially ordered set*, or* poset*, we which represent by $\langle A, R\rangle$. We tend to represent a generic partial ordering with the symbol $\preccurlyeq$.*

Example 4.1.4 *Given a set A, the set-inclusion relation $\subseteq$ makes its powerset $\mathcal{P}(A)$ a poset — you may recall from Section 2 that, given any two sets A and B, $A = B$ if and only if $A \subseteq B$ and $B \subseteq A$, which shows that the relation is antisymmetric. It is easy to show that it is also reflexive and transitive.*

A different kind of ordering is, for example, the relation $<$ on $\mathbb{N}$. Although this relation is both antisymmetric and transitive, it is obviously not reflexive and, therefore, not a partial ordering. This relation is irreflexive in the following sense:

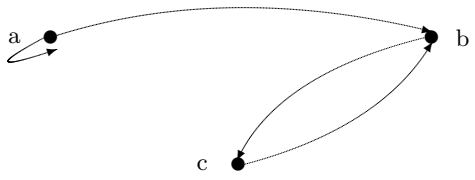
Definition 4.1.5 *A relation R on A is* irreflexive *if there is no element $a \in A$ such that $(a, a) \in R$.*

Example 4.1.6 *The relation $R = \{(n, m) \mid n\%3 < m\%3\}$ on $\mathbb{S}_4$ is irreflexive. The graphical representation is*



and there are no edges from a node to itself.

Notice that it is possible for a relation to be neither reflexive nor irreflexive. For example, the relation R on $\{a, b, c\}$ whose graph is shown below is not reflexive because $(c, c) \notin R$; it is not irreflexive either because $(a, a) \in R$. Therefore, if a relation is not reflexive, we cannot conclude that it is irreflexive (or the other way around).



Definition 4.1.7 *A* strict *partial ordering on a set A is a relation R on A that is irreflexive and transitive. We tend to represent a generic strict partial ordering with the symbol $\prec$.*

Notice that every strict partial ordering is antisymmetric: suppose that, for some $a \neq b$, $a \prec b$ and $b \prec a$; by transitivity, we would get that $a \prec a$, which would contradict the fact that $\prec$ is irreflexive.

Every partial order $\preccurlyeq$ on a set A defines the strict partial order $\prec$ on A where, for every $a, b \in A$, $a \prec b$ if and only if $a \preccurlyeq b$ and $a \neq b$. Likewise, every strict partial order $\prec$ on a set A defines the partial order $\preccurlyeq$ on A where, for every $a, b \in A$, $a \preccurlyeq b$ if and only if $a \prec b$ or $a = b$. Therefore, when working with a partial ordering $\preccurlyeq$, we will denote by $\prec$ the corresponding strict partial ordering, and the other way around.

Definition 4.1.8 *A* total *(or* linear*) order on a set A is a partial ordering such that every two elements $a, b \in A$ are* comparable*, i.e. either $a \preccurlyeq b$ or $b \preccurlyeq a$. In this case, the poset is also called a* chain*.*

Example 4.1.9 *For example, the relation $\leq$ is a total ordering on $\mathbb{Z}$ but $\subseteq$ is not a total ordering on $\mathcal{P}(\{0, 1\})$ because, for example, the subsets $\{0\}$ and $\{1\}$ are not comparable. (Can you think of a set A such that $\mathcal{P}(A)$ is a chain?)*

A typical example of the use of posets in Computer Science concerns distributed systems.

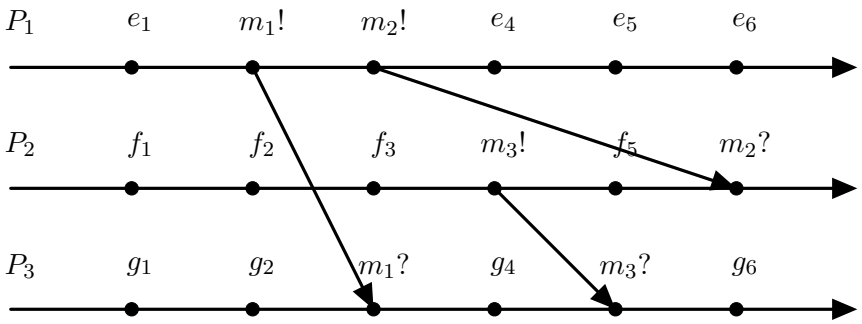
Example 4.1.10 *A distributed program P consists of a finite set $\{P_1, \ldots, P_n\}$ of processes running on different processors and communicating via messages. Each process P_i generates a sequence of events, which gives rise to a chain $\langle E_i, \rightarrow_i \rangle$ where E_i is the set of events that P_i executes (including sending and receiving messages) and, given two events $e, f \in E_i$, $e \rightarrow_i f$ if the event e precedes f in that sequence.*

The fact that the processes communicate via messages requires those orderings to be extended to the distributed programme as a whole. This can be obtained by taking the transitive closure of the following set of pairs:

- *All pairs (e, f) such that $e \rightarrow_i f$ for some process P_i*
- *For every message m, the pair $(m!, m?)$ where $m!$ is the event of sending m and $m?$ is the event of receiving m.*

We denote the resulting partial order on P by $\rightarrow$.

For example, in the example below, we have $e_1 \rightarrow_1 m_1!$ (and therefore $e_1 \rightarrow m_1!$), $m_1? \rightarrow_3 g_4$ (and therefore $m_1? \rightarrow g_4$), and $m_1! \rightarrow m_1?$. By transitivity it therefore results that $e_1 \rightarrow g_4$.



Definition 4.1.11 *Given a poset $\langle A, \preccurlyeq \rangle$, we say that $B \subseteq A$ is:*

- *a* lower set *if, for each $b \in B$, for all $a \in A$ such that $a \preccurlyeq b$, then $a \in B$.*
- *an* upper set *if, for each $b \in B$, for all $a \in A$ such that $b \preccurlyeq a$, then $a \in B$.*

That is, a lower (resp. upper) set is one that contains all elements that are less (greater) than any of its elements.

This is useful in relation to Example 4.1.10 as it allows us to define the notion of *global consistent state* of a distributed programme: any lower set of events, i.e., any set of events that contains all the events that precede them. For example, a global consistent state cannot contain the reception of a message that has not been sent.

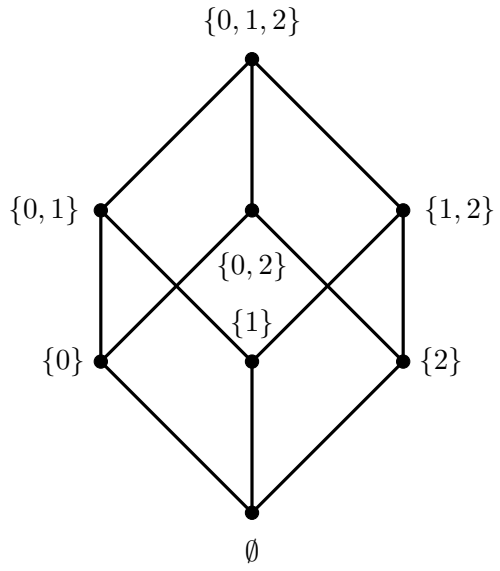
4.2 Representation of orderings

Representing posets using graphs as, for example, $\leq$ on $\mathbb{S}_5$ as depicted in 3.2.4, can be somewhat redundant in the sense that we already know that some of the edges need to be there: because the relation is reflexive, we know that there need to be loops at all vertices so we might not represent them; and because the relation is transitive, we know that all paths need to be represented, so we might represent only those that cannot be inferred. It we further assume that all directed edges point "upwards" and represent them as such, we omit the arrows.

If we make all those simplifications to the graphical representation of a poset, we obtain what is called the *Hasse diagram* of that poset.

Example 4.2.1 *The Hasse diagram of the poset $\langle \mathbb{S}_5, \leq \rangle$ depicted in 3.2.4 is:*

Example 4.2.2 *The Hasse diagram of the poset defined by* $\subseteq$ *on* $\mathcal{P}(\{0,1,2\})$ *is:*



4.3 Lexicographic orderings

The usual alphabetic ordering that you normally see in a list of words (say, a list of names or a dictionary) is a total ordering on the underlying set of words that is based on the ordering of the letters in the alphabet: $a \prec b \prec c \prec d \prec e$, etc. That ordering on words is an example of a class of orderings called lexicographic orderings.

Another example is the ordering on pairs of integers obtained by comparing first on the left element and then on the right element. For example, we would have $(1,7) \leq (2,4)$ and $(4,5) \leq (4,7)$, etc. The general rule that defines this ordering is

$$(n,m) < (s,r) \text{ if either } (n < s) \text{ or } (n = s \text{ and } m < r).$$

More generally, if we have a strict total ordering $\prec$ on a set A, we can use it to define an ordering on the cartesian product A^2 using the rule

$$(a, b) \prec (c, d) \text{ if } a \prec c \text{ or } (a = c \text{ and } b \prec d).$$

This is called the *lexicographic ordering generated by* $\prec$, or the *lexicographic extension of* $\prec$.

This construction can be extended to an arbitrary cartesian product A^n where $n > 0$ (see Section 2.1.6).

Definition 4.3.1 *Given a total ordering $\prec$ on a set A, we define its lexicographic extension on A^n for any n as follows: given any strings $(a_1, a_2, \ldots, a_n)$ and $(b_1, b_2, \ldots, b_n)$,*

$(a_1, \ldots, a_n) \prec (b_1, \ldots, b_n)$ *if and only if*
either $a_1 \prec b_1$, *or*
there exists $0 < i < n$ *such that* $a_1 = b_1, \ldots, a_i = b_i$, *and* $a_{i+1} \prec b_{i+1}$

In other words, one n-string is less than a second n-string if the entry of the first n-string in the first position where the two n-strings disagree is less than the entry in that position in the second n-string. An algorithm to check if one n-string is less than a second n-string could compare the first (left most) elements of the two strings; if these are the same then compare their second elements; if these are also the same then compare their third elements, and so on.

Example 4.3.2 *Suppose that $B = \{0, 1\}$ and $\leq$ is defined on B in the usual way so that $0 < 1$. The lexicographic extension of $\leq$ to B^4 includes the following pairs, among others:*

(0,0,0,0)	$<$	*(1,0,0,0)*	*(0,0,0,0)*	$<$	*(0,1,0,1)*
(0,0,0,0)	$<$	*(0,0,1,0)*	*(0,1,0,0)*	$<$	*(0,1,1,0)*
(1,0,0,0)	$<$	*(1,1,0,0)*	*(1,0,0,1)*	$<$	*(1,1,0,0)*

The examples that we gave at the start – a list of names or a dictionary – do not fit exactly any of the definitions above because names and words in a dictionary do not necessarily have the same length. Therefore, we now extend the lexicographic order to strings of arbitrary positive length:

Definition 4.3.3 *The lexicographic extension of a total ordering on a set A to A^+ (see Section 2.1.6) is as follows: consider two strings $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_n)$ and let r be the minimum of m and n (this is so that we can compare words of the same length); we can now define:*

$(a_1, \ldots, a_m) \prec (b_1, \ldots, b_n)$ *if and only if*
$(a_1, \ldots, a_r) \prec (b_1, \ldots, b_r)$, *or*
$(a_1, \ldots, a_r) = (b_1, \ldots, b_r)$ *and* $r = m$ *(i.e.* $m < n$*)*

That is to say, we truncate the longer string to the length of the shorter one, and then we compare the resulting words using the lexicographic order over words of the same length

bearing in mind that, if those two words are equal, the shortest string will be less than the longest.

For example, we have *communication* $\prec$ *computer*: we truncate *communication* to the length of *computer*, which gives *communic* $\prec$ *computer*. We also have *computer* $\prec$ *computers*: we truncate *computers* to the length of *computer*, which gives *computer* $=$ *computer*.

4.4 Well-founded posets

Some elements of posets have certain properties that are important for, example to know where search should start or when to stop.

Definition 4.4.1 *Let $\preccurlyeq$ be a partial ordering on a set A and $a \in A$.*

- *We say that a is* maximal *if there is no element $b \in A$ such that $a \prec b$.*
- *We say that a is* minimal *if there is no element $b \in A$ such that $b \prec a$.*

Maximal and minimal elements are easy to spot using a Hasse diagram: they are at the "top" and "bottom" of the diagram.

Note that a given poset may have none or multiple maximal/minimal elements.

Example 4.4.2 *The posets in examples 4.2.1 and 4.2.2 have both maximal and minimal elements: in the first case, 1 is minimal and 5 is maximal; in the second case, $\emptyset$ is minimal and $\{0, 1, 2\}$ is maximal.*

Theorem 4.4.3 *Every finite nonempty subset B of a poset $\langle A, \preccurlyeq \rangle$ has minimal elements and maximal elements.*

Proof *Suppose that $B = \{b_0, \ldots, b_{n-1}\}$ for some $n \geq 1$. Let $m_0 = b_0$ and define*

$$m_k = \begin{cases} b_k & \text{if } \mathrm{b_k < m_{k-1}} \\ m_{k-1} & \text{otherwise} \end{cases}$$

It is clear from the definition that, for all k, $m_k \leq m_{k-1}$ and $m_k \leq b_k$, and therefore $m_k \leq b_i$ for all $i = 0, \ldots, k$. Therefore, m_{n-1} is a minimal element as $m_{n-1} \leq b_i$ for all $i = 0, \ldots, n-1$. The proof of the existence of a maximal element is similar.

We can therefore conclude:

Corollary 4.4.4 *If A is a finite non-empty set then any poset $\langle A, \preccurlyeq \rangle$ has minimal and maximal elements.*

The maximal and minimal elements of Example 4.4.2 have additional properties: the minimal elements are such that all other elements are greater than them, and the maximal elements are such that all other elements are less than them.

Definition 4.4.5 *Let $\preccurlyeq$ be a partial ordering on a set A and $a \in A$.*

- *We say that a is* the greatest element *if, for all $b \in A$, $b \preccurlyeq a$.*
- *We say that a is* the least element *if, for all $b \in A$, $a \preccurlyeq b$.*

Notice that if $\preccurlyeq$ is a total order then any minimal (resp. maximal) element is also the least (resp. greatest) element.

Example 4.4.6 *If A is a set, $\langle \mathcal{P}(A), \subseteq \rangle$ has $\emptyset$ as its least element and A itself as its greatest element.*

Example 4.4.7 *In Java, inheritance hierarchies have a greatest element called* Object*, defined in the java.lang package, which defines and implements behaviour that is common to all classes, including the ones that programmers write. In the Java platform, many classes derive directly from Object, other classes derive from some of those classes, and so on, forming a hierarchy of classes with* Object *as the greatest class (the one all classes inherit from).*

Definition 4.4.8 *We say that a partial ordering $\preccurlyeq$ on a set A is* well-founded *— and that the poset $\langle A, \preccurlyeq \rangle$ is well-founded — if and only if every non-empty subset $B \subseteq A$ has a minimal element.*

If $\langle A, \preccurlyeq \rangle$ is a well-founded chain, then we also say that it is well-ordered*.*

Essentially, being well-founded ensures that there is no infinite descending chain: that is, there is no infinite sequence $a_0 \succ a_1 \succ a_2 \succ \dots$. The difference between working with a general poset and working with a chain is that, when building a descending chain over a general poset there may be alternative branches to explore whereas, over a chain, there is only one way down.

Example 4.4.9 *The chain $\langle \mathbb{N}, \leq \rangle$ is well-ordered.*

Example 4.4.10 *The chain $\langle \mathbb{Z}, \leq \rangle$ is not well-ordered because it does not have a least element.*

The following property follows from Theorem 4.4.3 and reflects the obvious fact that one cannot build infinite descending chains over a finite poset:

Corollary 4.4.11 *Every finite poset is well-founded and every finite chain is well-ordered.*

The following result is also useful.

Theorem 4.4.12 *The lexicographic extension of a well-ordered chain is also well-ordered.*

This theorem has useful applications in computer science as illustrated by the following example, which concerns termination of computations:

Example 4.4.13 *We are given a bag containing red, yellow, and blue counters. Over that bag, we execute the following procedure:*

- *If one chip remains in the bag, we remove it and terminate.*
- *If there are two or more counters in the bag, we remove two at random and:*
  1. *If one of the two is red, we keep them and execute the procedure again.*
  2. *If both are yellow, we put one yellow and five blue counters in the bag, and execute the procedure again.*
  3. *If one of the two is blue and the other is not red, we put ten red counters in the bag and execute the procedure again.*

Does this process terminate?

Solution
Let us first model the state of the program, which consists of the contents of the bag. We represent states as triples (y, b, r) where $y, b, r \in \mathbb{N}$ are the number of yellow, blue and red counters in the bag, respectively. Given a state (y_1, b_1, r_1), one step of the execution will produce a new state (y_2, b_2, r_2).

Consider the lexicographic extension of $<$ to $\mathbb{N}^3$, i.e. to the triples of natural numbers, and let us compare (y_1, b_1, r_1) and (y_2, b_2, r_2) according to the rules. Suppose we are in a state (y, b, r) such that $y + b + r > 1$ and we remove two counters at random:

1. The possibilities are:
   - the two counters are red, so we move to state $(y, b, r-2)$, which satifies $(y, b, r-2) < (y, b, r)$
   - one of the counters is red and the other is yellow, so we move to $(y-1, b, r-1)$, which satisfies $(y-1, b, r-1) < (y, b, r)$
   - one of the counters is red and the other is blue, so we move to the state $(y, b-1, r-1)$, which satisfies $(y, b-1, r-1) < (y, b, r)$
2. In this case, both the counters are yellow and we move to the state $(y-1, b+5, r)$, which satisfies $(y-1, b+5, r) < (y, b, r)$
3. The possibilities are:

- both counters are blue, so we move to the state $(y, b-2, r+10)$, which satisfies $(y, b-2, r+10) < (y, b, r)$
- one of the counters is blue and the other is yellow, so we move to $(y-1, b-1, r+10)$, which satisfies $(y-1, b-1, r+10) < (y, b, r)$

In summary, in all cases, given a state (y_1, b_1, r_1), one step of the execution will produce a new state $(y_2, b_2, r_2) < (y_1, b_1, r_1)$. Because $<$ over $\mathbb{N}$ is well-founded, according to Theorem 4.4.12 so is its lexicographic extension to $\mathbb{N}^3$. Therefore, there are no infinite descending chains in $\mathbb{N}^3$, which means that the procedure will necessarily terminate.

4.5 Lattices

We have already seen that, given a set A, set inclusion defines the poset $\langle \mathcal{P}(A), \subseteq \rangle$, of which $\emptyset$ is the least element and A itself is the greatest element. In this section, we study generalisations of two operations on sets that we studied in Chapter 2: set intersection $\cap$ and set union $\cup$.

Let us consider the following case:

Example 4.5.1 *The chain $\langle \mathbb{R}_0^+, \leq \rangle$ of the non-negative real numbers is not well-ordered: for example, the subset $\{1/2^n \mid n \in \mathbb{N}\}$ is not empty and does not have a least element.*

This case is interesting in the sense that, although $\{1/2^n \mid n \in \mathbb{N}\}$ does not have a least element, 0 seems to be a kind of limit: had we included it in the set, it would have been a minimal element (in fact, a least element).

Definition 4.5.2 *Let $\preccurlyeq$ be a partial ordering on a set A and $B \subseteq A$.*

- *We say that $a \in A$ is an* upper bound *of B if, for all $b \in B$, $b \preccurlyeq a$. We say that a is the* least *upper bound of B – denoted $lub(B)$ – if a is an upper bound that is less than every other upper bound of A.*
- *We say that a is a* lower bound *of B if, for all $b \in B$, $a \preccurlyeq b$. We say that a is the* greatest *lower bound of B – denoted $glb(B)$ – if a is a lower bound that is greater than every other lower bound of A.*

Example 4.5.3 *Consider the chain $\langle \mathbb{R}_0^+, \leq \rangle$. The greatest lower bound of the subset $\{1/2^n \mid n \in \mathbb{N}\}$ is 0. However, if we consider instead the chain $\langle \mathbb{R}^+, \leq \rangle$ of the positive real numbers, $\{1/2^n \mid n \in \mathbb{N}\}$ does not have any lower bound.*

The existence of upper or lower bounds is important as a structural feature of posets:

Definition 4.5.4 *A* lattice *is a poset $\langle A, \preccurlyeq \rangle$ that satisfies the following two properties:*

join: *every pair of elements* $a_1, a_2 \in A$ *has a least upper bound, usually called their* join *and denoted* $a_1 \vee a_2$*; and*

meet: *every pair of elements* $a_1, a_2 \in A$ *has a greatest lower bound, usually called their* meet *and denoted* $a_1 \wedge a_2$*.*

If the poset satisfies the first property, it is called a join semi-lattice*. If it satisfies the second, it is called a* meet semi-lattice*.*

The join and meet operators satisfy some of the properties of union and intersection of sets discussed in Section 2.2.3:

Theorem 4.5.5 *Given a lattice* $\langle A, \preccurlyeq \rangle$ *and elements* $a, b, c \in A$*, we have that:*

commutative: $a \wedge b = b \wedge a$ *and* $a \vee b = b \vee a$

associative: $(a \wedge b) \wedge c = a \wedge (b \wedge c)$ *and* $(a \vee b) \vee c = a \vee (b \vee c)$

absorption: $a \wedge (a \vee b) = a$ *and* $a \vee (a \wedge b) = a$

idempotent: $a \wedge a = a$ *and* $a \vee a = a$

We leave the proof of this theorem as an exercise.

There are many applications of lattices in Computer Science, including in breaking cryptosystems and constructing secure cryptographic algorithms. Another example are so-called *concept lattices*, which are the basis of *Formal Concept Analysis*, an area that has many applications in programming and software engineering, including refactoring and software product lines.

Definition 4.5.6 *A* formal context *is a triple* (G, M, I) *where* I *is a relation (called* incidence*) between a set* G *(of objects) and a set* M *(of attributes).*

Example 3.2.7 is such a formal context: text editors are the objects and the features are the attributes.

Given a set of objects $A \subseteq G$, we can define the set A' of all the attributes that are common to the objects in A, i.e.

$$A' = \{m \in M \mid \text{for all } g \in A,\ gIm\}$$

Likewise, given a set of attributes $B \subseteq M$, we can define the set B' of all the objects that have all the attributes in B, i.e.

$$B' = \{g \in G \mid \text{for all } m \in B,\ gIm\}$$

For example,

$$\{e_2, e_4\}' = \{f_1, f_2, f_3, f_4, f_7, f_8\}$$
$$\{f_1, f_2, f_3, f_4, f_7, f_8\}' = \{e_2, e_4\}$$

Definition 4.5.7 *A* formal concept *over a formal context* (G, M, I) *is a pair* (A, B) *with* $A \subseteq G$ *and* $B \subseteq M$ *such that* $A' = B$ *and* $B' = A$*. We call* A *the* extent *of the concept and* B *its* intent*.*

Therefore, the pair $(\{e_2, e_4\}, \{f_1, f_2, f_3, f_4, f_7, f_8\})$ is a formal concept of the formal context defined in Example 3.2.7. Whenever a change needs to be made to one feature, it is important for the company to know which editors use this feature. When the search space is very big, concepts provide a strategy for reducing the search size because they can be organised as a lattice.

Theorem 4.5.8 *Given a formal context* (G, M, I)*, the relation on its formal concepts defined by* $(A_1, B_1) \preccurlyeq (A_2, B_2)$ *if* $A_1 \subseteq A_2$ *is a partial ordering that makes* (G, M, I) *a lattice.*

Notice that $A_1 \subseteq A_2$ is equivalent to $B_2 \subseteq B_1$ (you can try to prove it as an exercise). The partial ordering on concepts is therefore $\subseteq$ on extents and $\supseteq$ on intents.

Returning to our example, there are four concepts in total:

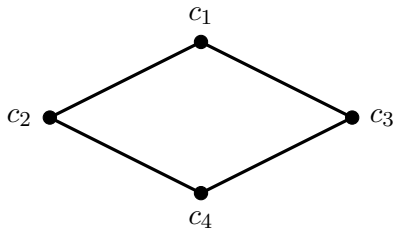
c_1**:** $(\{e_1, e_2, e_3, e_4\}, \{f_1, f_2, f_3, f_4\})$

c_2**:** $(\{e_2, e_4\}, \{f_1, f_2, f_3, f_4, f_7, f_8\})$

c_3**:** $(\{e_3, e_4\}, \{f_1, f_2, f_3, f_4, f_5, f_6\})$

c_4**:** $(\{e_4\}, \{f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8\})$

These concepts are ordered as follows: $c_4 \prec c_2$, $c_4 \prec c_3$, $c_2 \prec c_1$ and $c_3 \prec c_1$. Concept c_1 (the greatest concept) is the common feature set – often called common block – and c_2, c_3 and c_4 are optional feature sets – blocks of variations; concept c_4 (the least concept) is the software product that has all features. This is useful for identifying which features are common to all products, and which variations need to be separately maintained.



Building the concept lattice that corresponds to a formal context is far from trivial but there are software applications that implement that construction, which have been developed precisely to support applications in software engineering.

4.6 Exercises

1. Show that the relation $\leq$ on $\mathbb{N}$ is a partial ordering.

2. Prove that, for the lexicographic ordering over $\{0,1\}^4$ defined in Example 4.3.2, $(b_1, b_2, 0, b_4) < (1, 1, 1, 1)$ for every b_1, b_2 and b_4.

3. Show that any greatest (resp. least) element of a poset is a maximal (resp. minimal) element and is unique.

4. Given a non-empty set A, consider the prefix relation on the set A^+ of the non-empty finite strings of elements of A, i.e. given any $s, t \in A^+$, $s \prec t$ if there exists a sequence r such that t is the concatenation of s and r, i.e. if s is a prefix of t.

   (a) Show that A^+ has no maximal elements. Show that if A has two elements or more, A^+ has minimal elements but no least element.

   (b) Extend the prefix relation on A^+ to A^* (see Section 2.1.6) by adding all pairs $\epsilon \prec s$ where $s \in A^+$. Show that A^* has a least element.

5. Show that $\langle \mathcal{P}(A), \subseteq \rangle$ is a lattice, and that the meet (resp. join) of any two subsets of A is their intersection (resp. union).

6. Given a non-empty set A, show that A^* (the set of all finite strings on A, including the empty string) with its prefix order is a meet semi-lattice; what does the meet operator do to two strings? Give an example of a set A such that A^* is not a join semi-lattice; can you think of a class of alphabets such that A^* is also a join semi-lattice (and therefore a lattice)?

7. Show that not all lattices satisfy the distributive laws:
   $(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$ and $(a \vee b) \wedge c = (a \wedge c) \vee (b \wedge c)$.

Chapter 5

Functions and Cardinality

Most of the material in this chapter can be found in Sections 2.3 and 2.5 of Rosen.

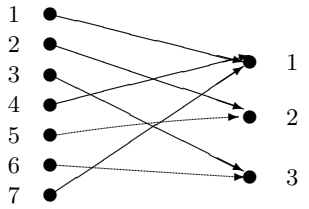
5.1 Functions

Functions (sometimes call maps) are another particular case of a relation between two sets, the domain and the codomain. The distinguish property of a function is that every element in the domain is related to one and exactly one element in the codomain.

For example, consider the relation $R \subseteq \mathbb{S}_7 \times \mathbb{S}_3$ defined by nRm if and only if $n \equiv m$ modulo 3, i.e.

$$R = \{(1,1),(2,2),(3,3),(4,1),(5,2),(6,3),(7,1)\}.$$

Each element of $\mathbb{S}_7$ appears in one and only one pair $(n,m) \in R$. Looking at its graphical representation, we can see that there is one and only one arrow *from* each element in $\mathbb{S}_7$.



Functions arise naturally in computer science. For example, many programs implement functions. The domain of a function implemented by a program is the set of inputs on which the program terminates, and the output of the program on every given input defines the pairs *(input,output)* of the relation. Another example are spreadsheets: if rows correspond to the domain (say, names of people), every column for which every cell is filled defines a function (say, age or date of birth); if for a given row some of the cells are not filled (say,

address) then it does not define a function (say, because some people have not provided an address).

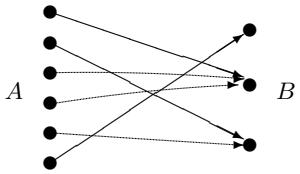
When a relation is a function we use functional notation rather than ordered pairs. So we write $f : A \to B$ instead of $f \subseteq A \times B$, and $f(a) = b$ rather than $(a, b) \in f$ or afb.

The following is the definition of function written in functional notation.

Definition 5.1.1 *A relation $f : A \to B$ is a function between A and B if*

1. *for every $a \in A$ there is some $b \in B$ such that $f(a) = b$,*
2. *for every $a \in A$, if $f(a) = b_1$ and $f(a) = b_2$ then $b_1 = b_2$.*

So for each $a \in A$ there is only one $b \in B$ such that $(a, b) \in f$. Graphically, there is exactly one arrow from each element of A.



Many of the familiar 'functions' from calculus are functions from $\mathbb{R} \to \mathbb{R}$. For example, $f(x) = x^2$ and $f(x) = cos(x)$ are all functions from $\mathbb{R}$ to $\mathbb{R}$.

A bit of care is needed here though. The map $f(x) = x^{-1}$ is *not* a function from $\mathbb{R}$ to $\mathbb{R}$ because it is not defined on 0.

Exercise 5.1.2 *Find a set A such that $f(x) = x^{-1}$ is a function from A to $\mathbb{Z}$.*

Definition 5.1.3 *Given a function $f : A \to B$*

- *We call A the* domain *of f and B the codomain of f.*
- *For every $a \in A$, we call $f(a)$ the* image *of a under f.*
- *We write $f(A) = \{f(a) \mid a \in A\}$, which we call the image of A in B.*

If a function is implemented by a program, the image $f(a)$ is the output produced by the program on the input a.

Example 5.1.4 *Let $A = \mathbb{S}_4$ and let $B = \mathbb{S}_5$. Decide which of the following relations are functions from A to B. For those which are functions, write down their domain and image.*

- $R_1 = \{(n, m) \mid n \in \mathbb{S}_4, m \in \mathbb{S}_5, m = n + 1\}$
- $R_2 = \{(n, m) \mid n \in \mathbb{S}_4, m \in \mathbb{S}_5, m\%3 = n\%3\}$

- $R_3 = \{(1,1),(2,1),(3,1),(4,1)\}$
- $R_4 = \{(1,1),(2,1)\}$

Solution

R_1**:** We start by checking the two conditions that need to be satisfied by a function.

1. For each $n \in \mathbb{S}_4$, we have $(n+1) \in \mathbb{S}_5$ and $(n, n+1) \in R_1$. So there is at least one $m \in \mathbb{S}_5$ such that $(n,m) \in R_1$.
2. If $(n,m),(n,r) \in R_1$ then $m = n+1 = r$, so $m = r$. Thus there is exactly one $m \in \mathbb{S}_5$ such that $(n,m) \in R_1$.

Thus R_1 is a function. Call this function f_1, so that $f_1(n) = n+1$. The domain of f_1 is $\mathbb{S}_4$, and the image of f_1 is $\{f_1(n) \mid n \in \mathbb{S}_4\} = \{2,3,4,5\}$.

R_2**:** $1\%3 = 1\%3$ and $1\%3 = 4\%3$ so $(1,1),(1,4) \in R_2$. Therefore R_2 is not a function.

R_3**:** We start by checking the two conditions that need to be satisfied by a function.

1. If $n \in \mathbb{S}_4$ then $(n,1) \in R_3$. So there is at least one $m \in \mathbb{S}_5$ such that $(n,m) \in R_3$.
2. Since all the elements of R_3 are of the form $(n,1)$, there is only one pair in the relation for each element of the domain.

Thus R_3 is a function. Call this function f_3, so that $f_3(n) = 1$. The domain of f_3 is $\mathbb{S}_4$ and the image of f_1 is $\{1\}$.

R_4**:** There is no pair containing 4 in R_4, i.e. there is no m such that $(4,m) \in R_4$. Therefore R_4 is not a function.

Definition 5.1.5 *Two functions f, g are equal if and only if they have the same domain, the same codomain and, for all x in the domain, $f(x) = g(x)$.*

For example, the functions $f : \mathbb{Z} \to \mathbb{Z}$ and $g : \mathbb{Z} \to \mathbb{Z}$ given by $f(x) = 2x$ and $g(x) = x + x$ are equal.

Note that the function $h : \mathbb{N} \to \mathbb{Z}$ given by $h(x) = 2x$ is not equal to f and g because it has a different domain: $\mathbb{N}$ instead of $\mathbb{Z}$.

5.2 Inverse functions

As a relation, every function has an inverse (see Section 3.3). However, it is not always the case that the inverse of a function is a function.

For example, the inverse of the function $f : \mathbb{Z} \to \mathbb{N}$ given by $f(n) = n^2$ is $R = \{(m, n) \mid m \in \mathbb{N}, n \in \mathbb{Z}, m = n^2\}$. This relation is not a function because, for example, $(1, 1), (1, -1) \in R$.

The inverse of the function $f : \mathbb{N} \to \mathbb{N}$ given by $f(n) = n^2$ is $R = \{(m, n) \mid m \in \mathbb{N}, n \in \mathbb{N}, m = n^2\}$. This relation is not a function either because, for example, there is no $n \in \mathbb{N}$ such that $(3, n) \in R$.

Definition 5.2.1 *Functions whose inverse relation is also a function are called* invertible. *We write $f^{-1} : B \to A$ for the inverse of $f : A \to B$.*

Invertible functions are important because they can be 'undone'. We have $f^{-1}(f(x)) = x$ for all $x \in A$ and $f(f^{-1}(y)) = y$ for all $y \in B$.

A practical example is given next.

5.2.1 Error detection in bit strings

When data is transmitted it is possible that errors can be introduced. In its simplest form, we can imagine transmitting bit strings of length 8; occasionally, one or more of the bits will become corrupted. For example, $(1, 0, 0, 1, 1, 0, 0, 1)$ is sent and $(1, 0, 1, 1, 1, 0, 0, 1)$ is received.

How can the receiver know that there has been an error, and ask for the data to be resent? The theory of error correcting codes is well advanced, and is based on vector spaces. We just discuss a simple method that will tell the receiver if there has been one error in the transmission.

The method consists of sending a string of length 9, instead of a string of length 8, where the last bit on the string is the sum of the previous 8 bits modulo 2, i.e., the sum in binary arithmetic. That is, if we wish to send $(b_1, b_2, \ldots, b_8)$ we actually send $(b_1, b_2, \ldots, b_8, b_9)$ where

$$b_9 = \begin{cases} 0 & \text{if } (\mathrm{b}_1 + \mathrm{b}_2 + \ldots + \mathrm{b}_8) \text{ iseven} \\ 1 & \text{if } (\mathrm{b}_1 + \mathrm{b}_2 + \ldots + \mathrm{b}_8) \text{ isodd} \end{cases}$$

Thus we send $(1, 0, 0, 1, 1, 0, 0, 1, 0)$ instead of $(1, 0, 0, 1, 1, 0, 0, 1)$.

The receiver then checks that the last digit is correct by computing the sum of the first 8 bits received. If it is not then an error has been introduced and a request is made that the string is sent again.

This method will detect that there has been an odd number of errors, but not if there have been an even number errors, which requires more sophisticated methods. For example, if we receive $(1, 0, 1, 1, 1, 0, 0, 1, 0)$ then we know there has been an error, but if we receive $(1, 0, 1, 0, 1, 0, 0, 1, 0)$ then we can't tell that there are errors.

What has this to do with invertible functions? We need a function to convert the message into a coded one, and an inverse function to decode the message at the other end.

For the error detecting method above we have a function $f : B^8 \to B^9$, where $B = \{0, 1\}$ defined by $f(b_1, b_2, \ldots, b_8) = (b_1, b_2, \ldots, b_8, b_9)$ where b_9 is as defined above, which is used by the sender. To decode the message, the receiver applies f^{-1} defined by $f^{-1}(b_1, b_2, \ldots, b_8, b_9) = (b_1, b_2, \ldots, b_8)$.

See Section 4.5 of Rosen for many other examples of error detection using more sophisticated functions.

5.3 Injective and surjective functions

For the relation R_3 defined in Example 5.1.4, we have that $f_3(1) = 1 = f_3(2)$. Therefore, knowing $f_3(n)$ is not enough for us to be able to tell what n is.

The relation R_1 defined in the same example has the property that if $f_1(n) = f_1(m)$ then $n = m$. In this case, if we know $f_1(n)$ then we can work out n.

Definition 5.3.1 *A function $f : A \to B$ is* injective *if, for all $a_1, a_2 \in A$, $f(a_1) = f(a_2)$ implies that $a_1 = a_2$. We also say that f is* one-to-one*, or* $1 - 1$.

Graphically, no element in B has two arrows going to it. That is, no two inputs can have the same output.

As another example, all the 'linear' functions $f : \mathbb{Z} \to \mathbb{Z}$ are injective, that is functions of the form $f(x) = ax + b$ where $a, b \in \mathbb{Z}$ and $a \neq 0$.

The definition tells us how to prove or disprove the fact that a given function is injective. For example, in the case above, we need to pick two arbitrary $n, m \in \mathbb{Z}$ and show that $f(n) = f(m)$ implies $n = m$. Suppose then that $f(n) = f(m)$. From the definition of f, this means that $an + b = am + b$, which in turn implies $an = am$. Because $a \neq 0$, we get $n = m$, which shows that f is injective.

Coming back to what 'arbitrary' means (recall Section 2.4.5), notice that choosing a or b in place of n or m would be wrong because a and b are constants used in the formulation of the function. Therefore the choice would no longer be arbitrary.

Also notice that the question 'is $f(x) = x^2 + 2$ an injective function?' is not well formulated because the domain and codomain are missing. For example, $f(x) = x^2 + 2$ is injective on $\mathbb{N}$ (prove it) but not on $\mathbb{Z}$. To show that it is not injective on $\mathbb{Z}$, all we need to do is find a counterexample (see Section 2.4.4), i.e. two inputs that produce the same output. For example, $f(1) = 1^2 + 2 = 1 + 2 = (-1)^2 + 2 = f(-1)$.

Coming back to R_3 above, we notice that there is no element $n \in \mathbb{S}_4$ such that $f_3(n) = 2$ while $2 \in \mathbb{S}_4$, i.e. there is an element of the codomain that is not an output.

Definition 5.3.2 *A function $f : A \to B$ is* surjective *if for every $b \in B$ there exists some $a \in A$ such that $f(a) = b$. We also say that f is* onto.

Graphically, a function is surjective if every node in B has some arrow going to it.

For example, neither of the functions R_1, R_3 from $\mathbb{S}_4$ to $\mathbb{S}_5$ is surjective. In fact there are no surjective functions $f : \mathbb{S}_4 \to \mathbb{S}_5$. Similarly, there are no injective functions $f : \mathbb{S}_5 \to \mathbb{S}_4$. Both of these facts are consequences of Theorem 5.3.6, which is given below.

Coming back to the linear functions $f : \mathbb{Z} \to \mathbb{Z}$ defined by $f(x) = ax + b$ for any $a, b \in \mathbb{Z}$ with $a \neq 0$, it depends on the choice of a and b whether they are surjective or not. For example, $f(x) = x$ (i.e. $a = 1$ and $b = 0$) is clearly surjective because f is the identity on $\mathbb{Z}$ (each output equals the input). However, $f(x) = 2x$ (i.e. $a = 2$ and $b = 0$) is clearly not surjective because its outputs are even.

On the other hand, if we change the domain and codomain to $\mathbb{R}$ instead of $\mathbb{Z}$ then $f(x) = ax + b$ is surjective. To prove it, choose an arbitrary $y \in \mathbb{R}$; to find an input that produces y as an output we need to solve the equation $y = ax + b$, which has the solution $x = (y - b)/a$ because $a \neq 0$ and $(y - b)/a \in \mathbb{R}$ if $y \in \mathbb{R}$.

Definition 5.3.3 *A function $f : A \to B$ that is both injective and surjective is called a* bijection.

Example 5.3.4 *The following functions are bijective:*

1. *$f : \mathbb{Z} \to \mathbb{Z}$ defined by $f(n) = n + 1$*
2. *$g : \mathbb{P} \to \mathbb{N}$ defined by $g(n) = n - 1$*
3. *$h : \mathbb{S}_4 \to \{a, b, c, d\}$ defined by $h(1) = a$, $h(2) = b$, $h(3) = c$, $h(4) = d$*

Theorem 5.3.5 *A function $f : A \to B$ is invertible if and only if it is a bijection.*

Proof *We need to prove both implications.*

$\boxed{\Rightarrow}$ *Assume that $f : A \to B$ is invertible, i.e. the relation f^{-1} is itself a function. We need to prove that f is both injective and surjective.*

injective *Let $a_1, a_2 \in A$ such that $f(a_1) = f(a_2)$. By applying f^{-1} on both sides of the equality (knowing that f^{-1} is a function) we get $f^{-1}(f(a_1)) = f^{-1}(f(a_2))$. Because $f^{-1}(f(x)) = x$ for any $x \in A$, we get $a_1 = a_2$, which proves that f is injective.*

surjective *Let $b \in B$. Because f^{-1} is a function there exists $f^{-1}(b) \in A$. But $f(f^{-1}(x)) = x$ for all $x \in A$ so $f(f^{-1}(b)) = b$. Therefore, $f^{-1}(b)$ is an input for f that produces b as an output, which shows that f is surjective.*

Being both injective and surjective, f is a bijection.

$\boxed{\Leftarrow}$ *Assume that f is a bijection, i.e. f is both injective and surjective. We need to prove that the relation f^{-1} is a function, which we do by checking the two properties that characterise a function.*

existence of an output *Let $b \in B$. Because f is surjective, we know that there exists $a \in A$ such that $f(a) = b$, i.e. $(a, b) \in f$ using the notation for f as a relation. But by definition of inverse $(b, a) \in f^{-1}$ so b has the output a through f^{-1}.*

uniqueness of the output *Let $a_1, a_2 \in A$ and $b \in B$ such that $(b, a_1), (b, a_2) \in f^{-1}$. By definition of inverse this means that $(a_1, b), (a_2, b) \in f$. But f is injective so $a_1 = a_2$, i.e. the two outputs for f^{-1} on b are the same.*

Therefore f^{-1} is a function.

5.3.1 Finite cardinality

The following theorems shows, for finite sets, useful relationships between injective and surjective functions and cardinality. They reflect what is called the *Pigeonhole Principle*:

> If k is a positive integer and $k+1$ or more objects are placed into k boxes, then there is at least one box containing two or more of the objects.

Theorem 5.3.6 *Let A, B be finite sets, and let $f : A \to B$ be a function.*

1. *$|f(A)| \leq |A|$.*
2. *If $|f(A)| = |A|$ then f is injective, and if $|f(A)| = |B|$ then f is surjective.*
3. *If f is injective then $|f(A)| = |A|$ and $|A| \leq |B|$.*
4. *If f is surjective then $f(A) = B$ and $|A| \geq |B|$.*
5. *If f is a bijection then $|A| = |B|$.*

From this theorem, the following properties follow:

Corollary 5.3.7 *Let A and B be finite sets.*

1. *If $|A| = |B|$ and f is injective then f is surjective.*
2. *If $|A| = |B|$ and f is surjective then f is injective.*

5.4 Cardinality of infinite sets

If A is a finite set then $|A|$ is the number of elements in A. What if A is infinite? Some infinite sets are 'larger' than others. For example, the 'size' of the natural numbers is

less than the 'size' of the reals. However, the 'size' of $\mathbb{N}$ is the same as the 'size' of $\mathbb{Z}$ and the 'size' of $\mathbb{Q}$. Furthermore, the 'size' of $\mathbb{R}$ is the same as the 'size' of the interval $(0,1) = \{r \in \mathbb{R} \mid 0 < r < 1\}$.

In Section 5.3.1, we showed how the fact that a function is injective or surjective relates to the cardinalities of its domain and codomain if they are finite. We now proceed along the same lines to compare $|A|$ and $|B|$ when these sets are infinite.

Definition 5.4.1 *Let A, B be any sets.*

1. *We say $|A| \leq |B|$ if there exists an injective function $f : A \to B$.*
2. *We say $|A| \geq |B|$ if there exists a surjective function $f : A \to B$.*

Note that if A is finite and B is infinite then there is an injective map $f : A \to B$, but there is no injective map $g : B \to A$ (or surjective map $f : A \to B$). So $|A| < |B|$.

Thus we have that $|A| = |B|$ if there exists a bijection $f : A \to B$, and we say that A and B have the same *cardinality*.

Theorem 5.4.2 $|\mathbb{N}| = |\mathbb{P}|$.

Proof *All we need to do is find a bijection between $\mathbb{N}$ and $\mathbb{P}$. Consider the function $+1 : \mathbb{N} \to \mathbb{P}$ given by, for all $n \in \mathbb{N}$: $+1(n) = n + 1$ This function is clearly a bijection, which proves the theorem.*

This function has been epitomised by *Hilbert's hotel*.

> Imagine a hotel with $|\mathbb{N}|$ many rooms where rooms are numbered $0, 1, 2, \ldots$. If the hotel is full, one more guest can be accommodated as follows: for every n, the guest in room n are asked to move to room $n + 1$, freeing room 0 for the new guest. This can be repeated any number of times so even if a bus arrives with new guests, they can all be accommodated.

What about $|\mathbb{Z}|$? Let us go back to Hilbert's hotel:

> What happens if a full bus with as many seats as $|\mathbb{N}|$ arrives (which is plausible if the hotel itself has infinite many rooms) and the hotel is full? We cannot possibly use the function $+1$ an infinite number of times (everyone would die in the meantime)! Instead, for every n, we can ask the guest in room n to move to room $2n + 1$. This frees room number 0 and all even numbered rooms, so all the new guests can be accommodated as well.

Theorem 5.4.3 $|\mathbb{Z}| = |\mathbb{N}|$.

Proof *All we need to do is find a bijection between $\mathbb{Z}$ and $\mathbb{N}$. Consider the function $f : \mathbb{Z} \to \mathbb{N}$ given by the rules, for all $n \in \mathbb{P}$:*

$$f(0) = 0, \quad f(n) = 2n, \quad f(-n) = 2n - 1.$$

This function can be easily proved to be a bijection, which proves the theorem.

Notice that the function is well defined because every integer is either 0 or positive or negative.

Definition 5.4.4 *A set A is said to be* countably infinite *if* $|A| = |\mathbb{N}|$.

Therefore, both $\mathbb{Z}$ and $\mathbb{Z}$ are countably infinite. What about the set $\mathbb{Q}$ of all rational numbers?

Given that every rational number can be expressed as a fraction n/m where $n \in \mathbb{Z}$ and $n \in \mathbb{P}$, we clearly have a surjective mapping $\mathbb{Z} \times \mathbb{P} \to \mathbb{Q}$, which allows us to conclude that $|\mathbb{Q}| \leq |\mathbb{Z} \times \mathbb{P}|$. Note that the mapping is not injective because the same rational number may be expressed by more than one fraction. On the other hand, the function $f : \mathbb{N} \to \mathbb{Q}$ defined by $f(n) = n$ is obviously injective (but not surjective), so $|\mathbb{N}| \leq |\mathbb{Q}| \leq |\mathbb{Z} \times \mathbb{P}|$.

Therefore the question now is how to calculate $|\mathbb{Z} \times \mathbb{P}|$. We can make a further simplification by using the bijections f and $+1$ defined in the proofs of Theorems 5.4.3 and 5.4.2. Consider the function $g : \mathbb{Z} \times \mathbb{P} \to \mathbb{N} \times \mathbb{N}$ defined by, for every $(n, m) \in \mathbb{Z} \times \mathbb{P}$, $g(n, m) = (f(n), +1(m))$. Because f and $+1$ are bijections, g can easily be proved to be a bijection as well. Therefore, $|\mathbb{Z} \times \mathbb{P}| = |\mathbb{N} \times \mathbb{N}|$ meaning that what remains is to calculate $|\mathbb{N} \times \mathbb{N}|$.

To understand how, let us go back to Hilbert's hotel:

> Imagine now that the hotel is empty and an infinite number of infinite buses arrive. Assume that buses are numbered $0, 1, \ldots$ and every bus has seats numbered $0, 1, \ldots$. We start by letting passenger 0 in bus 0 in, i.e., the pair $(0, 0)$. Then, we place passenger 0 in bus 1, i.e. the pair $(0, 1)$, and passenger 1 in bus 0, i.e. the pair $(1, 0)$. Then the pairs $(2, 0)$, $(1, 2)$ and $(0, 2)$. Then the pairs $(0, 3)$, $(1, 2)$, $(2, 1)$ and $(0, 4)$. And so on. This process is illustrated in Figure 5.1.

This mapping is defined for every pair of natural numbers (every passenger in every bus gets a room) and is clearly injective (no room will be allocated to more than one passenger). Therefore, we can conclude that $|\mathbb{Z} \times \mathbb{P}| = |\mathbb{N} \times \mathbb{N}| \leq |\mathbb{N}|$. Given that we had already proved that $|\mathbb{N}| \leq |\mathbb{Q}| \leq |\mathbb{Z} \times \mathbb{P}|$, if follows that $|\mathbb{Q}| = |\mathbb{N}|$ (and that $|\mathbb{N} \times \mathbb{N}| = |\mathbb{N}|$, which is also useful).

This proves the following theorem:

Theorem 5.4.5 *The rationals are countably infinite, i.e.* $|\mathbb{Q}| = |\mathbb{N}|$.

Among the infinite sets that we know well, that of the real numbers remains. Is $\mathbb{R}$ countably infinite as well? The map $f : \mathbb{N} \to \mathbb{R}$ given by $f(n) = n$ is clearly injective, so $|\mathbb{N}| \leq |\mathbb{R}|$. However, the converse is not true.

Theorem 5.4.6 $\mathbb{R}$ *is not countably infinite, so* $|\mathbb{N}| < |\mathbb{R}|$.

Proof *The proof is by contradiction (see Section 2.4.3). Suppose that* $|\mathbb{N}| \geq |\mathbb{R}|$. *Then there exists a surjective function* $f : \mathbb{N} \to \mathbb{R}$. *We write each element* $f(n)$ *as a decimal:*

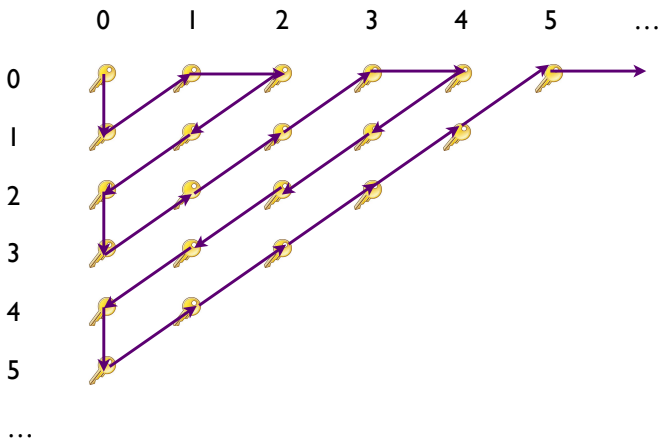


Figure 5.1: Placing $\mathbb{N} \times \mathbb{N}$ in $\mathbb{N}$

$$
\begin{aligned}
f(0) &= k_0.a_{00}a_{01}a_{02}a_{03}a_{04}\ldots \\
f(1) &= k_1.a_{10}a_{11}a_{12}a_{13}a_{14}\ldots \\
f(2) &= k_2.a_{20}a_{21}a_{22}a_{23}a_{24}\ldots \\
f(3) &= k_3.a_{30}a_{31}a_{32}a_{33}a_{34}\ldots \\
f(4) &= k_4.a_{40}a_{41}a_{42}a_{43}a_{44}\ldots \\
&\vdots
\end{aligned}
$$

Now define digits $b_0, b_1, b_2, b_3, \ldots$ *by the rule*

$$b_i = \begin{cases} 1 & \text{if } a_{ii} \neq 1 \\ 2 & \text{if } a_{ii} = 1 \end{cases}$$

So if $f(0) = 34.345124\ldots$ *then* $b_0 = 1$*, if* $f(1) = -4.115198\ldots$ *then* $b_1 = 2$*, etc. From the definition, we can see that* $b_i \neq a_{ii}$*, for all* $i \in \mathbb{N}$*. Now let* $r = 0.b_0b_1b_2b_3b_4\ldots$*.*

We have that $r \in \mathbb{R}$ *but:*

0) $r \neq f(0)$, *since* $b_0 \neq a_{00}$

1) $r \neq f(1)$, *since* $b_1 \neq a_{11}$

2) $r \neq f(2)$, *since* $b_2 \neq a_{22}$

...

i) $r \neq f(i)$, *since* $r_i \neq a_{ii}$

...

Therefore, f is not surjective. This is a contradiction, so we have that $|\mathbb{R}| \not\leq |\mathbb{N}|$. Hence, $|\mathbb{R}| \neq |\mathbb{N}|$, and $\mathbb{R}$ is not countably infinite.

The method we have used in the proof is called a 'diagonalization argument'. We listed all the numbers, then we took the digits from the diagonal and used them to construct a new number not on the list.

5.5 Indexing

Any function from a set A to the set of natural numbers defines a chain (see Definition 4.1.8) over A.

Proposition 5.5.1 *Given $f : A \to \mathbb{N}$ we define a chain $\langle A, \prec \rangle$ through $a \prec b$ if $f(a) < f(b)$.*

This process is usually referred to as *indexing* a set, which has many applications in real life, an example of which is given below.

5.5.1 Data compression

We can also use orderings, and in particular indexing, to compress data so that it can be electronically transmitted more quickly and stored in less space.

Words consist of strings of letters, and when a document is electronically transmitted each of the letters in each of the words must be transmitted. If we can code the words so that the coded word contains fewer characters than the letters which make up the word then transmitting and storing the text can be more efficient.

A simple coding is to assign a natural number to each word, and then, if there are not too many different words, the integer version of the document will be smaller than the original version. For example, if there are only 999 words all of length 4 then we will save between 3 and 1 characters per word when we represent the words as integers.

It is not just the length of the words which is important but also their frequency. For example, if a word of length three occurs twenty times then replacing it with 1 saves 40 characters, and replacing it with 12 saves 20 characters. Whereas if a word of length ten occurs only once, then replacing it with 1 saves 9 characters while replacing it with 12 saves 8 characters. So it is better to replace the word of length three with 1 and the word of length ten with 12, rather than the other way round.

Thus we do not want to pre-assign natural numbers to words, we don't want to use a fixed coding. The assignment of natural number to words is done separately for each file which is compressed.

The basis of many common compression algorithms is to order the words of the document according to the number of times the word appears in the document. The front of the

compressed document then contains a list of all the words in the document, and the order of this list defines the natural numbers which are used to represent the words.

In this set-up all the words in the document are stored in their full form once, so the compressed file is only smaller than the original file if some of the words are repeated. Although that this is usually the case, it is possible for a 'compressed' file to be larger than the uncompressed version.

Better compression can be achieved by indexing the strings of characters which appear in a file, rather than using the words (that is the spaces) to determine which strings will be represented as a natural number. For more on the topic of compression you should look up Huffman coding and LZ compression on the web.

5.6 Exercises

1. Prove that the relation $R = \{(x, 2^x) \mid x \in \mathbb{N}\}$ is a function $f : \mathbb{N} \to \mathbb{N}$. Decide whether it is injective, surjective, and/or a bijection.

2. Let $f : A \to B$ be an invertible function. Prove that f is a bijection.

3. Find the total number of functions from $\mathbb{S}_2$ to $\{a, b, c\}$.

4. Let $R = \{(1, 2), (2, 3), (3, 1)\}$ be a relation on $\mathbb{S}_3$. Show that R is a function f, that f is invertible, and give the graphical representation of f^{-1}.

5. Let $K = \{-n \mid n \in \mathbb{N}\}$, prove that K is countably infinite.

6. Let $A = \{a_1, a_2, \ldots, a_n\}$ be a finite set, and suppose that $a_i = a_j$ if and only if $i = j$. Prove that $|A| \leq |\mathbb{N}|$.

7. Prove that the map $f : \mathbb{Z} \to \mathbb{N}$ given by the rules

$$f(0) = 0, \quad f(n) = 2n, \quad f(-n) = 2n + 1, \quad \text{for } n \in \mathbb{P},$$

is a bijection.

Chapter 6

Recursion and Induction

Quantities have to be defined before they can be used, particularly before a computer can use them. For example, before proving that $n! > 2^n$ we need to know what $n!$ and 2^n are.

If there are only finitely many elements that can appear then we can, at least in principle, write them out. This is the core of a data base: all the elements are stored and when information is required the corresponding entry is looked up in the data base.

However, data bases can be very large and, for infinite sets of data, data bases cannot be constructed. An alternative approach is to give a basic set of data and a set of rules for calculating the other data elements. If the data set is infinite and the set of rules is to be finite (otherwise they cannot be given to a computer) then the rules must in some sense be recursive. In this chapter we shall look at simple recursive functions, and the related method of proof known as induction.

(*Most of the material in this chapter can be found in Chapter 5 of Rosen.*)

6.1 Recursion

6.1.1 Recursive definitions

We often write

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$$

$$2^n = 2 \times 2 \times 2 \times \ldots \times 2, \qquad n \text{ times.}$$

This works well for humans, but not necessarily for computers as they do not understand 'dots'. How can we then define $n!$ and 2^n in a way that can be programmed?

Recursion is such a method: we state the value of the function on a first value and we give an algorithm for calculating the $(n+1)^{st}$ case in terms of the previous n cases.

For example, we define $n!$ for $n \in \mathbb{N}$ as follows:

$$\begin{aligned} 0! &= 1 \\ (n+1)! &= (n+1) \times n! \end{aligned}$$

We can now compute factorials by following the rule: $0! = 1, \quad 1! = 1 \times 1, \quad 2! = 2 \times 1! = 2 \times 1 = 2, \quad 3! = 3 \times 2! = 3 \times 2 = 6, \quad 4! = 4 \times 3! = 4 \times 6 = 24.$

We think of factorial as a function from $\mathbb{N}$ to $\mathbb{N}$, and write an algorithm to implement it:

```
if n==0 then factorial(n) = 1
else factorial(n) = n * factorial(n-1)
```

This algorithm is *recursive* because it calls itself. It does not go on calling itself for ever though, because eventually it calls *factorial(0)* which is equal to 0.

Definition 6.1.1 *A definition of a function that uses its own definition on a smaller value is called a* recursive *or* inductive definition*.*

For example, a recursive definition of 2^n is

$$\begin{aligned} & & 2^0 &= 1 \\ \text{for } n \geq 1, & & 2^n &= 2^{n-1} \times 2. \end{aligned}$$

The corresponding recursive algorithm is

```
if n==0 then exponent_2(n) = 1
else exponent_2(n) = exponent_2(n-1) * 2
```

6.1.2 Recurrence relations

Sometimes the definition of an element may depend on several smaller elements. For example, we may wish to describe a function f by saying how to calculate it on n in terms of its value on $n-1$ and $n-2$:

$$f(n) = f(n-1) + f(n-2)$$

Then if we know $f(0)$ and $f(1)$ we can calculate $f(n)$ for all $n \in \mathbb{N}$.

Definition 6.1.2 *A* recurrence relation *is an equation that defines the n^{th} element of a sequence in terms of the preceding $n-1$ values.*

The sequence obtained by adding together the previous two values turns up in many different areas, and has been given a name: the Fibonacci sequence.

$$fib(0) = 1, fib(1) = 1 \text{ and, for } n \geq 2, \quad fib(n) = fib(n-1) + fib(n-2).$$

So the sequence begins

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

A recursive algorithm for calculating $fib(n)$, the n^{th} Fibonacci number is

```
if (n==0) or (n==1) then fib(n) = 1
else fib(n) = fib(n-1) + fib(n-2)
```

6.1.3 Recursion and iteration

It makes sense to *define* quantities such as $n!$ and the Fibonacci numbers recursively but that is not always the best way to calculate them. If we calculate $factorial(5)$ using its recursive definition, we need to find $factorial(4)$ and, before that, $factorial(3)$ and $factorial(2)$. This way we would have to do quite a bit of back substitution.

If we were to calculate 5! without the recursive definition we would start at 2 and multiply up until we reached 5, so $2 \times 3 = 6$, $6 \times 4 = 24$, $5 \times 24 = 120$. The same is true for recurrence definitions. If we have $G(2) = 2$, $G(3) = -1$, $G(n) = 2G(n-2) + 3G(n-1)$ and we want to find $G(6)$ we don't start with $G(6)$ and back substitute. We have

$$G(4) = 2 \times 2 + 3 \times -1 = 1, \quad G(5) = 2 \times -1 + 3 \times 1 = 1, \quad G(6) = 2 \times 1 + 3 \times 1 = 5$$

Recursion is natural for defining quantities but it also naturally leads us to calculate values from the top down, using back substitution. Calculation from the bottom up is familiar to computer scientists because programming languages usually support *iteration* in the form of `for` and `while` loops.

```
[int fac_x = 1
for i = 1 to 5 do
     fac_n = fac_n * i
]
```

```
[int G_a = 2, G_b = -1
int i = 4
while i <= 6 do
     G_n = 2*G_a +3*G_b
     G_a = G_b
     G_b = G_n
     i = i+1
]
```

We can find examples of notations in mathematics that correspond to `for` loops. For example, consider the following algorithm:

```
[int x = 0
for i = 1 to n do
     x = x+i
]
```

This algorithm defines a function $S : \mathbb{N} \to \mathbb{N}$ — given a value for `n`, its execution terminates and we can define $S(n)$ as the result of the variable `x` when it terminates. For example, if $n = 0$, the `for` loop is not executed and returns $x = 0$, so $S(0) = 0$; if $n = 1$, then we execute the `for` loop once and return $x = 0 + 1 = 1$, so $S(1) = 1$; if $n = 2$, then we execute the `for` loop twice: after the first execution we get $x = 0 + 1 = 1$, and after the second we get $x = 0 + 1 + 2 = 3$ so $S(2) = 3$.

It is intuitive to guess that

$$S(n) = 0 + 1 + 2 + \ldots + n = \sum_{i=0}^{n} i$$

That is, $\sum$ is the mathematical version of a `for` loop.

How can we be sure that this is indeed the case, i.e. that when the `for` loop is executed for a given n, at the end of the execution $x = S(n)$? Computer scientists like Tony Hoare and (the late) Edsger Dijkstra, both winners of the Turing Award (the Nobel prize of Computer Science), came up with methods through which this can be proven, which we now illustrate.

We start by giving a recursive definition of $S(n) = \sum_{i=0}^{n} i$. Given that the domain of S is $\mathbb{N}$, we start at $n = 0$, so

$$S(0) = \sum_{i=0}^{0} i = 0.$$

On the other hand,

$$S(n+1) = \sum_{i=0}^{n+1} i = \left(\sum_{i=0}^{n} i \right) + (n+1) = S(n) + (n+1)$$

The second step is to enrich our notation for algorithms with what we call *statements*. Statements are propositions (they can true or false) that are made part of the code between curly brackets: they are meant to be true after the code that precedes them, i.e., the preceding code establishes those propositions to be true. For example,

```
[int x = 0
{x=0}
for i = 1 to n do
     x = x+i
]
```

expresses that the variable x has the value 0 after it has been initialised to 0, which is obviously true. Notice that we use = in the code (as in Java) to express assignments but = in the statements (as in mathematics) to express equality between values.

Such statements are particularly important for expressing *invariants*, i.e. properties that are preserved by the body of the loop. For example,

```
[int x = 0
{x=0}
for i = 1 to n do
     {x=S(i-1)}
     x = x+i
     {x=S(i)}
]
```

That is, if when we enter the loop the property $x = S(i-1)$ is true, then after executing `x = x+i` (the body of the loop) the property $x = S(i)$ is also true. Because when we next enter the loop i is updated to $i+1$, this means that $x = S(i-1)$ will again be true for the new value of i, which is why it is called an invariant. How then can we prove this is correct?

Firstly, we need to prove that, when we enter the loop the first time, $x = S(i-1)$ is true. This is correct because the first time we enter the loop $i = 1$ and $x = 0$, and so $x = S(i-1) = S(0) = 0$

Secondly, we need to prove that if we enter the loop with $x = S(i-1)$ then $x = S(i)$ is true after we execute `x = x+i`. Now, after executing `x = x+i`, x is updated to $x + i$. Therefore, what we need to prove is that if $x = S(i-1)$ then $x + i = S(i)$. But this is true because, according to the recursive definition, $S(i) = S(i-1) + i$; therefore, if $x = S(i-1)$, we get $x + i = S(i-1) + i = S(i)$.

Finally, we exit the loop when $i = n$. If we replace i with n in $x = S(i)$ we get $x = S(n)$, which is what we wanted to prove.

We can use this approach for any function F defined on the integers. Suppose that we have an algorithm that consists of some initalisation code, `initialise`, which initialises a variable x, and a `for` loop with a code body, `body`, which updates the value of x.

```
[initialise
for i = k+1 to n do
     body
]
```

We can enrich this algorithm with statements that x has value $F(k)$ after the initialisation, $F(i-1)$ at the start of the loop, and $F(i)$ at the end of the loop.

```
[initialise
{int x=F(k)}
for i = k+1 to n do
```

```
     {x=F(i-1)}
     body
     {x=F(i)}
]
```

We have the following theorem.

Theorem 6.1.3 *Let F be a function defined on all integers $n \geq k$ where $k \in \mathbb{Z}$. If the following enriched* `for` *loop is correct*

```
[initialise
{x=F(k)}
for i = k+1 to n do
     {x=F(i-1)}
     body
     {x=F(i)}
]
```

then $x = F(n)$ at the end of the execution.

For example, consider the `for` loop for factorial.

```
[int x = 1
for i = 1 to n do
     x = x * i
]
```

Its enriched version is

```
[int x = 1
{x = 0!}
for i = 1 to n do
     {x = (i-1)!}
     x = x * i
     {x = i!}
]
```

The proof of correctness is as follows:

1. $x = 0!$ is true after the initialisation: this holds because `x` is initialised to 1 and $0! = 1$.

2. $x = (i-1)!$ is true the first time we enter the loop: this holds because the first time we enter the loop `i` is 1 and $x = 0! = (i-1)!$.

3. If $x = (i-1)!$ is true then $x = i!$ is true after executing `x = x*i`: this holds because if $x = (i-1)!$ then $x * i = (i-1)! * i = i!$ and $x * i$ is the value of `x` after executing `x = x*i`.

If you want to know more about how to check that programs are correct in relation to a specification of what they are meant to produce the following is a good book:

David Gries. *The Science of Programming.* (Library code 001.642 GR)

6.2 Induction

6.2.1 The towers of Hanoi

There is a well known puzzle (the towers of Hanoi Problem) which consists of three poles and a set of discs of different sizes. The discs are put on one pole in decreasing size order.

The challenge is to move all the discs from one pole to another pole without ever putting a larger disc on top of a smaller one, so that all the discs end up in the correct order on the other pole. The top image of Figure 6.1 shows one such starting configuration: all discs are on pole A. The challenge is to move them to either pole B or pole C under those constraints.

Can we prove that the challenge can be completed for any number of discs? The answer is 'yes' and we demonstrate it as follows:

1. If there is only one disc then we may move it where we want. So the challenge is easily met. This is illustrated in the second image of Figure 6.1.

2. Suppose that there are $n + 1$ discs, where n is a natural number and $n \geq 1$, and suppose that the challenge can be completed when there are n discs. So we assume that we can move the top n discs to either of the other two poles in the permitted manner.

3. Under that assumption, we first move the top n discs from pole A to pole B and then the largest disc to C (as in point 1). This is illustrated in the third image of Figure 6.1.

4. We now have n disks on pole B so, using our assumption, we can move them to pole C in the permitted manner. This is illustrated in the bottom image of Figure 6.1.

5. This completes the challenge.

What have we done? We have shown that the challenge can be met for one disc, and we have shown that if the challenge can be met for n discs, then it can be met for $n + 1$ discs.

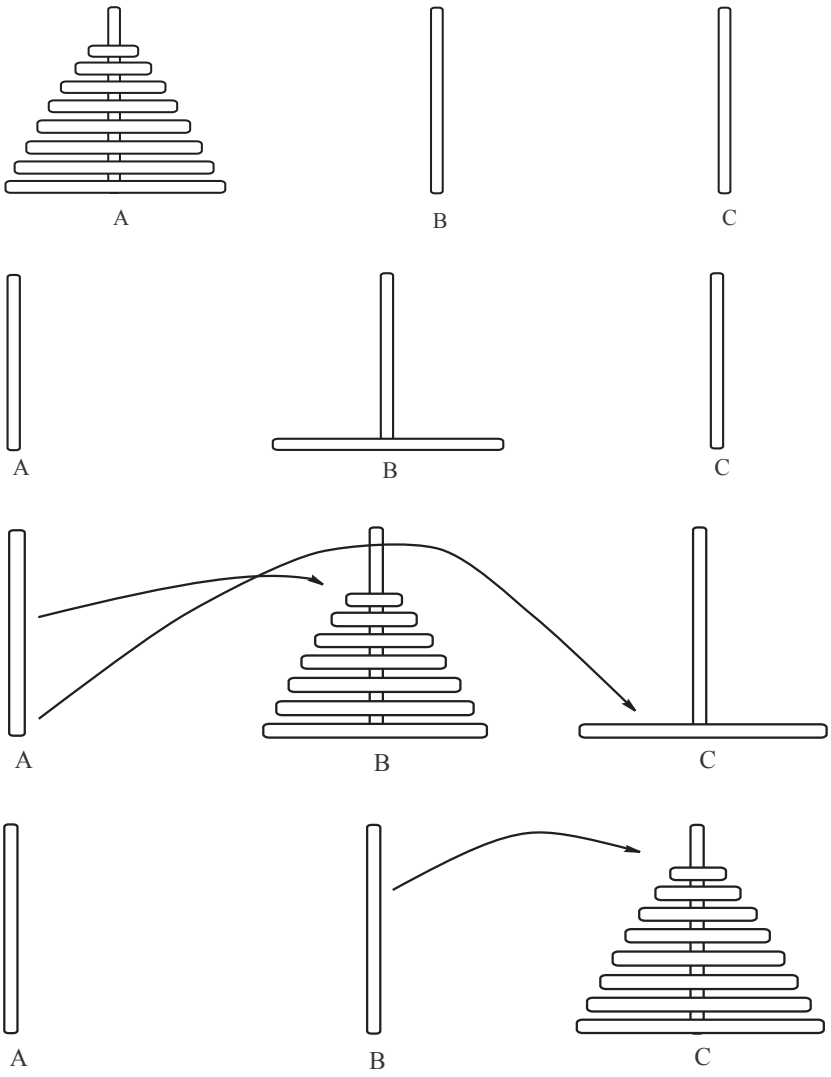


Figure 6.1: The towers of Hanoi

If we are given any positive integer k then this argument shows that the challenge can be completed for k discs. Thus we have shown that it can be done for all integers $k \geq 1$.

This method of proving a result for all integers in a given range is called *induction*.

6.2.2 Mathematical induction

The following result is the principle of mathematical induction:

Theorem 6.2.1 *To prove a property $P(n)$ of all integers $n \geq k$ (where $k \in \mathbb{Z}$), it is sufficient to*

1. *prove $P(n)$ when $n = k$ (so prove $P(k)$), and*
2. *prove that, for all $n \in \mathbb{Z}$, if $P(n)$ and $n \geq k$ are true then $P(n+1)$ is true*

Step (1) is called the *base case*, step (2) is called the *inductive step*, and '$P(n)$ and $n \geq k$ is true' is called the *induction hypothesis* (often written I.H.).

Example 6.2.2 *Prove that for all $n \in \mathbb{N}$ such that $n \geq 5$, $2^n > n^2$.*

Solution

Base case: When $n = 5$ we have $2^n = 2^5 = 32 > 25 = 5^2 = n^2$. Thus the base case is proved.

Induction hypothesis: Suppose that $2^n > n^2$ is true for a fixed arbitrary $n \geq 5$.

Inductive step: We have that

$$\begin{aligned} 2^{n+1} &= 2^n \times 2 \\ &> n^2 \times 2, \qquad \text{(by the induction hypothesis)} \\ &= n^2 + n^2 \\ (n+1)^2 &= n^2 + 2n + 1 \end{aligned}$$

Because $n \geq 5$, we have $n^2 \geq 5n = 2n + 3n \geq 2n + 15 > 2n + 1$.

So, $2^{n+1} > n^2 + n^2 > n^2 + 2n + 1 = (n+1)^2$.

This proves the inductive step.

So the result follows by induction.

Notice that the base case is $n = 5$ because we need to prove that $2^n > n^2$ holds for all $n \geq 5$.

Example 6.2.3 *Prove by induction that for all* $n \in \mathbb{N}$

$$S(n) \;=\; \sum_{i=0}^{n} i \;=\; \frac{n(n+1)}{2}$$

Solution

We start by defining $S(n)$ inductively: $S(0) = 0$, $S(n+1) = S(n) + (n+1)$.

Base case: When $n = 0$ we have

$$\frac{n(n+1)}{2} \;=\; \frac{0}{2} \;=\; 0 \;=\; S(0).$$

Thus the base case is proved.

Induction hypothesis: Suppose that $S(n) \;=\; \frac{n(n+1)}{2}$ for a fixed arbitrary $n \in \mathbb{N}$.

Inductive step: We have that

$$\begin{aligned} S(n+1) \;=\; S(n) + (n+1) \quad &= \quad \frac{n(n+1)}{2} + (n+1) \quad \text{(by induction hypothesis)} \\ &= \quad \frac{n^2 + n + 2n + 2}{2} \\ &= \quad \frac{(n+1)(n+2)}{2} \end{aligned}$$

Thus the result follows by induction.

Notice that the base case involves a calculation:

$$\frac{n(n+1)}{2} \;=\; \frac{0}{2} \;=\; 0$$

This is used to compare with the value of $S(0)$. Just repeating $S(0) = 0$, which is part of the definition of S, does not prove anything.

Also notice that the induction hypothesis is NOT

$$S(n) = \frac{n(n+1)}{2} \textit{ for all } n \in \mathbb{N}$$

That is the result that we had to prove! The induction hypothesis is about a single, fixed and arbitrary $n \in \mathbb{N}$.

The following example will be useful in Chapter 9:

Example 6.2.4 *Prove by induction that for all* $m \in \mathbb{N}$*, and* $x \in \mathbb{R}$ *such that* $x \neq 1$*:*

$$\sum_{i=0}^{m} x^i \;=\; \frac{x^{m+1} - 1}{x - 1}$$

Solution

Base case: When $m = 0$ we have

$$\sum_{i=0}^{0} x^i \ = x^0 = 1$$

On the other hand

$$\frac{x^{0+1} - 1}{x - 1} \ = \ \frac{x - 1}{x - 1} \ = 1$$

Thus the base case is proved.

Induction hypothesis: Suppose that $\sum_{i=0}^{m} x^i \ = \ \frac{x^{m+1} - 1}{x - 1}$ for a fixed arbitrary $m \in \mathbb{N}$.

Inductive step: We have that

$$\begin{aligned}
\sum_{i=0}^{m+1} x^i \ &= \left(\sum_{i=0}^{m} x^i \right) + x^{m+1} \\
&= \frac{x^{m+1} - 1}{x - 1} + x^{m+1} \quad \text{(by induction hypothesis)} \\
&= \frac{x^{m+1} - 1 + x^{m+2} - x^{m+1}}{x - 1} \\
&= \frac{x^{m+2} - 1}{x - 1}
\end{aligned}$$

The result thus follows by induction.

Induction arguments can involve more complex situations such in the following example.

Example 6.2.5 *Prove that for $n, m \in \mathbb{N}$, $a^n a^m = a^{n+m}$.*

Solution We prove the result by induction on n, i.e. the property $P(n)$ is 'for all $m \in \mathbb{N}$, $a^n a^m = a^{n+m}$'. We also use the recursive definition of a^n: $a^0 = 1$ and $a^{n+1} = a^n \times a$. We need to use the fact that, because multiplication is commutative, for all $n \in \mathbb{N}$, we have $a^n \times a = a \times a^n$.

Base case: For $n = 0$ we have $a^0 a^m = 1 \times a^m = a^m = a^{0+m}$.

Induction hypothesis: Suppose that, for a fixed arbitrary $n \in \mathbb{N}$, it is true that for all $m \in \mathbb{N}$, $a^n a^m = a^{n+m}$ (i.e. $P(n)$ is true).

Inductive step: We have

$$\begin{aligned} a^{n+1} \times a^m &= (a^n \times a) \times a^m && \text{(by definition of } a^{n+1}) \\ &= a^n \times (a \times a^m) && \text{(because } \times \text{ is associative)} \\ &= a^n \times (a^m \times a) && \text{(because } \times \text{ is commutative)} \\ &= a^n \times a^{m+1} && \text{(by definition of } a^{m+1}) \\ &= a^{n+(m+1)} && \text{(by the induction hypothesis)} \\ &= a^{(n+1)+m} \end{aligned}$$

Thus the result follows by induction.

6.2.3 Well-founded induction

The principle of mathematical induction is a particular case of a more general induction principle over well-founded posets (see Definition 4.4.8).

Theorem 6.2.6 *Let $\prec$ be a well-founded ordering on a set A. To prove that a property P holds for all elements of A it suffices to prove that, for all $a \in A$, P holds for a if P holds for all $b \prec a$.*

Proof *The proof is by contradiction. Suppose that the principle is not valid and therefore there is a property P such that, for any $a \in A$, $P(a)$ is true if $P(b)$ is true for all $b \prec a$, but $P(a_0)$ is false for some $a_0 \in A$. In that case, there must be $a_1 \prec a_0$ such that $P(a_1)$ is false (otherwise $P(a_0)$ would be true). But if $P(a_1)$ is false there must be $a_2 \prec a_1$ such that $P(a_2)$ is false. Continuing in this fashion, we can construct an infinite descending chain $a_0 \succ a_1 \succ a_2 \succ \ldots$, which contradicts the fact that $\prec$ is well-founded.*

Note that the base cases of the well-founded induction are the minimal elements of A, i.e. the elements $a \in A$ such that there is no $b \prec a$.

If we now consider $\mathbb{N}$ with the usual ordering $<$, which is well-founded, the corresponding induction principle is as follows:

Corollary 6.2.7 *To prove that a property P holds for all $n \in \mathbb{N}$ it suffices to prove that, for all $n \in \mathbb{N}$, $P(n)$ is true if $P(k)$ is true for all $k < n$.*

This is called the principle of *strong* induction.

Example 6.2.8 *Suppose that $G(0) = 1$, $G(1) = 3$ and that, for $n \geq 2$, $G(n) = 2G(n-1) + 3G(n-2) + 4$. Prove that, for all $n \in \mathbb{N}$, $G(n)$ is an odd number.*

Solution

Base case: If $n = 0$ then $G(n) = 1$, which is an odd number, and if $n = 1$, $G(1) = 3$ which is an odd number.

Induction hypothesis: Suppose that the result is true for all k such that $k \leq n$ for a fixed and arbitrary $n \geq 1$.

Inductive step: By the definition of G and given that $n \geq 1$ implies $(n+1) \geq 2$ we have

$$G(n+1) = 2G(n) + 3G(n-1) + 4$$

By the induction hypothesis we have that $G(n-1)$ is an odd number, so $3G(n-1)$ is also an odd number. Because $2G(n)$ and 4 are both even numbers, so is their sum. Therefore $G(n+1)$ is the sum of an even number and an odd number, which is an odd number as required.

Thus the result follows by strong induction.

6.3 Exercises

1. Write down $T(6)$ if $T(1) = 1$ and, for $n \geq 2$, $T(n)$ is given by the recurrence relation

$$T(n) = 2T(n-1) + 1$$

2. Give the recurrence relation that defines the (infinite) sequence $0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5....$

3. Write down $T(7)$ if $T(1) = 1, T(2) = 1$ and, for $n \geq 3$, $T(n)$ is given by the recurrence relation

$$T(n) = 3T(n-1) - 2T(n-2)$$

4. Give a proof by induction that $n^2 > 3n$ for all $n \in \mathbb{N}$ such that $n \geq 4$.

5. Prove by induction that, for $n \in \mathbb{N}$, if $n \geq 4$ then $n! > 2^n$.

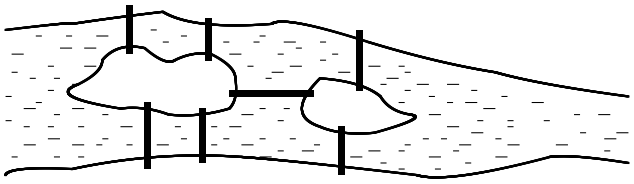
6. For $n \in \mathbb{N}$ and $a \in \mathbb{R}$ we use the following recurrence relation to define a^n: $a^0 = 1$, and for $n \geq 1$, $a^n = a^{n-1} \times a$. Prove by induction on n that for all $n, m \in \mathbb{N}$, $a^n \times a^m = a^m \times a^n$.

   (Note: you cannot use the fact that $a^n \times a^m = a^{n+m}$ because we used the property $a^n \times a^m = a^m \times a^n$ to prove this fact.)

Chapter 7

Graphs and Trees

In the Introduction we saw an example of the use of graphs to help solve the Exmoor villages problem. The use of graphs for problem solving began in the eighteenth century, when a mathematician called Euler was trying to solve what became known as the Königsberg bridge problem. This problem concerns seven bridges that connect two islands in a river to the mainland of the town of Königsberg. The bridges and islands are arranged as in the following picture.



The problem was that the citizens of Königsberg never managed to find a path (beginning and ending anywhere they liked) that crosses each bridge exactly once. Were they unlucky or was it impossible to find such a path? If the latter, how can this be established?

The method that Euler used to tackle this problem was one that computer scientists use many times: to abstract a mathematical model from the real-world situation on which one can reason or compute results. The particular mathematical structure that Euler chose was a graph. Then he developed graph theoretical results that he could apply to the problem.

(*Most of the material in this chapter can be found in Chapters 10 and 11 of Rosen.*)

7.1 Introduction to graph theory

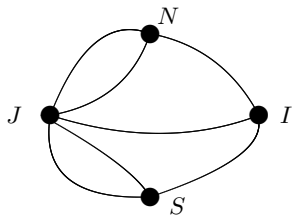
Definition 7.1.1 *A* graph *is a non-empty set of vertices (or nodes) and a set of edges where every edge connects two vertices (its endpoints), these are not necessarily distinct*

(i.e. the two endpoints may be same vertex).

It is possible for there to be more than one edge between two vertices. In this case we sometimes say that the graph is a multigraph.

The following is a multigraph that represents the Königsberg bridge situation. The process of abstraction consists of representing as vertices the locations that are relevant for the problem — the vertices N, S correspond to the north and south banks, and the vertices I, J correspond to the two islands — and representing the bridges as edges.

Notice that although two edges can share a vertex, this does not mean that the corresponding bridges start or end in exactly the same physical location! This is the beauty of abstraction: what matters for the problem is whether they connect the north bank and the first island, or the north bank and the second island, and so on.



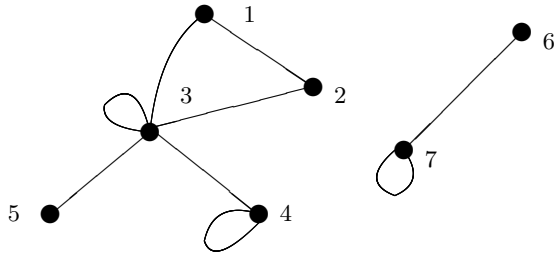
7.1.1 Walks, trails and paths

Definition 7.1.2 *Given a graph:*

- *Two vertices are* adjacent *if there is an edge between them.*
- *A* walk *from vertex N to vertex M is a sequence of vertices $N = N_0, N_1, N_2, \ldots, N_k = M$, with $k \geq 0$, such that N_i is adjacent to N_{i+1} for all $0 \leq i < k$ — i.e. there is an edge between any two consecutive nodes in the sequence.*

Note that if $k = 0$ then we have a walk that consists of a single vertex, which is usually called an *empty* walk.

Example 7.1.3 *Consider the following graph:*



The sequence $1, 2, 3, 5$ *is a walk from* 1 *to* 5*, and so is* $1, 3, 4, 4, 3, 5$*. There is no walk from* 1 *to* 6*.*

It is possible to get very long walks by moving backwards and forwards along the same edge: for example, the walk $1, 2, 1, 2, 1, 2, 3$ in the above graph. We often want to exclude such walks because they involve unnecessary redundancy.

Definition 7.1.4 *Given a graph:*

- *A* trail *is a walk that does not use any edge more than once (i.e. a walk in which no edge is repeated). A trail is also sometimes called a* simple walk*.*
- *A* path *is a walk that does not use any vertex more than once. So any path is also a trail.*

Proposition 7.1.5 *If there is a walk from* N *to* M*, we can find a trail from* N *to* M *and if* $N \neq M$ *this trail can be taken to be a path.*

Proof *If a vertex is repeated in the walk then the walk is of the form*

$$N, \ldots, K, N_i, N_{i+1}, \ldots, N_i, H, \ldots, M$$

In this case we remove everything after the first N_i *and before* H

$$N, \ldots, K, N_i, H, \ldots, M$$

We still have a walk from N *to* M*. We repeat these reductions until the walk contains no repeated vertices apart from the possibility that* $N = M$*. Thus we have a trail, and a path if* $N \neq M$*.*

Definition 7.1.6 *We say that* N, M *are* connected *if there is a walk from* N *to* M*. So* N *is connected to itself by the empty walk. (A vertex is not necessarily adjacent to itself but it is always connected to itself.)*

Definition 7.1.7 *If every vertex is connected to every other vertex we say that* the graph is connected*.*

7.1.2 Dijkstra's algorithm

In this section, we present one of the algorithms through which we can determine a shortest path between two nodes in a connected graph. This algorithm is due to the late Edsger Dijkstra, one of the most influential computer scientists and winner of the Turing Award.

The notion of 'shortest' subsumes a measure associated with edges, like distance in the case of the Exmoor roads problem.

Definition 7.1.8 *A* weighted *graph is a graph together with a function that assigns a number (called weight or cost) to every edge. We define the* length *of a path in a weighted graph to be the sum of the weights of the edges of this path.*

Dijkstra's algorithm takes as input a connected graph and computes a shortest path (with least cost) between an initial vertex N and a destination M. It works as follows:

(1) Set N to be the current vertex and assign it the empty path (with cost 0).

(2) Mark the current vertex as visited. For each of its unvisited neighbours, K,:

  (i) Let P be the concatenation of the path associated with the current vertex and the edge from the current vertex to K.

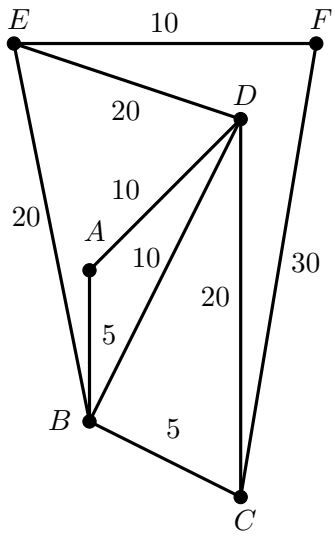
  (ii) If K has not been assigned a path, assign it P.

  (iii) Otherwise, if the cost of the path associated with K is greater than the cost of P, assign P to the vertex K.

(3) Stop if M has been visited. Otherwise, set as current the unvisited vertex with the path of least cost and go to step (2).

Each iteration adds a new vertex A to a set of visited notes and calculates the shortest path from N to A. Naturally, the algorithm terminates when we visit M (which is bound to happen because the graph is connected).

We illustrate the application of the algorithm to the graph of the Exmoor villages. Suppose that we want to find a shortest path between C and F.



The application of Dijkstra's algorithm requires that we mark vertices and remember shortest paths at every step. In this module, we will use a table with three columns — Step, Associated paths, Visited vertices — to document the application of the algorithm.

We start at vertex C.

Step	Associated paths	Visited vertices
1	–	C

Next we need to select the shortest edge that connects C to its nodes. We list the associated paths and the shortest is to B, so we mark B.

Step	Associated paths	Visited vertices
1	–	C
2	$(CB, 5), (CD, 20), (CF, 30)$	C, B

Next we need to consider the unvisited neighbours of B. The vertices A and E don't have assigned paths so we add them. Vertex D has already an assigned path – $(CD, 20)$. Because $(CBD, 15)$ is shortest, we use it instead. The shortest path to an unvisited vertex is now $(CBA, 10)$ so we mark A.

Step	Associated paths	Visited vertices
1	–	C
2	$(CB, 5), (CD, 20), (CF, 30)$	C, B
3	$(CB, 5), (CF, 30), (CBA, 10), (CBE, 25), (CBD, 15)$	C, B, A

Next we need to consider the unvisited neighbours of A; there is only one, D. Vertex D has already an assigned path – $(CBD, 15)$. Because $(CBAD, 25)$ is longer, we keep $(CBD, 15)$. The shortest path to an unvisited vertex is now $(CBAD, 15)$ so we mark D.

Step	Associated paths	Visited vertices
1	–	C
2	$(CB, 5), (CD, 20), (CF, 30)$	C, B
3	$(CB, 5), (CF, 30), (CBA, 10), (CBE, 25), (CBD, 15)$	C, B, A
3	$(CB, 5), (CF, 30), (CBA, 10), (CBE, 25), (CBD, 15)$	C, B, A, D

Next we need to consider the unvisited neighbours of D; there is only one, E. Vertex E has been assigned the path $(CBE, 25)$; because $(CBDE, 35)$ is longer, we keep $(CBE, 25)$. The shortest path to an unvisited vertex is now $(CBE, 25)$ so we mark E.

Step	Associated paths	Visited vertices
1	–	C
2	$(CB, 5), (CD, 20), (CF, 30)$	C, B
3	$(CB, 5), (CF, 30), (CBA, 10), (CBE, 25), (CBD, 15)$	C, B, A
4	$(CB, 5), (CF, 30), (CBA, 10), (CBE, 25), (CBD, 15)$	C, B, A, D
5	$(CB, 5), (CF, 30), (CBA, 10), (CBE, 25), (CBD, 15)$	C, B, A, D, E

Then only unvisited neighbour of E is F, which has been assigned the path $(CF, 30)$; because $(CBEF, 35)$ is longer, we keep $(CF, 30)$. Because F was the last unvisited vertex and is the destination, we stop.

Step	Associated paths	Visited vertices
1	–	C
2	$(CB, 5), (CD, 20), (CF, 30)$	C, B
3	$(CB, 5), (CF, 30), (CBA, 10), (CBE, 25), (CBD, 15)$	C, B, A
4	$(CB, 5), (CF, 30), (CBA, 10), (CBE, 25), (CBD, 15)$	C, B, A, D
5	$(CB, 5), (CF, 30), (CBA, 10), (CBE, 25), (CBD, 15)$	C, B, A, D, E
6	$(CB, 5), (CF, 30), (CBA, 10), (CBE, 25), (CBD, 15)$	C, B, A, D, E, F

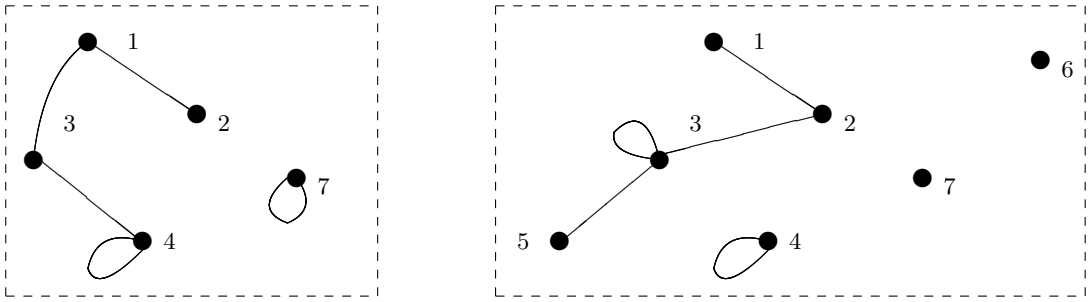
The shortest path from C to F is therefore CF with cost 30. As a by-product, we also have the shortest paths between C and all the other marked vertices, which in this case is all the vertices.

7.1.3 Spanning subgraphs

The Exmoor roads problem discussed in the Introduction could be stated as 'find the shortest connected subgraph that contains all the vertices'.

Definition 7.1.9 *If a graph G is obtained by removing some of the edges or some of the vertices from another graph G' then G is a* subgraph *of G'.*

For example, both of these graphs are subgraphs of the graph in Example 7.1.3.



In the Exmoor roads problem we are not allowed to remove villages from the graph, just roads. (We can't solve the problem just by 'closing' the villages.)

Definition 7.1.10 *A subgraph of G is a* spanning subgraph *if it contains all the vertices of G.*

For example, the subgraph on the right above is a spanning subgraph, the subgraph on the left is not.

The Exmoor roads problem can now be stated precisely as follows:

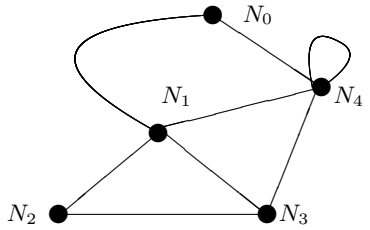
'Find the shortest connected spanning subgraph of the graph of existing roads.'

7.2 Trees

Trees are an important subclass of the class of graphs, for example we shall see that shortest connected spanning subgraphs are trees. In this section we define trees and discuss some of their properties, including the calculation of the shortest connected spanning subgraph of a graph.

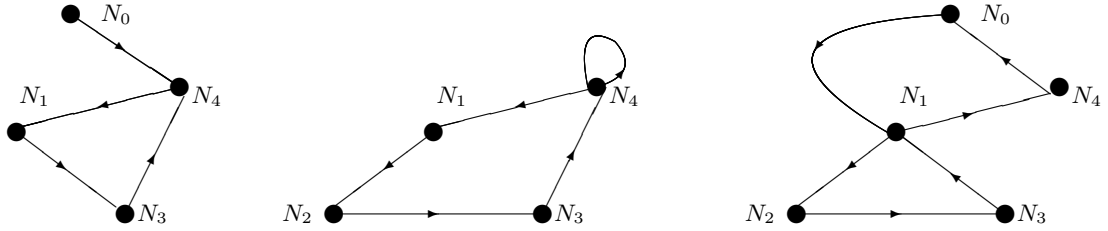
7.2.1 Definitions and examples

In many graphs it is possible to start at some vertex and follow a walk that eventually ends up back at the same vertex again. For example, in the following graph there is a walk N_4, N_1, N_3, N_4.

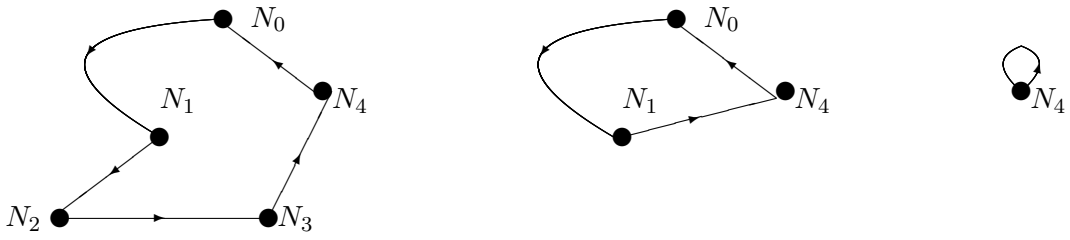


Definition 7.2.1 *A* loop *is an edge from a vertex to itself, and a* cycle *in a graph is a non-empty trail $N = N_0, N_1, \ldots, N_{k-1}, N_k = N$ from a vertex to itself where $N = N_0, N_1, \ldots, N_{k-1}$ is a path. (Non-empty just means that $k \geq 1$.)*

Example 7.2.2 *In the above graph, $N_0, N_4, N_1, N_3, N_4, N_0$ and $N_1, N_2, N_3, N_4, N_4, N_3, N_1$ and $N_0, N_1, N_2, N_3, N_4, N_0$ are not cycles,*



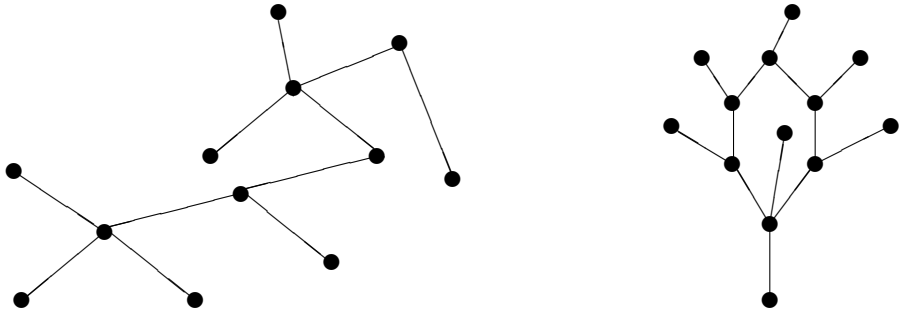
but $N_0, N_1, N_2, N_3, N_4, N_0$ *and* N_0, N_1, N_4, N_0 *and* N_4, N_4 *are cycles.*



A cycle that contains only one edge is a loop.

Definition 7.2.3 *A* tree *is a connected graph that has no cycles.*

The graph on the left below is a tree but the graph on the right is not, i.e. it's not the overall shape of the graph that matters.



Trees often occur as data structures in programs. In this module we just look at some basic properties of trees.

Although the first three examples above were not cycles, they contained cycles. It is always true that if there are two different trails between a pair of vertices then there is a cycle in the graph. This is a fundamental property of graphs:

Proposition 7.2.4 *If a graph has two vertices connected by two different trails then it has a cycle.*

The next proposition, which can be proved by induction on the number of edges in the graph, shows that in a connected graph the number of edges will dominate the number of vertices. So when considering how 'large' a graph might be for implementation purposes, if it is connected we need to consider the number of edges not the number of vertices.

Proposition 7.2.5 *Every connected graph with* n *vertices has at least* $n - 1$ *edges.*

We now give several equivalent conditions for a tree to be a graph. To prove that a graph is a tree it is sufficient to show that any one of the last four conditions in the following theorem holds.

Theorem 7.2.6 *Let G be a graph with n vertices. The following properties are equivalent.*

(1) G is a tree.

(2) There exists exactly one trail between any pair of vertices.

(3) G is connected and has exactly $n-1$ edges.

(4) G is connected and either $n=1$ and G has no edges, or $n \geq 2$ and removing any edge in G makes the graph unconnected.

(5) G has no cycles, and adding an edge between any pair of non-adjacent edges N, M creates a cycle containing N and M

The proofs that the other properties are equivalent to (1) are not given in these notes. Some of the techniques required for these proofs form the basis for algorithms in programs which manipulate trees, so it is worth looking up these proofs in one of the text books.

We first met graphs when we were looking for 'shortest road systems'. We can now prove that the shortest connected spanning subgraph of any connected graph will always be a tree. This allows us to make the algorithm for finding the shortest subsystem more efficient.

Theorem 7.2.7 *Let G be a connected graph. The shortest connected spanning subgraphs of G (the ones with the least number of edges) are trees. Furthermore, if G has n vertices then any shortest connected spanning subgraph has exactly $(n-1)$ edges.*

Proof *Let G' be a shortest spanning subgraph of a graph G with n nodes.*

$\boxed{n=1}$ *G' then consists of the single vertex of G and no edges, which is a tree by property (4) of Theorem 7.2.6 and has $0 = n-1$ edges.*

$\boxed{n \geq 2}$ *Since G' is a shortest connected subgraph, removing an edge from G' makes the graph unconnected. Thus G' has property (4) of Theorem 7.2.6, and so G is a tree. By property (3) of Theorem 7.2.6, G' has exactly $(n-1)$ edges.*

Finally, we return again to the Exmoor roads problem. There are 6 villages, so the graph of existing roads has 6 vertices. Theorem 7.2.7 shows that the minimum road system connecting the villages is a tree with 5 roads.

We next discuss an algorithm for calculating the shortest connected spanning subgraph of a graph.

7.2.2 Prim's algorithm

One of the algorithms that can be used to find a shortest connected spanning subgraph (tree) is due to Robert Prim. This algorithm works as follows:

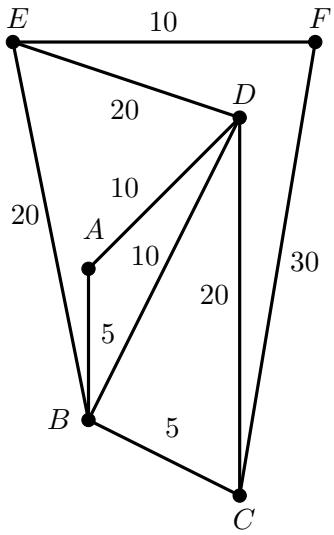
(1) Start at any node and mark it.

(2) Select the shortest non-marked edge that connects a marked node to an unmarked node, K say, and mark both that edge and K.

(3) Repeat step 2 until all nodes are marked.

The marked vertices and edges are a shortest spanning tree.

We illustrate the application of the algorithm to the graph of the Exmoor villages.



The application of Prim's algorithm requires that we mark nodes and edges at every step. In order to document the application of the algorithm, we will use a table with three columns labelled Step, Marked edges, and Marked nodes.

We start by choosing a node, say C. It doesn't matter which node we choose to start at; we may obtain different trees but they will all be shortest.

Step	Marked edges	Marked nodes
1	–	C

Next we need to select the shortest edge that connects C to an unmarked node: that is CB with weight 5, so we mark both CB and C.

Step	Marked edges	Marked nodes
1	–	C
2	CB	C, B

Next we need to select the shortest unmarked edge that connects either C or B to an unmarked node: that is BA with weight 5. So we mark both BA and A.

Step	Marked edges	Marked nodes
1	–	C
2	CB	C, B
3	CB, BA	C, B, A

Next we need to select the shortest unmarked edge that connects either C or B or A to an unmarked node: we have a choice between AD and BD, both with weight 10. Say we choose AD (which one we choose does not matter), so we mark both AD and D.

Step	Marked edges	Marked nodes
1	–	C
2	CB	C, B
3	CB, BA	C, B, A
4	CB, BA, AD	C, B, A, D

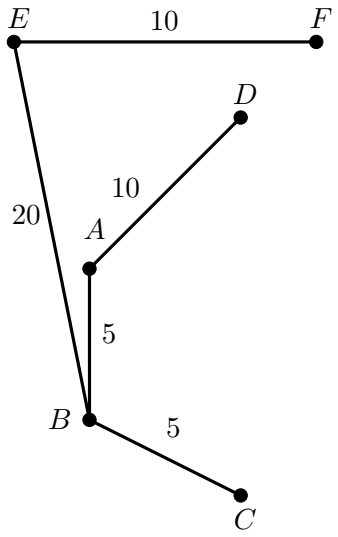
In the next step, we have a choice between BE and DE. Say we choose BE

Step	Marked edges	Marked nodes
1	–	C
2	CB	C, B
3	CB, BA	C, B, A
4	CB, BA, AD	C, B, A, D
5	CB, BA, AD, BE	C, B, A, D, E

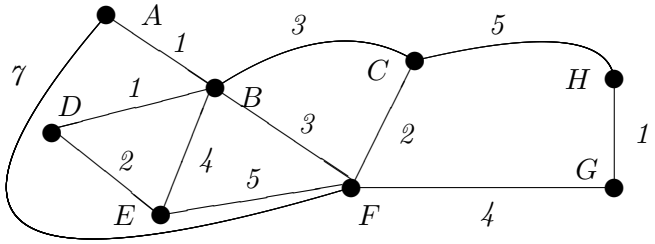
In the next step, the shortest is EF.

Step	Marked edges	Marked nodes
1	–	C
2	CB	C, B
3	CB, BA	C, B, A
4	CB, BA, AD	C, B, A, D
5	CB, BA, AD, BE	C, B, A, D, E
6	CB, BA, AD, BE, BF	C, B, A, D, E, F

All nodes have now been marked, so we stop. Notice that the tree that we obtain has indeed $5 = 6 - 1$ edges.



Example 7.2.8 *Find the shortest connected spanning subgraph of the following graph.*



Solution We start by choosing a node, say F.

Step	Marked edges	Marked nodes
1	–	F

Next we need to select the shortest edge that connects F: that is FC with weight 2, so we mark both FC and C.

Step	Marked edges	Marked nodes
1	–	F
2	FC	F, C

Next we need to select the shortest unmarked edge that connects either F or C to an unmarked node: we have a choice between FB and CB, both with weight 3. It does not matter which one we choose, say CB. So we mark both CB and B.

Step	Marked edges	Marked nodes
1	–	F
2	FC	F, C
3	FC, CB	F, C, B

Next we need to select the shortest unmarked edge that connects either F or C or B to an unmarked node: again we have a choice, this time between BA and BD, both with weight 1. Say we choose BA, so we mark both BA and A.

Step	Marked edges	Marked nodes
1	–	F
2	FC	F, C
3	FC, CB	F, C, B
4	FC, CB, BA	F, C, B, A

In the next step, the shortest is BD.

Step	Marked edges	Marked nodes
1	–	F
2	FC	F, C
3	FC, CB	F, C, B
4	FC, CB, BA	F, C, B, A
5	FC, CB, BA, BD	F, C, B, A, D

In the next step, the shortest is DE.

Step	Marked edges	Marked nodes
1	–	F
2	FC	F, C
3	FC, CB	F, C, B
4	FC, CB, BA	F, C, B, A
5	FC, CB, BA, BD	F, C, B, A, D
6	FC, CB, BA, BD, DE	F, C, B, A, D, E

In the next step, the shortest is FG.

Step	Marked edges	Marked nodes
1	–	F
2	FC	F, C
3	FC, CB	F, C, B
4	FC, CB, BA	F, C, B, A
5	FC, CB, BA, BD	F, C, B, A, D
6	FC, CB, BA, BD, DE	F, C, B, A, D, E
7	FC, CB, BA, BD, DE, FG	F, C, B, A, D, E, G

Only H remains unmarked and the shortest edge that connects it is GH.

Step	Marked edges	Marked nodes
1	–	F
2	FC	F, C
3	FC, CB	F, C, B
4	FC, CB, BA	F, C, B, A
5	FC, CB, BA, BD	F, C, B, A, D
6	FC, CB, BA, BD, DE	F, C, B, A, D, E
7	FC, CB, BA, BD, DE, FG	F, C, B, A, D, E, G
8	$FC, CB, BA, BD, DE, FG, GH$	F, C, B, A, D, E, G, H

All nodes have now been marked, so we stop. Notice that the tree that we obtain has indeed $7 = 8 - 1$ edges.

7.3 Directed graphs and rooted trees

As we have already mentioned, searching data is an important part of computing, and it can be very time consuming. Thus it is necessary to take steps to make searches efficient. One method of organising data so that it can be searched more efficiently is to use a *rooted tree*.

For example, many 'family' medical books have a section which helps to diagnose illnesses from given symptoms. There is an initial question, then subsequent questions which depend on the answer to the first question. This is illustrated in Figure 7.1.

An alternative is, for each combination of symptoms, to list the diagnosis. Then search the list until the matching symptoms are found. The list will have 8 entries, thus the longest search will involve 8 items. With the tree method the maximum search has length 4 (the longest path in the tree).

The tree method of storage makes searching more efficient, and this efficiency is greater when the tree is bigger. The increased efficiency is illustrated by the following party game, the decision tree for which is shown in Figure 7.2.

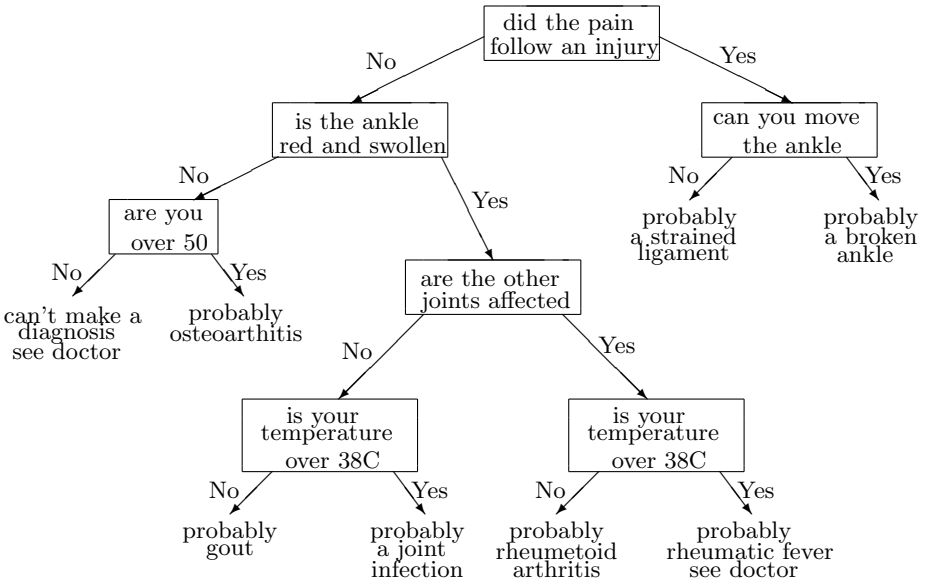


Figure 7.1: Diagnosis tree

> One person asks another to think of a number n between 1 and 31. The first person then proceeds to try to guess n. The second person is required to respond to each guess with either 'Yes', 'smaller', or 'larger', according to whether n is equal to, smaller than or larger than the number guessed. The first person guarantees to guess n in at most 5 guesses.

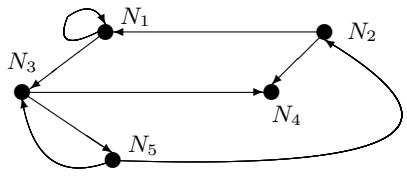
This uses a maximum of 5 tests instead of a possible 31.

Definition 7.3.1 *A tree which has a distinguished start vertex (called the* root*) such that there is a path from the root to any other vertex is called a* rooted tree*.*

Rooted trees really belong to the family of directed graphs. We have already met directed graphs as a method of representing relations on finite sets.

Definition 7.3.2 *A* directed graph *or* digraph *is a set of vertices and a set of arrows (directed edges) between pairs of vertices.*

For example,



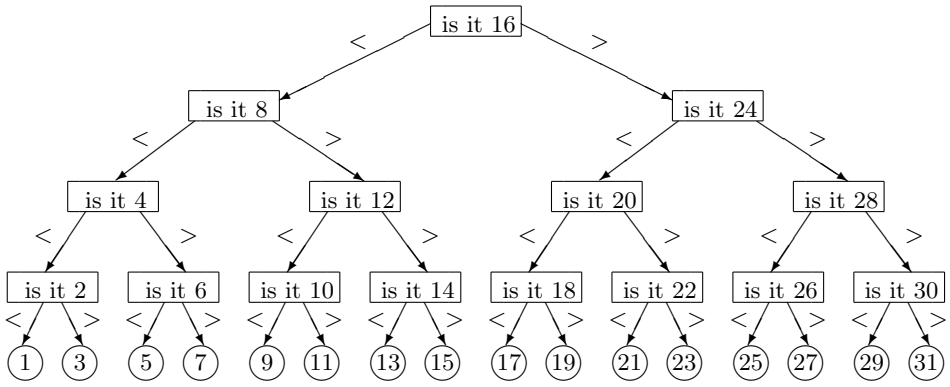


Figure 7.2: Guess-a-number tree

Some of the definitions given in Section 7.1 need to be adapted for digraphs.

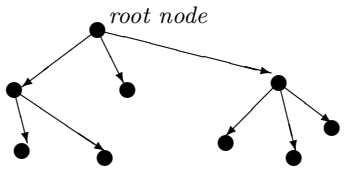
Definition 7.3.3 *Given a digraph:*

- *A vertex N is* adjacent *to a vertex M if there is an arrow from N to M. We call N an* antecedent *of M, and we call M a* successor *of N.*
- *A* walk *in a digraph is a sequence $N = N_0, N_1, N_2, \ldots, N_{k-1}, N_k = M$, where $k \geq 0$ and for $0 \leq i \leq (k-1)$, N_i is adjacent to N_{i+1}.*
- *A* trail *in a digraph is a walk that has no repeated edges and a* path *in a digraph is a walk with no repeated vertices.*
- *A* cycle *is a walk $N_0, N_1, N_2, \ldots, N_{k-1}, N_k, N_0$ such that $N_0, N_1, N_2, \ldots, N_{k-1}, N_k$ is a path.*

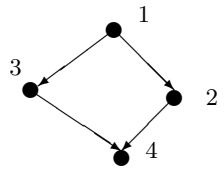
For example, in the above digraph, N_2, N_1, N_3, N_5, N_2 and N_3, N_5, N_3 are cycles.

Definition 7.3.4 *The* underlying graph *of a digraph is the graph obtained by replacing arrows with undirected edges. A digraph is* connected *if its underlying graph is connected.*

Definition 7.3.5 *A* rooted tree *is a digraph in which there is one vertex (called the* root vertex*) that has a path to any other vertex and in which each vertex has at most one antecedent (called its* parent*).*



Note that a digraph without cycles is not necessarily a rooted tree. For example, the digraph below has no cycles (although the underlying graph does) but it is not a rooted tree because 4 has two parents: 3 and 2.



7.4 Eulerian paths

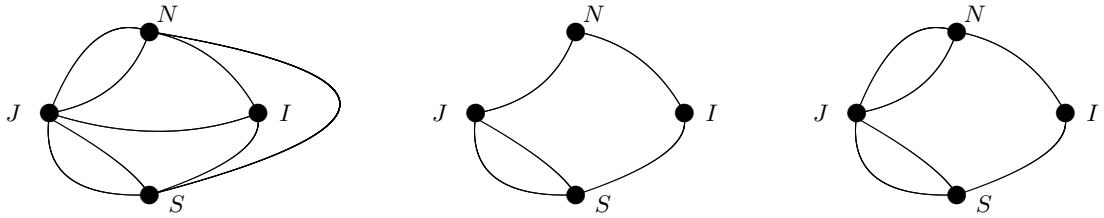
Recall the challenge that was set at the beginning of this chapter: is there a path in the Königsberg bridges graph which uses each edge exactly once?

Definition 7.4.1 *A trail in a graph that uses all the edges exactly once is called an* Eulerian path.

It is not possible to meet the above challenge; no Eulerian path exists. How do we know this?

There are only 7 edges in the graph. Thus, in theory it is possible to write all the possible trails down, and demonstrate that none of them includes all the edges. But there are at least 60 such trails, and how do we know that we haven't missed any?

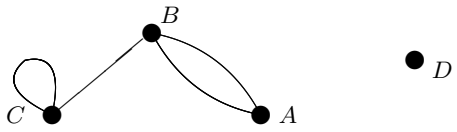
Notice that, if a bridge is removed or if a new one is added then it is possible to complete the challenge: the three graphs below have Eulerian paths.



It turns out that the existence of an Eulerian path depends only on the number of vertices which have an odd number of attached edges.

Definition 7.4.2 *The number of edges attached to a vertex is called the* degree *of the vertex.*

In the following graph vertex A has degree 2, vertices B and C have degree 3, and vertex D has degree 0.

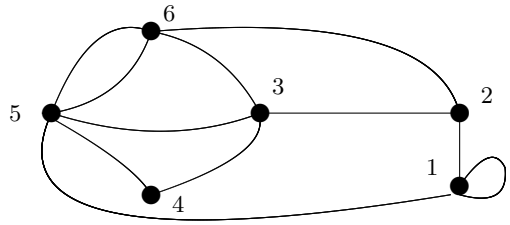


The Königsberg bridges graph has exactly 4 vertices whose degree is odd: 3 vertices of degree 3 and one of degree 5. The following theorem, whose proof is not covered in this module, shows that the Königsberg graph does not contain an Eulerian path.

Theorem 7.4.3 *A connected graph G has an Eulerian path if and only if either all its vertices have even degree, or exactly two of its vertices have odd degree.*

7.5 Exercises

1. Write down an Eulerian path in the following (multi)graph.



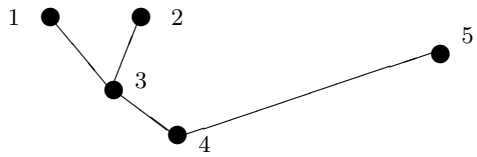
2. For the graph in Question 1, write down

   (a) The vertices adjacent to 6.

   (b) The vertices adjacent to 1.

   (c) A walk from 5 to 3 that is *not* a trail.

3. Consider the following tree, T:



   Write down all the graphs which can be constructed by adding one new edge between any two non-adjacent vertices in T. Thus verify that (5) of Theorem 7.2.6 holds for this graph.

4. Consider the following relations on $\mathbb{S}_4$.

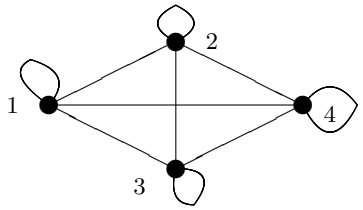
$$R_1 = \{(1,1),(1,2),(1,3),(2,4)\}$$

$$R_2 = \{(1,2),(2,3),(2,4),(3,4)\}$$

$$R_3 = \{(2,1),(2,3),(1,4)\}$$

Decide whether the graphical representation of each relation is a rooted tree, and if it is give the root vertex.

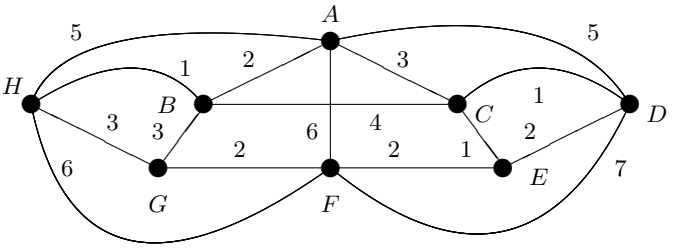
5. A graph in which every vertex is adjacent to all the vertices in the graph is called a *complete* graph. The following is the complete graph with four vertices.



Write down all the spanning subtrees of the complete graph with four vertices.

6. Write down all the possible graphs that have four vertices and 0, 1, or 2 edges and no loops. Hence verify that Lemma 3 holds for graphs with four vertices.

7. Consider the following road network.



Find the shortest road network which connects all the towns, and which uses only existing roads. (Justify your answer.)

8. The following is a chart which allows a person to select clothing.

temperature	raining	windy	wear
hot	no	no	shirt
warm	no	no	jumper
warm	yes	yes	shirt, kagool
cold	yes	no	overcoat, umbrella
hot	yes	no	shirt, umbrella
hot	no	yes	shirt
cold	no	no	overcoat
warm	no	yes	jumper, umbrella
cold	no	yes	overcoat
hot	yes	yes	shirt, kagool
cold	yes	yes	jumper, kagool
warm	no	yes	jumper

Draw a decision tree (a rooted tree) which corresponds to the above chart. What is the root of your tree?

Construct a different rooted tree which also corresponds to the above chart. What is the root of this tree?

Chapter 8

Probability

Probability theory is used in many areas of Computer Science. For example, whenever several applications have to share resources it is necessary to calculate the probable usage requirements of each application to decide how best to assign the resources.

Probability calculations are used extensively in network reliability analysis. A rather popular model for studying computer/communication network reliability is a graph with nodes and/or edges with an associated probability of failure.

More generally, in safety-critical areas where software failure may cause injury or loss of life, it is necessary to calculate the probability that a fault will occur. This is then compared with the level of damage that would be caused by the fault to give a measure of the 'safety' of the system.

Many expert systems, as used for example in medical diagnosis or weather forecasting, take a large set of actual case histories and use this to construct a decision algorithm. The algorithm is then used to decide the most likely diagnosis/weather prediction, based on the given symptoms/climatic conditions. The decision algorithm is constructed using probability theory.

Artificial Intelligence systems based on neural networks 'learn' using probabilistic reasoning; for example, the cost of insurance is determined by probability calculations that predict the likelihood of a claim.

(*Most of the material in this chapter can be found in Chapter 7 of Rosen.*)

8.1 Elementary probability

Probability is concerned with events that are assumed to be random. If a coin is tossed and required to spin several times before landing, experience shows that we cannot predict which side will face upwards when it lands. We say a coin is *fair* if each face has the same

properties, so the coin is not weighted or bent. We assume that it is not possible for a person to toss a coin, which has to spin several times, in a way that guarantees that it will always lie 'head' side up. Experience supports this assumption.

8.1.1 Sample spaces and probability functions

Definition 8.1.1 *A process is* random *if it is not possible to predict the outcome of the process in advance. An* event *is a possible outcome of a process (experiment). The set of all possible outcomes of an experiment (all possible events) is called the* sample space *of the experiment.*

For example, 'heads' is an event that is a possible outcome of the process of tossing a coin. The sample space for tossing a coin (once) is $\{heads, tails\}$.

Another familiar example is the process of rolling a fair die. We assume that the die is not 'weighted' and that it must roll several times before stopping. Experience shows that the number that will appear on the top face of the die is unpredictable. We assume that it is a random process. The sample space associated with the 'rolling a die' experiment is the set $\{1, 2, 3, 4, 5, 6\}$, the values on each of the faces of the die.

In both of the above examples each of the events in the sample space is 'equally likely'. The 'chances' of rolling a 1 are the same as the chances of rolling a 2, etc. It is possible to construct random experiments whose outcomes have different probabilities. For example, suppose we toss a fair coin twice and record the number of heads and the number of tails. There are 4 possibilities:

$$\text{HH} \qquad \text{HT} \qquad \text{TH} \qquad \text{TT}$$

The chances of two heads are 1 in 4, the chances of two tails are 1 in 4, but the chances of one of each are 2 in 4.

Suppose we roll two identical fair dice A and B and add together the total on each; then the sample space is $\{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$. It is not true that each of these events is equally likely. Let us record the values on each die as an ordered pair, so that $(3, 1)$ means 3 on die A and 1 on die B. There is only one way to get 2, by throwing two 1s i.e. $(1, 1)$. However, there are six ways of getting 7,

$$(1, 6) \quad (2, 5) \quad (3, 4) \quad (4, 3) \quad (5, 2) \quad (6, 1)$$

There are 36 possible results, 6 possible values on die A, and 6 on die B. The probability of scoring 2 with the dice is 1 in 36, and the probability of scoring 7 is 6 in 36, or 1 in 6.

Definition 8.1.2 *Given a sample space $S = \{s_1, s_2, \ldots, s_n\}$, a* probability function *is a function $p : S \to [0, 1] = \{r \in \mathbb{R} \mid 0 \leq r \leq 1\}$ such that*

$$p(s_1) + p(s_2) + \ldots + p(s_n) = 1.$$

Example 8.1.3 *The following are examples of probability functions:*

(a) When tossing a coin and observing 'heads' or 'tails' we have $S = \{H, T\}$ *and* $p : S \to [0, 1]$ *is given by*

$$p(H) = 1/2, \quad p(T) = 1/2$$

Notice that $p(H) + p(T) = 1$.

(b) For rolling a die and observing the number on the face showing up we have $S = \{1, 2, 3, 4, 5, 6\}$ *and* $p : S \to [0, 1]$ *is given by*

$$p(1) = p(2) = p(3) = p(4) = p(5) = p(6) = 1/6$$

Notice that $p(1) + \ldots + p(6) = 1$.

(c) When tossing a coin twice and recording the number of heads and the number of tails we have $S = \{2H, 2T, 1H1T\}$ *and* $p : S \to [0, 1]$ *is given by*

$$p(2H) = 1/4, \quad p(2T) = 1/4, \quad p(1H1T) = 1/2$$

Notice that $p(2H) + p(2T) + p(1H1T) = 1$.

(d) For rolling two dice and calculating the sum of the two values showing up we have $S = \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$ *and* $p : S \to [0, 1]$ *is given by*

$$p(2) = 1/36,\ p(3) = 2/36,\ p(4) = 3/36,\ p(5) = 4/36,\ p(6) = 5/36,\ p(7) = 6/36$$
$$p(8) = 5/36,\ p(9) = 4/36,\ p(10) = 3/36,\ p(11) = 2/36,\ p(12) = 1/36.$$

Notice that $p(1) + \ldots + p(12) = 1$.

8.1.2 Compound events

Sometimes we want to know the probability that an event has a particular property. For example, if we roll a die what is the probability that the number will be odd? This is the same as asking for the probability that the event is in the set $\{1, 3, 5\}$.

Definition 8.1.4 *Any subset of a sample space is called a* compound event. *The probability of a compound event is the sum of the probabilities of the individual events.*

For example, the probability of rolling an odd number is the probability of the compound event $\{1, 3, 5\}$:

$$p(odd) = p(\{1, 3, 5\}) = p(1) + p(3) + p(5) = 1/6 + 1/6 + 1/6 = 1/2.$$

Generally, if $\{a_1, a_2, \ldots, a_k\} = A \subseteq S$ then

$$p(A) = p(a_1) + p(a_2) + \ldots + p(a_k)$$

The whole sample space S is a compound event, called the *certain* event; it is certain that any event that actually occurs will be in S. We have $p(S) = 1$.

The empty set $\emptyset$ is also a compound event, called the *impossible* event; no event may occur in $\emptyset$. Therefore we have $p(\emptyset) = 0$.

The probability that an event (single or compound) will *not* occur, is the probability of the complement event in S. For example, the probability of not rolling a 3 in one die roll is

$$p(S\backslash\{3\}) = p(\{1, 2, 4, 5, 6\}) = 5/6$$

and the probability of *not* rolling an odd number is

$$p(S\backslash\{1, 3, 5\}) = p(\{2, 4, 6\}) = 1/2.$$

Theorem 8.1.5

(1) If A is a compound event in the probability space S then

$$p(S\backslash A) \;=\; 1 - p(A).$$

(2) If B is also a compound event then

$$p(A \cup B) \;=\; p(A) + p(B) - p(A \cap B).$$

Example 8.1.6 *The (compound) event that 2 dice give a total score that is a multiple of 6 is $A = \{6, 12\}$. The event that 2 dice give a total score that is a multiple of 4 is $B = \{4, 8, 12\}$. Then we have*

$$\begin{aligned} p(A) &= 5/36 + 1/36 = 1/6 \\ p(B) &= 3/36 + 5/36 + 1/36 = 1/4 \\ p(A \cap B) &= p(\{12\}) = 1/36 \\ p(A \cup B) &= p(\{4, 6, 8, 12\}) = 7/18. \end{aligned}$$

$A \cup B$ is the event that the score is a multiple of 4 or 6, and $A \cap B$ is the event that the score is a multiple of 4 and a multiple of 6. We can verify that as stated in Theorem 8.1.5,

$$p(A \cup B) = p(A) + p(B) - p(A \cap B) = 1/6 + 1/4 - 1/36 = 7/18$$

We can also understand why we need to subtract $p(A \cap B)$, otherwise the event 12 would be counted twice.

Example 8.1.7 *A pack of 52 playing cards is shuffled in to random order, and then one card is selected at random. What is the probability that this card is either a spade, an ace or a two?*

Solution Let A be the set of 13 spades, and let B be the set of 4 aces and 4 twos. The event that the selected card is either a spade, an ace or a two is the set $A \cup B$. Thus we want $p(A \cup B)$. Since $A \cap B = \{AceSpades, TwoSspades\}$ we can use Theorem 8.1.5 to get

$$p(A \cup B) = p(A) + p(B) - p(A \cap B) = 13/52 + 8/52 + 2/52 = 19/52.$$

8.2 Conditional probability

So far we have assumed that we cannot predict anything about the events which may occur. However, sometimes we have additional information that we can rely on, which changes the probabilities of the events.

For example, suppose that someone rolls a die, looks at the result, and tells everyone that the number is odd. This is the same as saying that the event which has occurred belongs to $B = \{1, 3, 5\}$. Knowing that the number is odd, we know that the probabilities of the event being 2, 4, or 6 are 0. The remaining numbers — 1, 3 and 5 — are still equally likely, but their probability knowing B is now $1/3$.

Definition 8.2.1 *When we have additional information that an event must be in a set B, the probability of an event is called the* conditional probability *given B. The conditional probability is written $p(s|B)$.*

So, for the case above, we have $p(1|B) = p(3|B) = p(5|B) = 1/3$. We also have $p(B|B) = 1$ and $p(\{2, 4, 6\}) = p((S \backslash B)|B) = 0$.

Notice that, still for the case above, we have

$$p(B) = p(1) + p(3) + p(5) = 1/6 + 1/6 + 1/6 = 1/2$$

and

$$p(1|B) = 1/3 = \frac{(1/6)}{(1/2)} = \frac{p(1)}{p(B)}$$

(and the same for $p(3|B)$ and $p(5|B)$). More generally, given a sample space S and $B = \{s_1, s_2, \ldots, s_k\}$, then for all $1 \leq i \leq k$ we have

$$p(s_i|B) = \frac{p(s_i)}{p(B)}$$

Notice that, as expected,

$$\frac{p(s_1)}{p(B)} + \ldots + \frac{p(s_k)}{p(B)} = \frac{p(s_1) + \ldots + p(s_k)}{p(B)} = \frac{p(B)}{p(B)} = 1$$

Example 8.2.2 *Suppose that a coin is tossed twice and you are told that there is at least one head. What is the probability that there is one head and one tail?*

Solution We have $S = \{2H, 2T, 1H1T\}$ and $B = \{2H, 1H1T\}$ which corresponds to HH, HT, TH, so $p(B) = 3/4$. We get 1H1T if either HT or TH occurs, so $p(1H1T) = 2/4$. Therefore, $p(1H1T \mid B) = p(1H1T)/p(B) = 2/3$.

Returning to our general case above, we note that if $s \in S$ but $s \notin B$, then $p(s|B) = 0$. This means that, if we now consider the general case of a compound event $A \subseteq S$, to calculate $p(A|B)$ we only need to consider the events of A that belong to B, i.e. the events in $A \cap B$.

Theorem 8.2.3 *Given a sample space S and two compound events $A, B \subseteq S$,*

$$p(A|B) = \frac{p(A \cap B)}{p(B)}.$$

For example, if we throw two dice, and if we are told that the total score is odd, what is the probability that the score is prime?

We have $B = \{3, 5, 7, 9, 11\}$ and we want to know the probability that the event is in $A = \{2, 3, 5, 7, 11\}$. Then A can only happen if the score is 3, 5, 7, or 11, i.e. if $A \cap B$ occurs. We have $p(B) = p(3) + p(5) + p(7) + p(9) + p(11) = 18/36$ and $p(A \cap B) = p(3) + p(5) + p(7) + p(11) = 14/36$. Therefore,

$$p(A|B) = \frac{p(A \cap B)}{p(B)} = \frac{14/36}{18/36} = \frac{14}{18} = \frac{7}{9}.$$

Theorem 8.2.3 can also be written as

$$p(A \cap B) = p(B)p(A|B) = p(A)p(B|A)$$

which can be very useful when having to calculate the probability of intersections of compound events. For example, if we have three compound events, A, B, and C, we get

$$p(A \cap B \cap C) = p(A \cap B)p(C|(A \cap B)) = p(A)p(B|A)p(C|A \cap B)$$

Example 8.2.4 *Three cards are drawn, without replacement, from a standard pack of playing cards. What is the probability that all three cards are aces?*

Solution Let A be the event that the first card is an ace, let B be the event that the second card is an ace, and let C be the event that the third card is an ace. We need $p(A \cap B \cap C)$.

We have $p(A) = 4/52$. If we know that A is true then there are 3 aces left, and 51 cards from which to choose the second ace; thus we have $p(B|A) = 3/51$. If we know that $A \cap B$ is true then there are 2 aces left, and 50 cards from which to choose the third ace; thus we have $p(C|A \cap B) = 2/50$. Therefore,

$$p(A \cap B \cap C) = p(A)p(B|A)p(C|A \cap B) = 4/52 \times 3/51 \times 2/50 = 1/5525.$$

In general, we have

$$p(A_1 \cap A_2 \cap \ldots \cap A_k) = p(A_1)p(A_2|A_1)p(A_3|A_1 \cap A_2) \ldots p(A_k|A_1 \cap A_2 \cap \ldots \cap A_{k-1}).$$

8.2.1 Independent events

When we select two cards from a pack, the selection we make first affects the probability of the card that will be selected second. If we select a heart, then the probability of selecting heart the second time is slightly reduced. This dependence does not always exist. If we select a card, replace it, shuffle the cards and select a second card, then the probabilities remain the same. If we selected a heart the first time, there is still a 1 in 13 probability of selecting a heart the second time.

If we roll a die and get 6, the probability that we will get 6 again the next time we roll is still 1/6. Furthermore, if we toss a coin 20 times and get all heads, the probability of getting heads on the 21st toss is still 1/2 (provided that the coin is 'fair').

Definition 8.2.5 *We say that two events A, B are independent if $p(A \cap B) = p(A)p(B)$.*

From the formula for conditional probabilities we see that this is the same as saying

$$p(A|B) = p(A)$$

8.3 Bayes' Theorem

If two sets S_1 and S_2 are a partition of the sample space S then we have $S_1 \cup S_2 = S$ and $S_1 \cap S_2 = \emptyset$. This means that, for any compound event $A \subseteq S$, we have

$$A = S \cap A = (S_1 \cup S_2) \cap A = (S_1 \cap A) \cup (S_2 \cap A)$$

$$(S_1 \cap A) \cap (S_2 \cap A) = (S_1 \cap S_2) \cap A = \emptyset \cap A = \emptyset$$

Therefore, using the rule for calculating the probability of unions, we get

$$p(A) = p(S_1 \cap A) + p(S_2 \cap A).$$

In general, if $S_1, S_2, \ldots, S_k$ is a partition of S, and if A is a compound event, then

$$p(A) = p(S_1 \cap A) + p(S_2 \cap A) + \ldots + p(S_k \cap A).$$

Using now Theorem 8.2.3 we obtain:

Lemma 8.3.1 *If $S_1, S_2, \ldots, S_k$ are a partition of S and A is a compound event, then*

$$p(A) = p(A|S_1)p(S_1) + p(A|S_2)p(S_2) + \ldots + p(A|S_n)p(S_n).$$

That is, we can calculate $p(A)$ by determining the conditional probabilities $p(A|S_i)$, which is useful when those conditional probabilities are given, or easy to obtain.

Example 8.3.2 *Imagine that there is a rare illness that affects 1 in 10,000 people. Suppose that we have a medical scanner that detects the illness in 99.9% of people who have it, but which also incorrectly reports that 0.1% of non-suffers have the illness. If you have been diagnosed by the scanner as having the illness what is the probability that you actually* do *have it?*

Solution First we establish some notation for the possible events.

I = the event that a person has the illness = the set of people with the illness

H = the event that a person does not have the illness = the set of 'healthy' people

D = the event that the scanner diagnoses the illness in a person

We are asked to determine $p(I|D)$.

From the information that we are given we have

$p(I) = 1/10000$

$p(D|I) = 999/1000$

$p(D|H) = 1/1000$

Because I, H form a partition of the sample space (set of all people) we also have that

$$p(H) = 1 - p(I) = 9999/10000$$

From Lemma 8.3.1 we can conclude

$$p(D) = p(D|I)p(I) + p(D|H)p(H).$$

Because by Theorem 8.2.3 $p(I|D) = p(I \cap D)/p(D)$, it remains to calculate $p(I \cap D)$. Using Theorem 8.2.3 once again, we know that $p(I \cap D) = p(D|I)p(I)$, which is useful because we know $p(D|I)$.

In summary, we arrive at

$$p(I|D) = \frac{p(D|I)p(I)}{p(D|I)p(I) + p(D|H)p(H)}$$

and we know all the quantities on the right-hand side. We can therefore calculate:

$$p(I|D) = \frac{\frac{999}{1000} \times \frac{1}{10000}}{\frac{999}{1000} \times \frac{1}{10000} + \frac{1}{1000} \times \frac{9999}{10000}} = \frac{999}{10998} < 9.1\%$$

Thus we have that the probability of having the illness given a positive diagnosis by the scanner is 999 in 10998 is less than 9%!

The method that we used has a special name: *Bayes' Theorem.*

Theorem 8.3.3 *If $S_1, S_2, \ldots, S_k$ is a partition of a sample space S and $B \subseteq S$ is a compound event, then*

$$p(S_i|B) = \frac{p(B|S_i)p(S_i)}{p(B|S_1)p(S_1) + p(B|S_2)p(S_2) + \ldots + p(B|S_k)p(S_k)}$$

for all $i = 1, 2, \ldots, k$.

Example 8.3.4 *Assume that 25% of babies born in this country have mothers that smoked during pregnancy. Suppose it is known that 60% of the babies born whose mothers smoke weigh less than 6lbs. Suppose that it is also known that 20% of babies whose mothers do not smoke weigh less than 6lbs at birth. If a baby is picked at random and found to have birth weight less than 6lbs, use Bayes' Theorem to calculate the probability that the baby's mother smokes.*

Solution As in the previous example, we first establish some notation for the possible events.

B = the event that a baby weighs less than 6lbs

M = the event that a baby's mother smokes

N = the event that a baby's mother does not smoke

We are asked to determine $p(M|B)$.

We are given that $p(M) = 25/100 = 0.25$, $p(B|M) = 60/100 = 0.6$, and $p(B|N) = 20/100 = 0.2$. Because M, N partition the sample space we have $p(N) = 1 - p(M) = 0.75$ and we can use Bayes' Theorem to conclude

$$p(M|B) = \frac{p(B|M)p(M)}{p(B|M)p(M) + p(B|N)p(N)} = \frac{0.6 \times 0.25}{0.6 \times 0.25 + 0.2 \times 0.75} = \frac{0.15}{0.30} = 0.5.$$

Thus the probability of a low weight baby having a mother who smokes is 50%.

8.4 Exercises

1. The 13 hearts from a standard pack of cards are shuffled in to a random order, then one card is selected. What is the probability that this card is the ace of hearts?

2. Three dice are rolled and their total score is noted. Write down the sample space S of this experiment. What is $p(3)$? What is $p(\{3, 18\})$? What is $p(S \backslash \{3, 18\})$?

3. A die is thrown and you are told that the number is a multiple of 3. What is the probability that the number *is* 3?

4. 4 cards are drawn without replacement from a standard pack. What is the probability that they are a mix of Kings and Queens?

5. Let A, B, C be three independent compound events, with $A \cup B \cup C = S$. If $E \subseteq S$, write down the probability of A given E in terms of $p(E|A)$, $p(E|B)$, $p(E|C)$, $p(A)$, $p(B)$, and $p(C)$.

6. A coin is tossed three times. Find the probability that the number of heads is 1 and the number of tails is 2.

7. Let $S = \{s_1, s_2, s_3, s_4, s_5\}$ be a sample space, and suppose that $p : S \to [0, 1]$ is given by

$$\begin{array}{lll} p(s_1) = 1/9, & p(s_3) = 1/9, & p(s_5) = 1/9 \\ p(s_2) = 4/9, & p(s_4) = 2/9. & \end{array}$$

   If $A = \{s_1, s_2, s_4\}$, $B = \{s_3, s_5\}$, find $p(A)$, $p(S \backslash A)$, $p(A \cup B)$.

8. It is known that 50% of the students doing Maths A-level are also doing Physics A-level. It is also known that 25% of all students doing A-level are taking Maths as one of their A-levels, and 10% of all students doing A-levels are taking Physics. If an A-level student is picked at random and found to be taking Physics A-level, use Bayes' Theorem to calculate the probability that student is also taking Maths A-level.

Chapter 9

Statistical distributions

In this chapter we build on notions of probability discussed in Chapter 8 and look at what happens when the same experiment is repeated many times. The sort of questions we consider are: if an experiment is repeated several times how often is a certain event likely to occur?

(*Most of the material in this chapter can be found in Chapter 7 of Rosen.*)

9.1 Random variables

Suppose that we roll two dice four times and count the number of times that we throw a 'double', i.e. that both dice have the same value. In four throws we can get 0, 1, 2, 3, or 4 doubles. What is the likelihood of each possibility?

Let D be the event that we throw a double, and let N be the event that the result is not a double. Let S be the sample space that consists of all the possible outcomes of four throws. It is easy to see that S contains 16 events; the event that all four throws are doubles, D, D, D, D, the event that the first three throws are doubles but the last is not D, D, D, N, etc. More precisely, S consists of the following events:

D,D,D,D	D,D,D,N	D,D,N,D	D,N,D,D	N,D,D,D	D,D,N,N
D,N,D,N	N,D,D,N	D,N,N,D	N,D,N,D	N,N,D,D	D,N,N,N
N,D,N,N	N,N,D,N	N,N,N,D	N,N,N,N		

We can define a function $X : S \to \mathbb{R}$ by setting $X(s) =$ the number of doubles in s. So $X(D, D, D, D) = 4$, $X(N, D, N, D) = 2$, and so on. Such a function is called a *random variable.*

Definition 9.1.1 *Let S be a sample space. A random variable is a function $X : S \to \mathbb{R}$.*

The probability that the random variable has a value r is denoted $p(X = r)$, that is

$$p(X = r) = p(\{s \in S \mid X(s) = r\}).$$

Therefore, if $\{s \in S \mid X(s) = r\} = \{s_1, s_2, \ldots, s_k\}$, i.e. $s_1, s_2, \ldots, s_k$ are exactly the events for which $X(s) = r$, then $p(X = r) = p(s_1) + p(s_2) + \ldots + p(s_k)$.

The probability that a random variable takes a given value is always a number between 0 and 1, thus we can define a function $f_X : R \to [0, 1]$ by

$$f_X(r) = p(X = r).$$

Moreover, the sum of all of the $f_X(r)$ is 1. This is because X is a function, i.e. no element of the sample space can be assigned more than one value by X and X assigns a value to all elements of the sample space.

Proposition 9.1.2 *The function f_X defined by $f_X(r) = p(X = r)$ is a probability function on $\mathbb{R}$ (see Section 8.1). It is called the* probability distribution function *for X.*

Example 9.1.3 *In 8.1.3 we can find several examples of random variables and their probability functions.*

(b) When rolling a die and observing the number on the face showing up we have $S = \{1, 2, 3, 4, 5, 6\}$, $X(s) = s$ (we observe a number), and $f_X : \mathbb{R} \to [0, 1]$ is given by

$$p(X = 1) = p(X = 2) = p(X = 3) = p(X = 4) = p(X = 5) = p(X = 6) = 1/6$$

For all other values of $r \in \mathbb{R}$, $f_X(r) = 0$.

(c) When tossing a fair coin twice, we have $S = \{HH, HT, TH, TT\}$ and X counts the number of 'heads', i.e. $X(HH) = 2$, $X(HT) = X(TH) = 1$ and $X(TT) = 0$. The probability distribution function $f_X : \mathbb{R} \to [0, 1]$ is given by

$$p(X = 2) = 1/4, \quad p(X = 1) = 1/2, \quad p(X = 0) = 1/4$$

For all other values of $r \in \mathbb{R}$, $f_X(r) = 0$.

(d) When rolling two fair dice, X counts the sum of the two values showing up. The probability distribution function $f_X : \mathbb{R} \to [0, 1]$ is given by

$$\begin{array}{llll} p(X = 2) = 1/36, & p(X = 3) = 2/36, & p(X = 4) = 3/36, & p(X = 5) = 4/36, \\ p(X = 6) = 5/36, & pX = (7) = 6/36, & p(X = 8) = 5/36, & p(X = 9) = 4/36, \\ p(X = 10) = 3/36, & p(X = 11) = 2/36, & p(X = 12) = 1/36. & \end{array}$$

For all other values of $r \in \mathbb{R}$, $f_X(r) = 0$.

Often we want to know the probability that a random number has a value below a given bound. In other words, we want to know $p(X \leq k)$.

Definition 9.1.4 *Given a random variable X, the function $F_X : R \to \mathbb{R}$ given by*

$$F_X(r) = p(X \leq r)$$

is also a probability function on $\mathbb{R}$. It is called the cumulative distribution function *for X.*

Some probability distribution functions have special characteristics and applications that merit separate study. In the next section, we analyse three of them.

9.2 Examples of statistical distributions

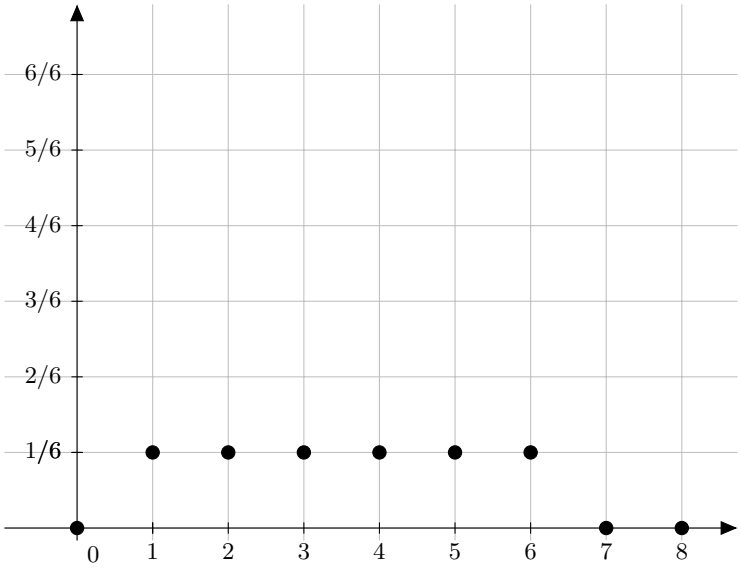
9.2.1 Uniform distribution

Definition 9.2.1 *A random variable has* uniform probability distribution *if the probability of all its (non-zero) values are the same.*

For example, the random variable in Example 9.1.3(b) where we roll a fair die and observe the number on the face showing up has uniform probability distribution:

$$p(X = 1) = p(X = 2) = p(X = 3) = p(X = 4) = p(X = 5) = p(X = 6) = 1/6$$

We can give a graphical representation of f_X :



9.2.2 Binomial distribution

Definition 9.2.2 *Any experiment that is repeated several times is called a* trial*. If the experiment has only two possible outcomes s and f say, then it is a* binomial trail *(or a Bernoulli trial, after James Bernoulli, who made important contributions to probability theory); one of the outcomes, say s, is called a* success*, and the other outcome f is called a* failure*.*

For example, the experiment of tossing a fair coin can be repeated several times (provided we don't lose or damage the coin, of course) and has only two possible outcomes: 'heads' or 'tails'. If we are interested in repeating this experiment several times and counting the number of 'heads', the outcome 'heads' would be a success (and 'tails' a failure).

Many problems can be formulated in terms of the probability of obtaining k successes when an experiment consists of n *mutually independent* binomial trials, i.e. if the outcome of a trial is not influenced by the outcomes of previous trials (the conditional probability of success on any given trial is the same whatever the outcomes of the other trials.)

Example 9.2.3 *A situation that a computer scientist might face is:*

> *Bits are sent over a communications channel in packets of 12. If the probability of a bit being corrupted over this channel is 0.1 and such errors are independent, what is the probability that no more than two bits in a packet are corrupted?*

Here the experiment is sending a bit over a communication channel, and we repeat it 12 times. The experiment has two possible outcomes: either the bit is corrupted or it is not. Because we are interested in counting the number of corrupted bits, success is observing a corrupted bit. Therefore, we have a binomial trial. Furthermore, we are told that errors are independent and we are given the probability of observing a success (i.e. a corrupted bit): 0.1. We are asked to calculate $p(X \leq 2)$.

Definition 9.2.4 *A random variable that counts the number of successes obtained from n independent binomial trials is said to have a* binomial distribution *with parameters n (number of trials) and p =probability of success.*

For example, and going back to the case with which we started this chapter, the random variable that counts the number of doubles when we roll, independently, two fair dice four times has a binomial distribution with parameters 4 (we roll the dice four times) and $6/36 = 1/6$ (the probability of rolling a double).

In the example of the bits, the random variable that counts the number of corrupted bits sent in a packet of 12 has a binomial distribution with parameters 12 (we send 12 bits) and 0.1 (the probability that a bit is corrupted).

The question now is how to calculate the probability distribution function. Let us go back to the random variable that counts the number of doubles when we roll, independently, two fair dice four times. It is easy to see that

$$p(X = 4) = p(DDDD) = p(D) \times p(D) \times p(D) \times p(D) = p(D)^4 = (1/6)^4$$

Note that $p(DDDD) = p(D) \times p(D) \times p(D) \times p(D)$ because the trials are mutually independent.

This was easy because there is only one way of getting four doubles. Similarly, we get

$$p(X = 0) = p(NNNN) = p(N) \times p(N) \times p(N) \times p(N) = p(N)^4 = (1 - (1/6))^4$$

To calculate $p(X = 1)$ we observe that events that produce only one double are

$$\{DNNN, NDNN, NNDN, NNND\}$$

i.e. we either obtain the double in the first, or in the second, or in the third, or in the fourth trial. Notice that the probability of each event is the same: $(1/6) \times (1 - (1/6))^3$. Again, this is because the trials are mutually independent so it doesn't matter at which trial the double was obtained. Therefore,

$$p(X = 1) = 4 \times (1/6) \times (1 - (1/6))^3$$

We multiply by 4 because there are four events, each with the same probability. The number 4 counts the number of times we can select one element (D) from a set of four outcomes.

The same reasoning gives us

$$p(X = 3) = 4 \times (1/6)^3 \times (1 - (1/6))$$

This is because there are also four ways of selecting three elements from a set of four outcomes.

What about $p(X = 2)$? We can count the number of events where there are exactly two doubles:

$$\{DDNN, DNDN, DNND, NDDN, NDND, NNDD\}$$

So, there are six ways of selecting two elements from a set of four outcomes. The probability of each such event is $(1/6)^2 \times (1 - (1/6))^2$ — two successes and two failures. Therefore,

$$p(X = 2) = 6 \times (1/6)^2 \times (1 - (1/6))^2$$

Now, given a total of n trials, let $C(n, k)$ be the number of way we can select exactly k successes out of the n outcomes. This number is often written

$$C(n, k) = \left(\begin{array}{c} n \\ k \end{array} \right)$$

Theorem 9.2.5 *The probability distribution function of a random variable X that counts the number of successes obtained from n independent binomial trials with $p(s)$ being the probability of observing success is defined by, for $k = 0, \ldots, n$:*

$$f_X(k) \;=\; p(X = k) \;=\; \left(\begin{array}{c} n \\ k \end{array} \right) p(s)^k (1 - p(s))^{n-k}$$

For all other values of $r \in \mathbb{R}$, $f_X(r) = 0$.

The numbers $C(n, k)$ are called *binomial coefficients* and can be calculated as follows:

$$C(n, k) = \left(\begin{array}{c} n \\ k \end{array} \right) = \frac{n!}{k! \times (n - k)!}$$

For example,

$$C(n, 0) = \left(\begin{array}{c} n \\ 0 \end{array} \right) = \frac{n!}{0! \times n!} = 1$$

$$C(n, 1) = \left(\begin{array}{c} n \\ 1 \end{array} \right) = \frac{n!}{1! \times (n - 1)!} = \frac{n!}{(n - 1)!} = n$$

$$C(n, n - 1) = \left(\begin{array}{c} n \\ n - 1 \end{array} \right) = \frac{n!}{(n - 1)! \times 1!} = n$$

$$C(n, n) = \left(\begin{array}{c} n \\ n \end{array} \right) = \frac{n!}{n! \times 0!} = 1$$

Therefore, the probability distribution function of the random variable that counts the number of doubles when we roll, independently, two fair dice four times is:

$$p(X = k) = \left(\begin{array}{c} 4 \\ k \end{array} \right) (1/6)^k (5/6)^{4-k} = \left(\begin{array}{c} 4 \\ k \end{array} \right) \frac{5^{4-k}}{6^4} = \left(\begin{array}{c} 4 \\ k \end{array} \right) \frac{5^{4-k}}{1296}$$

Example 9.2.6 *The random variable in Example 9.1.3(c) where X counts the number of 'heads' when we toss a fair coin twice has a binomial distribution with parameters $n = 2$ (we toss the coin twice) and $p = 1/2$ (the probability of observing 'heads'). Therefore, its probability distribution function is given by:*

$$p(X = k) = \left(\begin{array}{c} 2 \\ k \end{array} \right) (1/2)^k (1/2)^{2-k} = \left(\begin{array}{c} 2 \\ k \end{array} \right) \frac{1}{4}$$

We can confirm the values that we have previously calculated:

$$p(X=2) = \left(\begin{array}{c} 2 \\ 2 \end{array} \right) \frac{1}{4} = \frac{1}{4}$$
$$p(X=1) = \left(\begin{array}{c} 2 \\ 1 \end{array} \right) \frac{1}{4} = 2 \times \frac{1}{4} = \frac{1}{2}$$
$$p(X=0) = \left(\begin{array}{c} 2 \\ 0 \end{array} \right) \frac{1}{4} = \frac{1}{4}$$

Example 9.2.7 *Going back to Example 9.2.3, where X counts the number of corrupted bits in a packet of 12, the probability distribution function is given by:*

$$p(X=k) = \left(\begin{array}{c} 12 \\ k \end{array} \right) (0.1)^k (0.9)^{12-k}$$

We are asked to calculate $p(X \leq 2) = p(X=0) + p(X=1) + p(X=2)$:

$$p(X=2) = \left(\begin{array}{c} 12 \\ 2 \end{array} \right) (0.1)^2 (0.9)^{10} = \frac{12!}{2! \times 10!} \times (0.1)^2 (0.9)^{10} = 66 \times (0.1)^2 (0.9)^{10} \approx 0.23$$
$$p(X=1) = \left(\begin{array}{c} 12 \\ 1 \end{array} \right) (0.1)(0.9)^{11} = 12 \times (0.1) \times (0.9)^{11} \approx 0.377$$
$$p(X=0) = \left(\begin{array}{c} 12 \\ 0 \end{array} \right) (0.1)^0 (0.9)^{12} = (0.9)^{12} \approx 0.282$$

Therefore, $p(X \leq 2) \approx 0.889$

9.2.3 Geometric distribution

Suppose that n players decide to roll a fair die in turns until a 6 comes up. The first player to roll a 6 wins. If nobody rolls a 6, then they start again. What is the probability distribution of the random variable X that returns the winner of the game?

Let us calculate first the probability W that there will be a winner in any given round. We have $W = 1 - NW$ where NW is the probability that there is no winner. It is easy to calculate NW: it is the probability that no 6 is rolled in n trials — $(5/6)^n$. Therefore, $W = 1 - (5/6)^n$.

If we know that there was a winner in the round, the probability that it is player k is $5/6^{k-1}(1/6)$: the first $k-1$ players did not roll a 6 and the k^{th} player did. Therefore,

$$p(X=k) = \frac{(5/6)^{k-1}(1/6)}{1-(5/6)^n}$$

Definition 9.2.8 *A random variable that counts the number of independent trials that need to be repeated until a success is observed in rounds of n trials is said to have a* truncated

geometric distribution *with parameters n (number of trials in a round) and $1 \geq p > 0$ (probability of success). For any $k = 1, \ldots, n$*

$$p(X = k) = \frac{p(1-p)^{k-1}}{1-(1-p)^n}$$

We should check that this is a proper probability function, i.e.

$$\sum_{k=1}^{n} \frac{p(1-p)^{k-1}}{1-(1-p)^n} \;=\; 1$$

For that, we can use the property that we proved in Example 6.2.4: for any $x \in \mathbb{R}$ with $x \neq 1$

$$\sum_{i=0}^{m} x^i \;=\; \frac{x^{m+1}-1}{x-1}$$

If we make $i = (k-1)$, $n = (m+1)$ and $x = (1-p)$ (which satisfies $x \neq 1$ because $p > 0$) we obtain:

$$\sum_{k=1}^{n} (1-p)^{k-1} \;=\; \frac{(1-p)^n - 1}{(1-p)-1} \;=\; \frac{1-(1-p)^n}{p}$$

Given that

$$\sum_{k=1}^{n} \frac{p(1-p)^{k-1}}{1-(1-p)^n} \;=\; \frac{p}{1-(1-p)^n} \times \sum_{k=1}^{n} (1-p)^{k-1}$$

it follows that

$$\sum_{k=1}^{n} \frac{p(1-p)^{k-1}}{1-(1-p)^n} \;=\; 1$$

This distribution is called *truncated* because the trials are organised in rounds. If we now make an infinite round, i.e. if we keep rolling a die until a 6 comes out and use X to count the number of trials until that happens, we get $p(X = k) = (5/6)^{k-1} \times 1/6$ — we roll $k-1$ times a number other than 6 followed by a 6. This random variable has a geometric distribution.

Definition 9.2.9 *A random variable that counts the number of independent trials that need to be repeated until a success is observed is said to have a* geometric distribution *with parameter $1 \geq p > 0$ (probability of success). For any $k \in \mathbb{P}$*

$$p(X = k) = p(1-p)^{k-1}$$

Notice that X is no longer bounded: it can take any positive value.

To check that the probability function is well defined, recall that Example 6.2.4 gives us

$$\sum_{k=1}^{n}(1-p)^{k-1} \;=\; \frac{1-(1-p)^n}{p}$$

and therefore

$$\sum_{k=1}^{n}p(1-p)^{k-1} \;=\; 1-(1-p)^n$$

Because $1 \geq p > 0$ we know that $0 \leq (1-p) < 1$. Therefore $\lim_{n\to\infty}(1-p)^n = 0$ and

$$\sum_{k=1}^{\infty}p(1-p)^{k-1} \;=\; 1$$

That is, we have a proper probability function.

Notice that the calculations above also tell us how to calculate the cumulative probability distribution of the geometric distribution:

$$F_X(n) = p(X \leq n) = \sum_{k=1}^{n}p(1-p)^{k-1} \;=\; 1-(1-p)^n$$

Example 9.2.10 *One percent of bits transmitted through a digital transmission are received in error. What is the probability that the first error is received in the first 10 bits transmitted?*

Solution The random variable X that counts the number of bits transmitted until one is received in error has a geometric distribution with parameter $p = 0.01$. Using the cumulative probability distribution we have that

$$p(X \leq 10) = 1-(0.99)^{10} \approx 0.0956 \quad (9.56\%)$$

Notice that in some textbooks you may find that the geometric distribution is associated with the random variable $Y = (X-1)$, i.e. where Y counts the number of failures before the first success (not including the success).

9.3 Expected value

Consider the following game. Two people take it in turns to throw a die, and each person moves forward the number of meters equal to the value that they have thrown. The winner is the first person to get to the hundred meter line. How long (how many die throws) do we expect the game to last?

Of course it is possible (but unlikely) that both players will always throw 1s, in which case the game will take 199 throws. It is also possible that the first player will always throw a 6, in which case the game will last for 33 throws. But how long do we expect the game to last on average?

We know that the probability of throwing each of the possible numbers is equally likely. Then the average value of a throw is the sum of each possible score divided by its probability. So the average score is

$$1/6 + 2/6 + 3/6 + 4/6 + 5/6 + 6/6 = 21/6 = 3.5$$

Thus, on average, we expect to move 3.5 meters per throw (of course we won't actually move 3.5 meters on any one throw!). Since $3.5 \times 28 = 98$ and $3.5 \times 29 = 101.5$ we expect the game to last 58 throws, on average.

We often need to estimate the average outcome of an experiment. This average may not be an actual outcome, as in the game described above, but it gives a way of calculating the expected result of repeating an experiment a large number of times.

Definition 9.3.1 *Let X be a random variable over a sample space S. Its* expected value *(or* mean value *or simply* mean*), which we denote by $E(X)$, is given by*

$$E(X) = \sum_{s \in S} X(s) \times p(s) = \sum_{r \in X(S)} r \times p(X = r)$$

This means that if a random variable takes values $x_1, x_2, \ldots$, then

$$E(X) = x_1 p(X = x_1) + x_2 p(X = x_2) + \ldots$$

Example 9.3.2 *Let X be the random variable that returns the number obtained by rolling a die once. We have seen in Section 9.2.1 that X has a uniform probability distribution with $f_X(k) = 1/6$ for $1 \leq k \leq 6$. Thus we get that*

$$E(X) = 1(1/6) + 2(1/6) + 3(1/6) + 4(1/6) + 5(1/6) + 6(1/6) = 3.5$$

More generally:

Theorem 9.3.3 *If X has a uniform probability distribution with values $x_1, \ldots, x_n$ then*

$$E(X) = \frac{x_1 + x_2 + \ldots + x_n}{n}$$

Example 9.3.4 *The random variable X that counts the number of 'heads' obtained by tossing a fair coin 4 times has a binomial probability distribution with parameters 4 and 1/2:*

$$p(X = k) = \left(\begin{array}{c} 4 \\ k \end{array} \right) (1/2)^k (1/2)^{4-k} = \left(\begin{array}{c} 4 \\ k \end{array} \right) \frac{1}{16}$$

Thus we get that

$$\begin{aligned} E(X) &= 0\left(\begin{array}{c}4\\0\end{array}\right)\frac{1}{16} + 1\left(\begin{array}{c}4\\1\end{array}\right)\frac{1}{16} + 2\left(\begin{array}{c}4\\2\end{array}\right)\frac{1}{16} + 3\left(\begin{array}{c}4\\3\end{array}\right)\frac{1}{16} + 4\left(\begin{array}{c}4\\0\end{array}\right)\frac{1}{16} \\ &= 0 \times 1 \times \frac{1}{16} + 1 \times 4 \times \frac{1}{16} + 2 \times 12 \times \frac{1}{16} + 3 \times 4 \times \frac{1}{16} + 4 \times 1 \times \frac{1}{16} \\ &= \frac{4 + 12 + 12 + 4}{16} \\ &= 2 \end{aligned}$$

That is, the expected value is actually two 'heads'.

Notice that that the expected value 2 is the product of the two parameters 4 and 1/2. More generally:

Theorem 9.3.5 *If the random variable X has a binomial distribution with parameters n and p then*

$$E(X) = n \times p$$

Considering now a random variable X with a geometric distribution with parameter $p > 0$, we have

$$E(X) = \sum_{k=1}^{\infty} kp(1-p)^{k-1}$$

We have already shown that

$$\sum_{k=1}^{\infty} p(1-p)^{k-1} \;=\; 1$$

which implies

$$\sum_{k=1}^{\infty} (1-p)^{k-1} \;=\; \frac{1}{p}$$

If we take the derivative on p we get

$$\sum_{k=1}^{\infty} (k-1)(-1)(1-p)^{k-2} \;=\; \frac{-1}{p^2}$$

which is equivalent to

$$\sum_{k=1}^{\infty} k(1-p)^{k-1} \;=\; \frac{1}{p^2}$$

because for $k = 1$ the corresponding term is 0. Therefore, if we multiply by p on both sides, we get

$$E(X) = \sum_{k=1}^{\infty} kp(1-p)^{k-1} = p \times \sum_{k=1}^{\infty} k(1-p)^{k-1} = p \times \frac{1}{p^2} = \frac{1}{p}$$

Theorem 9.3.6 *If the random variable X has a geometric distribution with parameter p then*

$$E(X) = \frac{1}{p}$$

Returning to Example 9.2.10, we can conclude that the expected number of bits that will be transmitted until one is received in error is $1/0.01 = 100$.

9.4 Variance

Another important measure associated with a random variable is that of how far a data set is spread out in relation to the mean. Two random variables may have exactly the same mean but the way their values are distributed may be significantly different. For example, if we consider the random variables associated with the marks in modules, two modules may have the same mean but the actual marks may be spread quite differently. The measure of this spread may provide significant information on how assessment is effectively capturing the diversity of a cohort of students.

If X is a random variable and $E(X)$ is its expected value, then $X - E(X)$ is also a random variable and its values are precisely the distance in relation to the mean for each value of X. Because that distance can be negative, it is more convenient to use instead $(X - E(X))^2$.

Definition 9.4.1 *Given a random variable X on a sample space S, the* variance *of X, denoted by $V(X)$, is given by*

$$V(X) = E((X - E(X))^2) = \sum_{s \in S} (X(s) - E(X))^2 \times p(s) = \sum_{r \in X(S)} (r - E(X))^2 \times p(X = r)$$

This means that if a random variable takes values $x_1, x_2, \ldots$, then

$$V(X) = (x_1 - E(X))^2 p(X = x_1) + (x_2 - E(X))^2 p(X = x_2) + \ldots$$

That is, $V(X)$ is the weighted average of the square of the deviation of X in relation to its mean value $E(X)$. The *standard deviation* of X, denoted by $\sigma(X)$, is defined to be $\sqrt{V(X)}$.

The following theorem provides a useful way of simplifying the calculation of the variance:

Theorem 9.4.2 *Given a random variable X, $V(X) = E(X^2) - E(X)^2$.*

The proof of this theorem is quite simple:

$$\begin{aligned} V(X) &= \textstyle\sum_{s\in S}(X(s)-E(X))^2 \times p(s) \\ &= \textstyle\sum_{s\in S}((X(s)^2 - 2X(s)E(X) + E(X)^2) \times p(s) \\ &= \textstyle\sum_{s\in S} X(s)^2 \times p(s) - 2E(X)\sum_{s\in S} X(s) \times p(s) + E(X)^2 \sum_{s\in S} p(s) \\ &= E(X^2) - 2E(X)E(X) + E(X)^2 \quad (\text{because } \textstyle\sum_{s\in S} p(s) = 1) \\ &= E(X^2) - E(X)^2 \end{aligned}$$

For the statistical distributions studied in Section 9.2 we have:

Theorem 9.4.3 *Given a random variable X:*

- *If X has a uniform probability distribution with values $x_1, \ldots, x_n$ then*

$$V(X) = \frac{x_1^2 + x_2^2 + \ldots + x_n^2}{n} - \left(\frac{x_1 + x_2 + \ldots + x_n}{n}\right)^2$$

- *If X has a binomial distribution with parameters n and p, then*

$$V(X) = n \times p \times (1-p)$$

- *If X has a geometric distribution with parameter p, then*

$$V(X) = \frac{(1-p)}{p^2}$$

9.5 Exercises

1. Two dice are thrown 6 times and $X : S \to \mathbb{R}$ counts the number of doubles, so $X(s) = 0, 1, 2, 3, 4, 5, 6$. Write down the formula for $p(X = k)$, the probability that the random variable X has value k. Hence find $p(X = 3)$.

2. A pyramid die has 4 equal faces each of which is a triangle. The faces have the numbers 2,4,6 and 8 on them. Suppose that X is the random variable given by rolling this die once and recording the score. Calculate $E(X)$.

3. An experiment has outcomes a and b, and $p(a) = \frac{1}{3}$. The experiment is repeated 8 times and $X(s) =$ 'number of occurrences of a in s'. Find $f_X(r)$ for all $r \in \mathbb{R}$, and plot a graph of f_X.

4. Calculate $E(X)$ for the random variable X in Exercise 3.

5. Let $X : S \to \mathbb{R}$ have values 0,1,2,3,4,5,7. Suppose that X has binomial distribution with associated probability $p = 1/4$. Use the formula for f_X to find $E(X)$. Hence verify that $E(X) = 7p$, as required by Theorem 9.3.6.

Chapter 10

Some properties of numbers

Summarized below are some properties of numbers which you need to know. (You are not expected to be able to prove these properties, but you must know them and be able to use them.)

1. A number n is *prime* if
   (a) $n \in \mathbb{N}$ and $n \geq 2$, and
   (b) If $m \in \mathbb{P}$ and m divides n then $m = n$ or $m = 1$.

   The set of primes is infinite. The first few primes are

   $$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, \ldots$$

2. Every integer $n \in \mathbb{Z}$ can be written as a product of primes

   $$n = p_1 p_2 p_3 \ldots p_k \qquad \text{or} \qquad n = -p_1 p_2 p_3 \ldots p_k$$

   Where each p_i is a prime.

3. If p is a prime and p divides nm, where n and m are integers, then either p divides n or p divides m.

4. For any natural number $r \in \mathbb{N}$, every element of $\mathbb{Z}$ can be written in exactly one of the following forms:

   $$kr, \quad kr + 1, \quad kr + 2, \quad \ldots, \quad kr + (r - 1),$$

   where $k \in \mathbb{Z}$.

5. If p is a prime and p divides n^2 then p^2 divides n^2. (This follows from 3.)

6. If $k, h, r \in \mathbb{Z}$ and if r divides h and $(h + k)$, then r divides k.

7. The *logarithm* to base a of $x \in \mathbb{R}$, when $x > 0$, is the (real) number r such that $a^r = x$ (such a number always exists). We write $log_a(x) = r$.

   In particular, $log_a(1) = 0$, $log_a(a) = 1$, and $a^{log_a(x)} = x$.

   If $b > a > 1$ then the number r such that $a^r = s$ will need to be bigger than the number l such that $b^l = s$. So we see that if $b > a$ then $log_b(x) < log_a(x)$, for all $x > 1$.

   For all $a, x, y > 0$ we have $log_a(xy) = log_a(x) + log_a(y)$ and $log_a(x^y) = ylog_a(x)$.

   For all $a, b, x > 0$ we have $log_b(x) = log_b(a)log_a(x)$.