MATH2301: GAMES, GRAPHS AND MACHINES

Elizabeth Ormerod

Australian National University

Revised: Tamiru Jarso 2007
Malcolm Brooks 2012 - 2017

CONTENTS

1. RELATIONS

Elements within a set, or elements of different sets, often have a special connection with one another that can be described as a *relation*. Binary relations are relations between pairs of elements, and these will be the most important class of relations that we study. One type of binary relation with specified properties is called a *partial ordering*, and elements related by a partial ordering can be represented graphically in a very simple way. Another type of binary relation is an *equivalence relation*, and elements related by an equivalence relation can be grouped into classes. An extension of the binary relation is the *n*-ary relation. These *n*-ary relations form an important theoretical model for *relational databases*.

1.1 Binary Relations

If we learn that two people, Julie and Brian, say, are related we understand that there is a family connection between them. Perhaps they are cousins, siblings, niece and uncle, or whatever. They stand out from other people as a pair because there is a relationship that they satisfy. The mathematical analogue is to distinguish certain ordered pairs of objects from other ordered pairs, because the components of the "distinguished" pairs satisfy some relationship that the components of the other pairs don't.

Example

Let $S = \{1,2\}$ and $T = \{2,3\}$. Then $S \times T = \{(1,2),(1,3),(2,2),(2,3)\}$. If we are interested in the relation "is equal to", then the only distinguished pair of $S \times T$ is $(2,2)$. If we are interested in the relation "is less than", we would choose $(1,2),(1,3)$ and $(2,3)$ as the distinguished pairs of $S \times T$.In this example we could choose the distinguished pairs (x,y) by saying that $x = y$ in the first case, or $x < y$ in the second. Similarly the notation $x\boldsymbol{R}y$ indicates that the ordered pair (x,y) satisfies a relation $\boldsymbol{R}$.

The relation $\boldsymbol{R}$ may be defined by some verbal description or simply by listing the distinguished pairs of $\boldsymbol{R}$.

1 Definition. *Given two sets S and T, a* binary relation *on $S \times T$ is a subset of $S \times T$*

More Examples

1. Let $S = \{1,2\}$ and $T = \{2,3,4\}$. Let $\boldsymbol{R}$ be the relation on $S \times T$ given by the description $x\boldsymbol{R}y \leftrightarrow x+y$ is odd. Then $(1,2) \in \boldsymbol{R}$, $(1,4) \in \boldsymbol{R}$, and $(2,3) \in \boldsymbol{R}$. On the other hand, $(2,2) \notin \boldsymbol{R}$ and $(2,4) \notin \boldsymbol{R}$. We can also write $\boldsymbol{R} = \{(1,2),(1,4),(2,3)$ where we list all the ordered pairs from $S \times T$ that belong in R.

2. Take $S = \{1,2,3,4\}$ and $T = \{a,b,c\}$. We can define a relation $\boldsymbol{R}$ on $S \times T$ simply by listing the ordered pairs that we want to belong to $\boldsymbol{R}$. These ordered pairs do not need to have a verbal description. Thus, we may define $\boldsymbol{R}$ to be: $\boldsymbol{R} = \{(1,a),(1,c),(2,b),(4,a),(4,b)\}$. The only reason that we have chosen these particular ordered pairs is that we like them.

3. For each of the following binary relations $\boldsymbol{R}$ on $\mathbb{N} \times \mathbb{N}$, decide whether the ordered pairs belong in $\boldsymbol{R}$. ($\mathbb{N} = \{1,2,3,\ldots\}$, the set of natural numbers.)

   (a) $x\boldsymbol{R}y \leftrightarrow x = y+1$; $(2,2),(2,3),(3,3),(3,2)$.

   (b) $x\boldsymbol{R}y \leftrightarrow x$ divides y: $(2,4),(4,2),(2,5),(2,6),(9,9)$.

   (c) $x\boldsymbol{R}y \leftrightarrow x$ is odd; $(2,3),(3,4),(4,5),(5,6)$.

   (d) $x\boldsymbol{R}y \leftrightarrow x > y^2$; $(1,2),(2,1),(5,2),(6,4),(4,3),(100,9),(8,65)$.

Directed Graphs of Relations

Any relation on a set A can be represented by a picture called a *directed graph* or *digraph*. Note that a "relation on a set A" is a shortcut for saying a "subset of $A \times A$". For example, suppose $A = \{1,2,3,4\}$ and

$$\boldsymbol{R} = \{(1,2),(1,3),(1,4),(2,1),(2,4),(3,3),(3,4)\}.$$

To represent $\boldsymbol{R}$ as a graph, we draw *vertices*, one for each element of A, and arrows or *directed edges*, one for each member of $\boldsymbol{R}$. If $(a,b) \in \boldsymbol{R}$, then the vertex a is connected to the vertex b by a directed edge (or arrow) from a to b. A digraph for the relation $\boldsymbol{R}$ of this example follows. Since $1\boldsymbol{R}2$, $1\boldsymbol{R}3$ and $1\boldsymbol{R}4$, there are arrows from vertex 1 to

vertices 2, 3, and 4. Similarly there are arrows from vertex 2 to vertices 1 and 4, and arrows from vertex 3 to itself and to vertex 4. The digraph is shown below. (Figure 1.1)

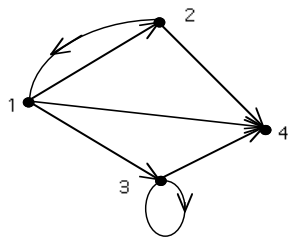


Figure 1.1: Digraph for the example above

It is also true that every digraph determines a relation, so that we may recover $\boldsymbol{R}$ from the diagram above. For example, since there is a directed edge from vertex 2 to vertex 4, we know that $(2,4) \in \boldsymbol{R}$. The loop at vertex 3 indicates that $(3,3) \in \boldsymbol{R}$.

Matrices for Relations

A matrix can also represent a relation between the finite sets S and T. Suppose that $S = \{s_1, s_2, \ldots, s_m\}$ and $T = \{t_1, t_2, \ldots, t_n\}$. Here the elements of S and T have been ordered in a particular way. The actual ordering is arbitrary, but once decided it must remain the same. A relation $\boldsymbol{R}$ between S and T can be represented by a matrix $M_R = [m_{ij}]$ where

$$m_{ij} = \begin{cases} 1 & \text{if } (s_i, t_j) \in \boldsymbol{R} \\ 0 & \text{if } (s_i, t_j) \notin \boldsymbol{R}. \end{cases}$$

The matrix M_R consists of zeros and ones only, and there is a 1 at the (i, j) entry when s_i is related to t_j.

Examples

1. Let $S = \{1,2,3\}$ and $T = \{2,3,4,5\}$, and suppose $\boldsymbol{R} = \{(1,2),(1,4),(2,2),(2,5),(3,2)\}$. Then the matrix for $\boldsymbol{R}$ is given by

$$M_R = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

   We usually omit the labelling of the rows and columns, but it is used here to help clarify the definition.

2. Let $A = \{1,2,3,4,5\}$ and $\boldsymbol{R} = \{(1,2),(1,3),(1,4),(2,3),(4,4),(4,2)\}$. The graph of this relation is shown below (Figure 1.2)

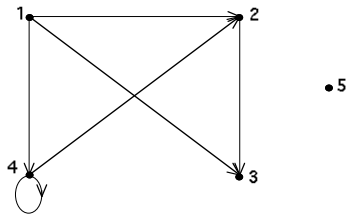


Figure 1.2: Digraph for Example 2 above

   and its matrix is

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

1.2 Properties of Relations

A binary relation on a set may have certain important properties, some of which are used to classify relations.

2 Definition. *Let $\boldsymbol{R}$ be a binary relation on a set S. ($\boldsymbol{R}$ is a subset of $S \times S$.) Then*

(a) *$\boldsymbol{R}$ is* reflexive *if $(x,x) \in \boldsymbol{R}$ for* all *$x \in S$.*

(b) *$\boldsymbol{R}$ is* symmetric *if $(x,y) \in \boldsymbol{R}$ implies that $(y,x) \in \boldsymbol{R}$ for any elements $x, y \in S$.*

(c) *$\boldsymbol{R}$ is* transitive *if whenever both $(x,y) \in \boldsymbol{R}$ and $(y,z) \in \boldsymbol{R}$, then (x,z) is also in $\boldsymbol{R}$, for elements $x, y, z \in S$.*

Examples

1. Let $\mathcal{H}$ be the set of all living humans and let $\boldsymbol{R}_1$ be the relation defined by

$$x\boldsymbol{R}_1 y \leftrightarrow y \text{ is a biological parent of } x,$$

where $x, y \in \mathcal{H}$. Then $\boldsymbol{R}_1$ is not reflexive, because for any human being x it is not true that x is the biological parent of x. Similarly $\boldsymbol{R}_1$ is not symmetric, since for humans x and y, if y is a biological parent of x, then x is not a biological parent of y. I will leave the discussion of transitivity to you.

2. Let $\boldsymbol{R}_2$ be the relation defined on $\mathbb{N}$ as follows:

$$\boldsymbol{R}_2 = \{(x,y) \in \mathbb{N} \times \mathbb{N} \mid x+y \text{ is even}\}.$$

Then $\boldsymbol{R}_2$ is reflexive since for all $x \in \mathbb{N}$, $x+x = 2x$ which is always even.. This means that $(x,x) \in \boldsymbol{R}_2$ for all $x \in \boldsymbol{R}_2$.

$\boldsymbol{R}_2$ is symmetric since if $(x,y) \in \boldsymbol{R}_2$, this means that $x+y = 2k$ for some integer k (this is a mathematical way of saying that $x+y$ ie even). It is clear then that $y+x = 2k$ also, and so $(y,x) \in \boldsymbol{R}_2$. Hence $(x,y) \in \boldsymbol{R}_2$ implies that $(y,x) \in \boldsymbol{R}_2$.

$\boldsymbol{R}_2$ is also transitive, since if $(x,y) \in \boldsymbol{R}_2$ and $(y,z) \in \boldsymbol{R}_2$, then we have (i) $x+y = 2k$ and (ii) $y+z = 2j$ for some integers k and j. We now want to show that $x+z$

must always be even, and so we look for an expression for $x+z$. From (i) we get that $x = 2k - y$ and from (ii), $z = 2j - y$. Hence

$$\begin{aligned} x+z = & \quad (2k-y)+(2j-y) \\ = & \quad 2k+2j-2y \\ = & \quad 2(k+j-y). \end{aligned}$$

This shows that $x+z$ is even, and hence $(x,z) \in \boldsymbol{R}_2$.

3. Consider the relation "$\leq$" on the set $\mathbb{N}$ of natural numbers. The relation is reflexive since for any positive integer x, it is true that $x \leq x$. It is also a transitive relation since for any positive integers x, y and z, if $x \leq y$ and $y \leq z$, then it is always true that $x \leq z$.

   However, this relation is not symmetric, since for example, $3 \leq 4$ does not imply that $4 \leq 3$. The relation "$\leq$" does have another special property. If x and y are integers and $x \leq y$ and $y \leq x$, we conclude that $x = y$. This characteristic is described by saying that "$\leq$" is antisymmetric.

3 Definition. *Let $\boldsymbol{R}$ be a binary relation on a set S. Then $\boldsymbol{R}$ is* antisymmetric *if for $x, y \in \boldsymbol{R}$,*

$$\textit{whenever } (x,y) \in \boldsymbol{R} \textit{ and } (y,x) \in \boldsymbol{R}, \textit{ then } x = y.$$

More Examples

1. Let $S = \wp(\mathbb{N})$, the set of subsets of the natural numbers. Define a binary relation $\boldsymbol{R}$ on S for subsets A and B by

$$A\boldsymbol{R}B \leftrightarrow A \subseteq B.$$

   Then $\boldsymbol{R}$ is *reflexive* since every subset A is a subset of itself. That is, $A \subseteq A$, and so $A\boldsymbol{R}A$ for all $A \in S$.

   Also $\boldsymbol{R}$ is *transitive* because if $A \subseteq B$ and $B \subseteq C$ for subsets A, B and C, then $A \subseteq C$.

   Finally, $\boldsymbol{R}$ is *antisymmetric* since if $A \subseteq B$ and $B \subseteq A$, then $A = B$.

2. Let $A = \{1,2,3,4,5\}$ and let $\boldsymbol{T}$ be the relation on A given by

   $$\boldsymbol{T} = \{(1,1),(1,3),(1,5),(2,2),(2,3)(2,4),(3,1),(3,2),(3,3),(4,4),(4,5),(5,3),(5,5)\}.$$

   Then the graph of T is shown below (Figure 1.3):

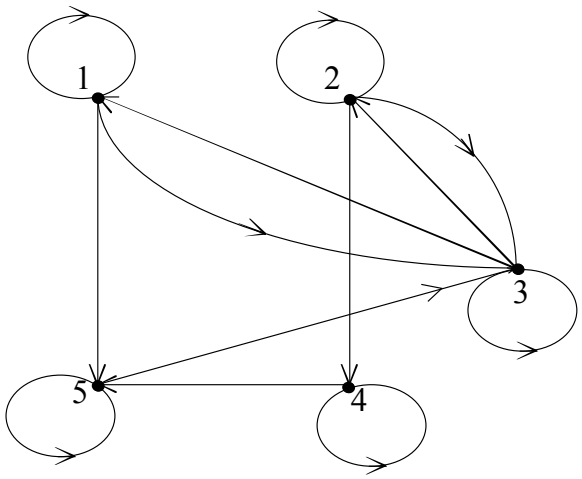


Figure 1.3: Digraph for the relation T

The relation $\boldsymbol{T}$ is *reflexive*. The easiest way to see this is to note that there are loops at each of the vertices of the graph representing $\boldsymbol{T}$. Alternatively, we see that $(a,a) \in \boldsymbol{T}$ for $a = 1,2,3,4,5$.

$\boldsymbol{T}$ is *not symmetric*, since while we have both $(1,3)$ and $(3,1)$ in $\boldsymbol{T}$, we also have $(1,5) \in \boldsymbol{T}$ but $(5,1) \notin \boldsymbol{T}$. From the graph we can see that $\boldsymbol{T}$ is not symmetric because we have directed lines in only one direction between 1 and 5, 2 and 4, and 4 and 5. For the relation to be symmetric, every time there is a directed edge say from vertex a to vertex b, there would need to be a directed edge from vertex b to vertex a.

$\boldsymbol{T}$ is also *not antisymmetric*. This is because we have both $(1,3)$ and $(3,1)$ in $\boldsymbol{T}$, but clearly $1 \neq 3$. From the graph we detect that $\boldsymbol{T}$ is not antisymmetric precisely because there are directed edges in both directions between the vertices 1 and 3, and between the vertices 2 and 4.

Finally, $\boldsymbol{T}$ is not transitive because, for example, $(5,3) \in \boldsymbol{T}$, $(3,1) \in \boldsymbol{T}$, but $(5,1) \notin \boldsymbol{T}$. From the graph we can see the lack of transitivity by noticing that we have a path from 5 to 3 to 1, but we have no shortcut from 5 to 1. To show that $\boldsymbol{T}$ is not transitive, we only need to find one example of a missing shortcut. As an exercise, find other missing shortcuts.

Matrices and the Properties of Relations

The matrix of a relation on a set A can be used to determine whether the relation has some of these properties. First note that the matrix for a relation on a set A is a square matrix. The number of rows and columns is the same as the number of elements in the set A. Recall that a relation $\boldsymbol{R}$ on a set A is reflexive if $(a,a) \in \boldsymbol{R}$ for all $a \in A$. So if $A = \{a_1, a_2, \ldots, a_n\}$, $\boldsymbol{R}$ is reflexive if and only if $(a_i, a_i) \in \boldsymbol{R}$ for $i = 1, 2, \ldots, n$. This means that in the matrix M_R of $\boldsymbol{R}$, $m_{ii} = 1$ for $i = 1, 2, \ldots, n$. In less technical language, this says that the relation $\boldsymbol{R}$ is reflexive, if and only if its matrix M_R has 1's on its main diagonal.

Examples

1. Let $A = \{1, 2, 3, 4\}$ and let $\boldsymbol{R}$ be the relation "$\leq$" on A. This means that $(a,b) \in \boldsymbol{R}$ if and only if $a \leq b$, for $a, b \in A$. The matrix for R is given by

$$M_R = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

   Since this matrix has 1's on the main diagonal, we can easily see that $\boldsymbol{R}$ is reflexive. We can also see that $\boldsymbol{R}$ is not symmetric, since M_R is not symmetric, and it is easy to deduce that the matrix of a symmetric relation must be symmetric.

   The pattern of a matrix that determines whether or not a relation is antisymmetric is a little more difficult ro describe. Recall that a relation $\boldsymbol{R}$ is antisymmetric if and only if $(x,y) \in \boldsymbol{R}$ and $(y,x) \in \boldsymbol{R}$ together imply that $x = y$. This means that if $\boldsymbol{R}$ is antisymmetric and $x \neq y$ and $(x,y) \in \boldsymbol{R}$ then it is impossible for (y,x) to belong in $\boldsymbol{R}$. Consequently, in the matrix M of an antisymmetric relation, if $m_{ij} = 1$ and $i \neq j$, then $m_{ji} = 0$. In other words, either $m_{ij} = 0$ or $m_{ji} = 0$ (or both) when $i \neq j$. This form is illustrated in the diagram below.

$$M_R = \begin{bmatrix} \ddots & & 0 & & \\ & \ddots & \iddots & 0 & \\ & \iddots & \ddots & \iddots & 1 \\ 1 & & \iddots & \ddots & \iddots \\ & 0 & & \iddots & \ddots \\ & & 0 & & \ddots \end{bmatrix}$$

From this we can easily see that the matrix for the relation "$\leq$" on $A = \{1,2,3,4,5\}$ indicates that the relation is antisymmetric.

2. Test the following binary relations on the given sets A for reflexivity, symmetry, antisymmetry and transitivity.

   (a) $A = \mathbb{N}$: $x\boldsymbol{R}y \leftrightarrow x - y$ is even.

   (b) $A = \mathbb{N}$: $x\boldsymbol{R}y \leftrightarrow x$ divides y.

   (c) $A = \{$all lines in the plane$\}$: $x\boldsymbol{R}y \leftrightarrow x$ is parallel to y or x coincides with y.

   (d) $A = \mathbb{N}$: $x\boldsymbol{R}y \leftrightarrow x = y^2$.

   (e) $A = \{0,1\}$: $x\boldsymbol{R}y \leftrightarrow x = y^2$.

   (f) $A = \{1,2,3\}$: $\boldsymbol{R} = \{(1,1),(2,2),(3,3),(1,2),(2,1)$;

   (g) $A = \mathbb{N}$: $x\boldsymbol{R}y \leftrightarrow x + y$ is odd.

   (h) $A = \mathbb{R}$ (the set of real numbers: $x\boldsymbol{R}y \leftrightarrow |x - y|$ is divisible by 3.

Relations and Functions

We will consider functions in more detail in a later section. But here we consider functions as a special type of relation. We take a function f from a set A to a set B. We usually write $f : A \to B$. This means that for every element $a \in A$, f assigns exactly one element of the set B. We usually write $f(a) = b$. We can identify f by writing a set of ordered pairs $(a,b) \in A \times B$ where $b = f(a)$. So suppose $A = \{1,2,3,4\}$ and $B = \{1,2,3,4,5,6,7,8\}$ and f is defined by $f(x) = 2x$. Then we

can write $f = \{(1,2),(2,4),(3,6),(4,8)\}$ and we recognise f as a relation. However, it is a special type of relation, as not all relations can be considered as functions. What distinguishes a function in the set of relations? You should notice that when we represent f as a set of ordered pairs (a,b), every element of A occurs once, and only once, as the first component of an ordered pair.

4 Definition. *A* function $f : A \to B$ *is a relation from A to B with the property that each* $a \in A$ *belongs to a unique ordered pair* (a,b) *in* f.

Examples

1. Let the function g assign to each name in the set $A = \{\text{Brett, Mark, David, Alan, Rachael}\}$ the number of different letters needed to spell the name. Then g is a function from A into $\mathbb{N}$, the set of natural numbers, and

$$g = \{(\text{Brett},5),(\text{Mark},4),(\text{David},5),(\text{Alan},4),(\text{Rachael},7)\}$$

.

2. We consider the following relations on the set $A = \{1,2,3\}$:

$$\begin{aligned} f &= \{(1,3),(2,3),(3,1)\}, \\ g &= \{(1,2),(3,1)\}, \\ h &= \{(1,3),(2,1),(1,2),(3,1)\}. \end{aligned}$$

Then

   (a) f is a function from A into A since each member of A appears exactly once as the first coordinate in an ordered pair in f. Here $f(1) = 3, f(2) = 3$ and $f(3) = 1$.

   (b) g is not a function from A into A since $2 \in A$ and 2 is not the first coordinate in any ordered pair in g.

   (c) Also h is not a function from A into A since 1 appears **twice** as the first coordinate of ordered pairs in h.

3. Let $A = \{a,b,c,d\}$. Then

   (a) $\{(b,a),(c,a),(d,a),(c,d),(a,d)\}$ is not a function from A to A as c appears as the first coordinate of two ordered pairs.

(b) $\{(d,d),(c,a),(a,b),(d,b)\}$ is not a function from A to A because d appears as the first coordinate of two ordered pairs. Another reason why f is not a function from A to A is that b does not appear as the first coordinate of any ordered pair. (Either reason just by itself is enough to stop f from being a function from A to A.)

(c) $\{(b,b),(d,b),(c,b),(a,b)\}$ is a function from A to A since each element in A appears exactly once as the first coordinate of an ordered pair.

With the definition of a function given above, it is easy to count the number of different functions from a finite set A to a finite set B. Suppose $|A| = m$ and $|B| = n$, then in a given function, each element of A must appear as the first coordinate of an ordered pair. The second coordinate can be any one of the n elements in B. That is, for each element in A there is a choice of n elements in B. The total number of functions we can write this way is $\underbrace{n \times n \times \cdots \times n}_{m} = n^m$.

Let $A = \{0,1,2\}$ and $B = \{a,b,c,d\}$, then there are $4^3 = 64$ different functions from A to B.

1.3 Partial Orderings

5 Definition. *A binary relation on a set A that is reflexive, antisymmetric and transitive is called a* partial ordering *on A. (The term is often slightly abbreviated to "partial order", dropping the "ing".) A partial ordering may be denoted by the symbol "$\leq$", even though it is not the usual inequality relation for real numbers. However, the alternative symbol "$\preceq$" is often used.*

From the previous examples we have the following instances of partial orderings.

- on $\mathbb{N}$: $x\boldsymbol{R}y \leftrightarrow x \leq y$,
- on $\wp(\mathbb{N})$: $A\boldsymbol{R}B \leftrightarrow A \subseteq B$,
- on $\{0,1\}$: $x\boldsymbol{R}y \leftrightarrow x = y^2$.

Example

Let $A = \mathbb{N}$, the set of natural numbers. we define the relation $\boldsymbol{R}$ on A as follows:

$$x\boldsymbol{R}y \quad \leftrightarrow \quad x \text{ divides } y$$

where $x,y \in A$. Then $\boldsymbol{R}$ is a partial ordering. It is *reflexive* since for any integer x, x divides itself. Thus $(x,x) \in \boldsymbol{R}$. $\boldsymbol{R}$ is *antisymmetric* since if x divides y and y divides

x, then $x = y$. Note that x divides y means that $y = kx$ for some integer k. Hence if x divides y and y divides x we have $y = kx$ and $x = jy$ for some (positive) integers k and j. Combining these equations gives us that $y = kjy$ or $kj = 1$. Since k and j are both positive integers (why?), we can only have $kj = 1$ if both k and j are equal to 1. This allows us to conclude that $x = y$.
Finally we want to show that $\boldsymbol{R}$ is *transitive*. Suppose that $x\boldsymbol{R}y$ and $y\boldsymbol{R}z$ for positive integers x, y, z. We want to show that $x\boldsymbol{R}z$. Our assumption means that x divides y, and y divides z. In more symbolic language we have

$$y = kx \qquad \text{and} \qquad z = jy$$

for some integers k and j. Combining these equations gives us $z = jkx = (jk)x$ and since jk is an integer this is enough to tell us that x divides z or $x\boldsymbol{R}z$ and hence that $\boldsymbol{R}$ is transitive.

Hasse Diagrams

A convenient way to represent a partial ordering on a finite set of moderate size is through the use of a *Hasse diagram*. These diagrams are named after Helmut Hasse (1898 - 1979), a German mathematician. In 1926 his textbook produced these diagrams as an aid to the study of polynomial equations. Read more about Helmut Hasse at the following site: http://www-gap.dcs.st-and.ac.uk/~history/Mathematicians/Hasse.html. Digraphs for partially ordered sets quickly become cluttered as the size of the underlying set increases. Hasse diagrams are an effort to 'de-clutter' the digraph by eliminating both obvious edges and those that can easily be reconstructed by "sight". In particular

- all edges implied by reflexivity (i.e. all loops) are omitted;
- the vertices are carefully arranged in the vertical dimension so that the direction of the edges are always upwards and hence all arrowheads are omitted:
- all edges implied by transitivity (i.e. all shortcuts) are omitted.

Five Examples

1. Let $X = \{a, b\}$ and let $S = \wp(X)$, the set of subsets of X. That the relation on S defined by $A \preceq B \leftrightarrow A \subseteq B$ is a partial order follows in exactly the same way as

for Example 1 on page 12. The digraph and Hasse diagram for this partial order are shown in Figure 1.4 below.

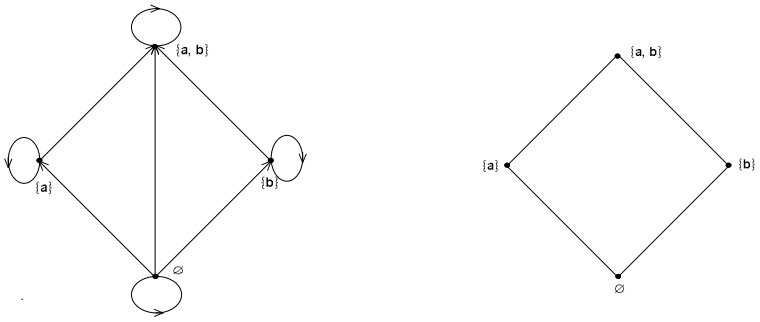


Figure 1.4: The digraph (left) of a partial order and the corresponding Hasse diagram.

The nine (directed) edges of the digraph are:
$(\emptyset,\emptyset)$, $(\emptyset,\{a\})$, $(\emptyset,\{b\})$, $(\emptyset,\{a,b\})$, $(\{a\},\{a\})$, $(\{a\},\{a,b\})$, $(\{b\},\{b\})$, $(\{b\},\{a,b\})$ and $(\{a,b\},\{a,b\})$.

The Hasse digram eliminates four edges because they are implied by the reflexive property: $(\emptyset,\emptyset)$, $(\{a\},\{a\})$, $(\{b\},\{b\})$ and $(\{a,b\},\{a,b\})$.

The Hasse digram also eliminates one edge because it is implied by the transitive property: $(\emptyset,\{a,b\})$.

2. Let $A = \{a,b,c,d\}$, and consider the relation

$$\boldsymbol{R} = \{(a,a),(a,c),(a,d),(b,b),(b,c),(b,d),(c,c),(c,d),(d,d)\}$$

We show that $\boldsymbol{R}$ is a partial ordering.

(a) It is *reflexive* because $(a,a),(b,b),(c,c)$ and (d,d) are all in $\boldsymbol{R}$.

(b) It is *antisymmetric* because there is no pair $(x,y), x \neq y$, in $\boldsymbol{R}$ for which (y,x) is also in $\boldsymbol{R}$.

(c) It is *transitive* because $\boldsymbol{R}$ includes all possible "shortcuts". We have $a\boldsymbol{R}c$, $c\boldsymbol{R}d$ and $a\boldsymbol{R}d$, and $b\boldsymbol{R}c$, $c\boldsymbol{R}d$ and $b\boldsymbol{R}d$. We also have situations like $a\boldsymbol{R}a$, $a\boldsymbol{R}c$ and $a\boldsymbol{R}c$ which we might call degenerate, because instead of having three variables involved like a,c,d, we only have two. While we need to be aware of these cases, it is clear that they provide the necessary "shortcuts".

Thus $\boldsymbol{R}$ is a partial ordering. The digraph and Hasse diagram for this partial order are shown in Figure 1.5 below.

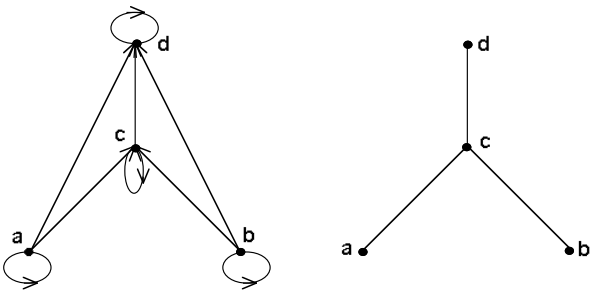


Figure 1.5: Digraph (left) and Hasse digram for Example 2

The edges eliminated in the Hasse diagram are (a,a), (b,b), (c,c), (d,d) because they are implied by reflexivity, and (a,d), (b,d) because they are implied by transitivity.

The matrix associated with $\boldsymbol{R}$ is

$$M_R = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Note that for the matrix the ordering of the rows and columns is a,b,c,d. If we had chosen a different ordering, the matrix would look different, even though it is the matrix for the same relation.

3. If a and b are positive integers, the notation "$a \mid b$" means that a is *divides* b, or equivalently, $b = qa$ for some integer q. We have shown in the example on page 17 that "a divides b" is a partial ordering on the natural numbers $\mathbb{N}$. It is also a partial ordering on any subset of $\mathbb{N}$, and hence it is a partial ordering $A = \{1,2,3,4,6,8,12,24\}$. The Hasse diagram for the relation "divides" on A is shown in Figure 1.6 below.

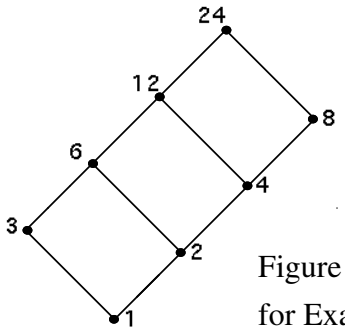


Figure 1.6: Hasse diagram for Example 3

4. As noted in the first of these five examples, the subset relation is a partial ordering on the set of subsets (the power set) of any set S. In this example we take $S = \{a,b,c,d\}$ and use Hasse diagram 1.7 below to show this relation.

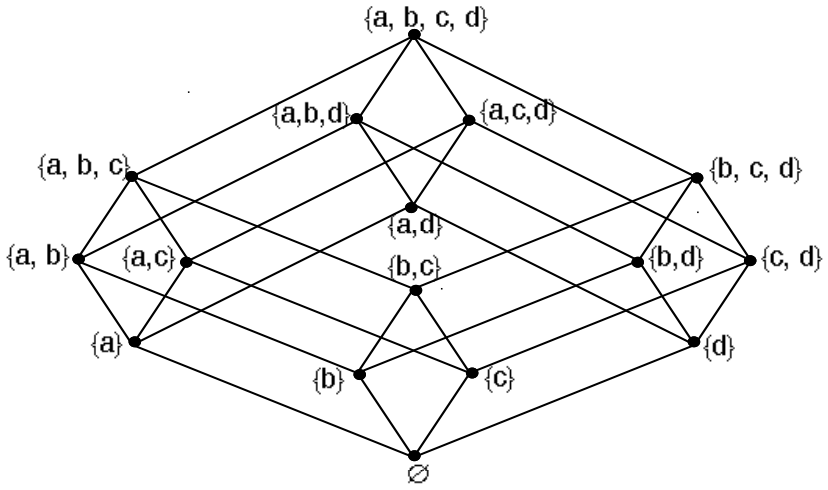


Figure 1.7: Hasse diagram for the subset relation on $\wp(\{a,b,c,d\})$

This is quite a complicated diagram, but the digraph of the relation would be so much more complicated. We just need to remember in reading the Hasse diagram that many subset relationships are implied rather than shown directly; namely all relationships implied by reflexivity (loops at every vertex) and all those implied by transitivity (shortcuts).

5. While the previous example is quite complicated, the diagrams of some partial orderings are very simple; for example the diagram at right (Figure 1.8). This is the Hasse diagram of an ordering that is called a *total ordering* because any two elements x and y are *comparable*: either $x \preceq y$ or $y \preceq x$. By contrast, in the second of these five examples (see Figure 1.5 on page 20) a and b are *incomparable*; that is neither $a \preceq b$ nor $b \preceq a$. Similarly in the third example (See Figure 1.6 on page 20) 3 and 8 (amongst others) are incomparable, as are the subsets $\{a,b\}$ and $\{b,c\}$ (among others) in the fourth example (see Figure 1.7 above). However, to repeat, in the ordering with Hasse diagram at right every pair of elements is comparable, so this ordering is total.

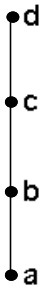


Figure 1.8: Hasse diagram for a total ordering on $\{a,b,c,d\}$

Because of the nature of their Hasse diagrams, total orderings or also called *linear orderings*. It would be a good exercise to construct the above linear ordering as a set of ordered pairs and to write out the associated matrix.

6 Definition. *A set on which a partial ordering is defined is called a* partially ordered set *or* poset*; if the ordering is total we have a* totally ordered set.

Counting Partial Orderings

Let $A = \{1,2,3\}$. We are going to count the number of different partial orderings that can be made on this small set A. The methods used are applicable to other counting arguments. Recall that a partial ordering, like any relation, is a subset of $A \times A$. In this case $A \times A = \{(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)\}$ and consists of nine ordered pairs. As a relation is a subset of A to count the number of relations that can be constructed on A we count the number of subsets of $A \times A$. Any set with 9 elements has 2^9 subsets, and so there are 2^9 relations on A. This includes all of $A \times A$, as well as the empty set.

We now want to know how many of these 2^9 relations are partial orderings. Since we can represent a partial ordering with a Hasse diagram, we use the Hasse diagram to help in the counting process. We can do this because the different types of Hasse diagram that we can draw with three vertices is very limited. In fact there are only five possible configurations for a Hasse diagram with three vertices. They are shown in Figure 1.9 below:

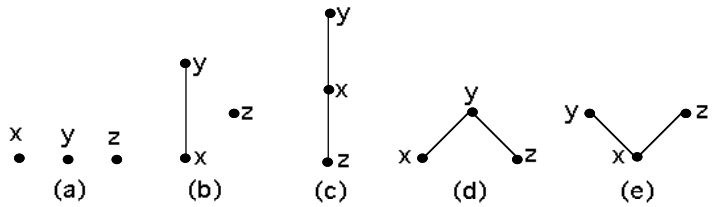


Figure 1.9: The five possible types of Hasse digram for a poset of three elements

To establish that these are the only types possible we argue as follows. If no element is comparable with any other element, we get type (a). If there is one pair of comparable elements, say x and y, with the third element z not comparable with anything else we get type (b). The remaining cases occur when there are 2 comparable elements, say $x \preceq y$, and the remaining element z is comparable with one or both of x and y.

- Case (c): z is comparable to both x and y (e.g $z \preceq x \preceq y$ as illustrated).
- Case (d): $z \preceq y$ but z is not comparable to x.
- Case (e): $x \preceq z$ but z is not comparable to y.

Most of these five types of diagram can give several different relations, depending on which elements from $A = \{1,2,3\}$ are called x, y and z. There is only one relation that has the Hasse diagram shown in (a). It really doesn't matter which of x, y, z are called $1, 2, 3$. the only partial order with this diagram is

$$\boldsymbol{R} = \{(1,1),(2,2),(3,3)\}.$$

This corresponds to the relation "is equal to".
For type (b) there are six different partial orderings. Their Hasse diagrams are shown in Figure 1.10 below.

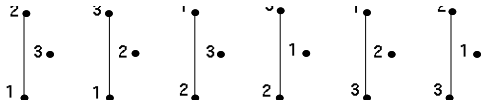


Figure 1.10: Hasse diagrams for the six posets of type (b)

For example, the Hasse diagram (i) represents the partial ordering

$$\boldsymbol{R} = \{(1,1),(1,2),(2,2),(3,3)\},$$

while the diagram (ii) represents

$$\boldsymbol{R} = \{(1,1),(1,3),(2,2),(3,3)\}.$$

Similarly there are six distinct partial orderings of type (c), and three each of types (d) and (e). The total number of partial orderings on a set with three elements is 1 + 6 + 6 + 3 + 3 = 19, and six of these are total orders.

Maximal and Minimal Elements

Recall that a poset is just a set, A say, together with a partial ordering $\preceq$. We denote such a poset by $(A, \preceq)$. Following the notation and terminology used for the special poset $(\mathbb{N}, \leq)$, we shall say that for any $x, y \in A$ if $x \preceq y$ but $x \neq y$ then x is *less than* y and y is *greater than* x, and will write $x \prec y$. When we study posets we find that some elements have certain properties that are important in many applications. The elements

we are interested in are the *maximal* and *minimal* elements. An element in a poset is *maximal* if it is not less than any other element in the poset, and an element in a poset is *minimal* if it is not greater than any other element.

7 Definition. (Maximal, Minimal) *Let $P = (X, \preceq)$ be a partially ordered set . An element $x \in X$ is called* maximal *if there is no $b \in X$ with $x \prec b$. Similarly, element x is called* minimal *if there is no $a \in X$ with $a \prec x$.*

Three Examples

1. Let $A = \{2,4,5,10,12,20,25\}$ and consider the poset $P = (A, |)$. (Recall that $x|y$ means that x divides y, so the notation $P = (A, |)$ means that P is the poset that consists of the set A and the relation "divides" on A.) The Hasse diagram for P is shown in Figure 1.11 below. The diagram is helpful in finding maximal and minimal elements.

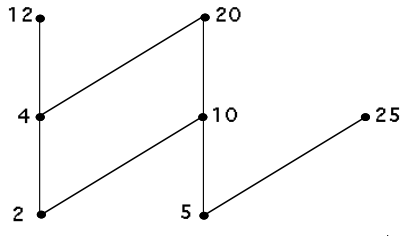


Figure 1.11: Hasse diagram for a poset with maximal and minimal elements

The maximal elements are 12, 20 and 25 and they are the elements at the top of any branch of the Hasse diagram. The minimal elements are 2 and 5 and they are the elements at the bottom of any branch of the diagram.

The concepts of *maximum* and *minimum* are similar to, but not the same as maxim*al* and minim*al*. An element x is maximum if all elements a in the poset can be compared with x and satisfy $a \preceq x$. Similarly, x is minimum if all elements b in the poset can be compared with x and satisfy $x \preceq b$. In the example above there is neither a maximum nor a minimum, but the next has both.

2. Figure 1.12 below is the Hasse diagram for the set of positive divisors of 36 ordered by divisibility. In this poset element 1 is minimum because it is strictly below all other elements in the poset. Element 36 is maximum because it strictly above all other elements in the poset. It is also true that 1 is a minimal element and 36 is a maximal element.

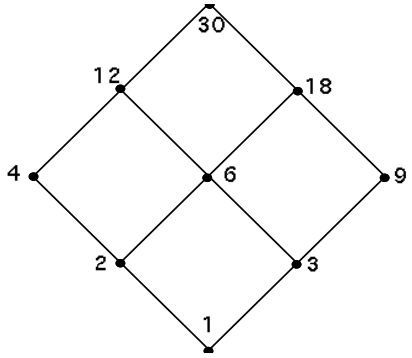


Figure 1.12: Hasse diagram for a poset with maximum and minimum elements

3. For a third example, we take the poset consisting of $A = \{1,2,3,4,5,6\}$ again ordered by divisibility The Hasse diagram is shown in Figure 1.13 below.

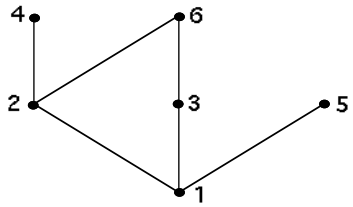


Figure 1.13: Hasse diagram for a poset with a minimum element but no maximum

In this example the elements 4, 6 and 5 are maximal, but there is no maximum. On the other hand, 1 is both minimal and minimum.

We summarise by giving the formal definition of maximum and minimum.

8 Definition. (Maximum and Minimum): *Let $P = (X, \preceq)$ be a partially ordered set. An element $x \in X$ is called* maximum *if, for all $a \in X$, we have $a \preceq x$. We call x* minimum *if, for all $b \in X$, we have $x \preceq b$.*

Upper and Lower Bounds

Let us consider the partial order "divides" on the set $\mathbb{N}$ of natural numbers. Take the two numbers 72 and 60, say. We are familiar with finding common divisors of two numbers like these. In this case the set of common divisors not including 1 is $\{2,3,4,6,12\}$. *All* the greater-than-one divisors of 72 and 60 are shown in Hasse Diagram 1.14 below.

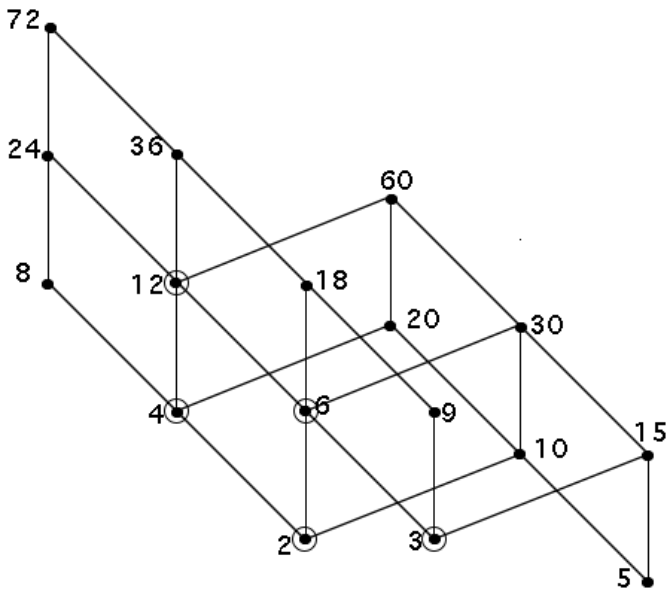


Figure 1.14: The greater-than-one divisors of 72 and 60

The common divisors are indicated on the diagram with circles around the vertices. It is good to understand where these vertices are in relation to 72 and 60. The largest of the common divisors is 12 and we can reach it from either 60 or 72 by coming along "downwards" paths. From 60 we come directly down to 12. From 72 we can come down via 36 or 24. The same is true of the other common divisors.
In a similar way we might be interested in the common multiples of say 3 and 4, that belong among the numbers in our diagram. It is easy to see that the common multiples of 3 and 4 are 12, 24, 36, 60 and 72. On the diagram we find these numbers by travelling along edges that go "upwards" from both numbers. So 20 is not a common multiple of 3 and 4, because on the diagram we reach it from 4 by an 'upwards' path, but not from 3. Because of the way we align our Hasse diagram, "upwards" just means up the page. It may not be directly up, but it must take us closer to the top of the page. We want to extend concepts like common divisors and common multiples to other posets, and we do it with the concept of upper and lower bounds.

9 Definition. (Lower and Upper Bounds):
Let $P = (X, \preceq)$ be a poset and let $a, b \in X$. We say that $x \in X$ is a lower bound *for a and b if $x \preceq a$ and $x \preceq b$. Similarly, we say that $x \in X$ is an* upper bound *for a and b if $a \preceq x$ and $b \preceq x$.*

The lower bound concept is an extension of the common divisor concept. In the example above, the common divisors of 72 and 60 are the same elements that satisfy the

definition of lower bounds of 72 and 60.

Another familiar concept is that of the greatest common divisor of two numbers. Looking at the divisors of 72 and 60 we identify that 12 is the greatest. On the diagram it is the common divisor that is nearest to both 60 and 72. We can also look for the lowest common multiples, and we see that the lowest common multiple of 3 and 4 is 12. Of the common multiples of 3 and 4, it is the one that is nearest to both 3 and 4.

We now define greatest lower bound and least upper bound for any poset.

10 Definition. (*Greatest Lower Bound, Least Upper Bound) :*
Let $P = (X, \preceq)$ be a poset and let $a, b \in X$. We say that $x \in X$ is a greatest lower bound *for a and b if*

1. *x is a lower bound for a and b, and*
2. *if y is a lower bound for a and b, then $y \preceq x$.*

Similarly, we say that $x \in X$ is a least upper bound *for a and b if*

1. *x is an upper bound for a and b, and*
2. *if y is an upper bound for a and b, then $x \preceq y$.*

Three More Examples

1. Let $A = \{2, 3, 6, 10, 12, 24, 60, 120\}$ and let $P = (A, |)$ where "$|$" indicates "divides" as before. Figure 1.15 below is the Hasse diagram for P.

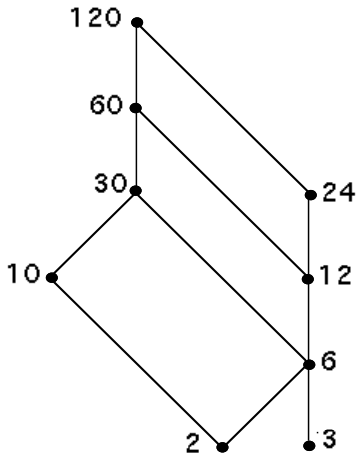


Figure 1.15: Some divisors of 120

The upper bounds of 3 and 10 are 30, 60 and 120, so the least upper bound of 3 and 10 is 30. The lower bounds of 24 and 30 are 2, 3 and 6, so the greatest lower bound of 24 and 30 is 6. We see that 2 and 3 have no lower bounds, while the upper bounds of 12 and 60 are 60 and 120 and hence the least upper bound is 60.

2. Let $(A, \preceq)$ be the poset with the Hasse diagram shown in Figure 1.16 below. (In this case the diagram does not merely illustrate the partial order, it *defines* it.)

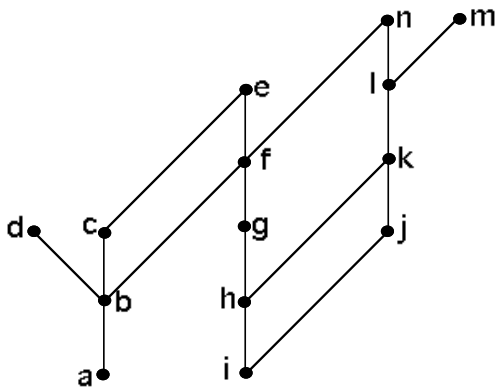


Figure 1.16: Hasse diagram for a poset with many missing bounds

The maximal elements are d, e, m and n, and the minimal elements are a and i. There is no maximum or minimum. Upper bounds for a and h are f, e, n, with the least being f. The only upper bound for a and k is n which is therefore also their least upper bound. Lower bounds for k and m are h, i, j, k and the greatest lower bound of k and m is k. There are no upper or lower bounds for c and k .

3. Let P be the poset defined by Hasse Diagram 1.17 below.

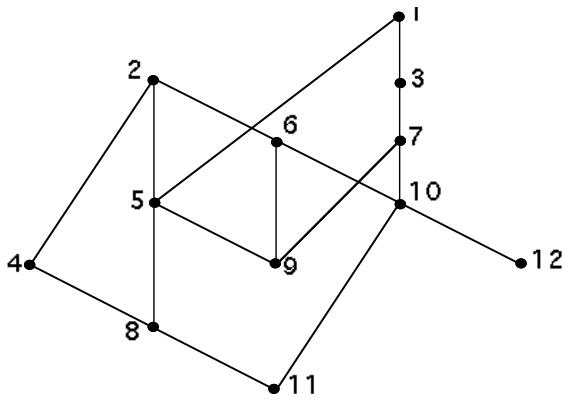


Figure 1.17: Hasse diagram for a poset with pairs with lower bounds but no greatest lower bound and pairs with upper bounds but no least upper bound

- Elements 2 and 3 have lower bounds 9, 10, 11 and 12. There is no greatest lower bound because, while $11 \preceq 10$ and $12 \preceq 10$, $9 \not\preceq 10$. Hence 10 is *not* a greatest lower bound. Also $10 \not\preceq 9$, so 9 cannot be a greatest lower bound.
  Elements 2 and 3 have no upper bounds and so no least upper bound.
- Elements 9 and 10 have upper bounds 1, 2, 3, 6 and 7, but they have no least upper bound. The element 7 is *not* a least upper bound because while $7 \prec 3$ and $7 \prec 1$, $7 \not\prec 6$. Similarly, $6 \not\prec 7$.
  Elements 9 and 10 have no lower bounds, and so no greatest lower bound.
- Elements 4 and 5 have 2 as their only, and hence least, upper bound have 8 and 11 as lower bounds, with 8 as the greatest lower bound.
- Elements 8 and 9 have upper bounds 1, 2 and 5 with least upper bound 5. On the other hand, 8 and 9 have no lower bounds and consequently no greatest lower bound.

If a pair of elements a and b of a poset have a greatest lower bound, it must be unique. For suppose that x and y were both greatest lower bounds of a and b. Then they must satisfy $x \preceq y$ since y is a greatest lower bound, and $y \preceq x$ since x is a greatest lower bound. But since a partial order is antisymmetric, this shows us that $x = y$. Similarly, if a and b have a least upper bound it must be unique.
The uniqueness of the greatest lower and least upper bound allows us to define binary operations that specify the least upper bound and greatest lower bound.

11 Definition. (Meet and Join):
Let $P = (X, \preceq)$ be a poset and let $a, b \in X$. If a and b have a greatest lower bound, it is called the meet *of a and b and it is denoted $a \wedge b$. If a and b have a least upper bound, it is called the join of a and b, and it is denoted $a \vee b$.*

Example

Let $A = \{a, b, c, d\}$ and let $X = \wp(A)$. (Recall that $\wp(A)$, the power set of A, is the set of subsets of A.) Fnally let $P = (X, \subseteq)$, so P is the poset of subsets of A where the partial order is set inclusion. Now consider $M = \{a, b\}$ and $N = \{b, c\}$. Then

$$M \wedge N = \{b\} = M \cap N \qquad \text{and} \qquad M \vee N = \{a, b, c\} = M \cup N.$$

This is not coincidence, as the symbol for meet, $\wedge$ is an abstraction of the symbol for intersection $\cap$, and the symbol for join, $\vee$ is an abstraction of the symbol for union $\cup$.

Lattices

In some posets meet and join are defined for all pairs of elements. Such a poset is called a lattice.

12 Definition. (Lattice):
Let P be a poset. We call P a lattice *if for all elements x and y of P, $x \wedge y$ and $x \vee y$ are defined.*

Four Examples of Lattices

1. Let P be the poset with Hasse diagram shown as Figure 1.18 below. The operations $\wedge$ and $\vee$ are defined for every pair of elements in this poset, as indicated by Table 1.1 that follows the diagram. Thus P is a lattice.

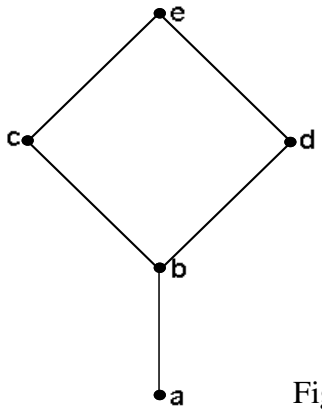


Figure 1.18: A simple lattice

$\wedge$	a	b	c	d	e
a	a	a	a	a	a
b	a	b	b	b	b
c	a	b	c	b	c
d	a	b	b	d	d
e	a	b	c	d	e

$\vee$	a	b	c	d	e
a	a	b	c	d	e
b	b	b	c	d	e
c	c	c	c	e	e
d	d	d	e	d	e
e	e	e	e	e	e

Table 1.1: Meet and Join tables for a simple lattice

2. Let $A = \{a,b,c,d\}$, let $X = \wp(A)$ and let $P = (X, \subseteq)$. Then, as indicated previously (see the example on page 29), every pair of subsets M and N of A has

$$M \wedge N = M \cap N \qquad \text{and} \qquad M \vee N = M \cup N.$$

Thus meet and join are defined for all elements of P, and so P is a lattice.

3. Consider the poset $(\mathbb{N}, |)$, the set of natural numbers ordered by divisibility. Recall that we define $\mathbb{N} = \{1,2,3,\ldots,\}$ and with this definition the poset $(\mathbb{N}, |)$ is a lattice, since for $x, y \in \mathbb{N}$, $x \wedge y$ is the greatest common divisor of x and y, and $x \vee y$ is their least common multiple.
However, if in the definition of $\mathbb{N}$ we include 0 (as some authors do), then $(\mathbb{N}, |)$ would not be a lattice since $0 \wedge 0 = \gcd(0,0)$ is not defined.

4. Let $P = (\mathbb{Z}, \leq)$, the set of integers ordered by $\leq$. For any $x, y \in \mathbb{Z}$,

$$x \wedge y = \begin{cases} x & \text{if } x \leq y \\ y & \text{if } x \geq y \end{cases} \qquad \text{and} \qquad x \vee y = \begin{cases} x & \text{if } x \geq y \\ y & \text{if } x \leq y. \end{cases}$$

We can rewrite this as $x \wedge y = \min\{x,y\}$ where $\min\{x,y\}$ stands for the smaller of x and y. Similarly, $x \vee y = \max\{x,y\}$, the larger of x and y. Thus P is a lattice.
A similar argument shows that every totally ordered set is a lattice.

It is interesting to ask what algebraic properties $\wedge$ and $\vee$ have. Some of the properties are summarised in the following theorem.

13 Theorem. *Any lattice $P = (X, \preceq)$ has the following properties. For all $x, y, z \in X$:*

- $x \wedge x = x \vee x = x$, *(idempotency),*
- $x \wedge y = y \wedge x$ *and* $x \vee y = y \vee x$, *(commutativity),*
- $(x \wedge y) \wedge z = x \wedge (y \wedge z)$ *and* $(x \vee y) \vee z = x \vee (y \vee z)$, *(associativity),*
- $x \wedge y = x \iff x \vee y = y \iff x \preceq y$.

The proofs are left as an exercise.

Before leaving this section on lattices we make a one final definition relating to the special case of total orderings (see example 3 above). Recall that total orderings are also called *linear orders*.

14 Definition. (Well Ordered Set): *A poset $(X, \preceq)$ is said to be* well ordered *if it is linearly ordered and every non-empty subset of X has a least element.*

The integers, $\mathbb{Z}$ with the usual ordering of $\leq$ are not well ordered since the negative integers do not have a least element. On the other hand, the natural numbers, $\mathbb{N}$, are well ordered. It is also possible to show that $\mathbb{N} \times \mathbb{N}$ with an appropriate ordering is well ordered.

Topological Sorting

Many sorting techniques are used in computer science. In this section we introduce a sorting process that may be used when we have a project, say, of 20 or more tasks where some tasks may be completed only after others have been finished. We may want to produce a schedule for doing these tasks that ensures that tasks are done in an appropriate order. This is the aim of topological sorting.
Let $(A, \preceq)$ be a poset (so $\preceq$ is a partial ordering on A). We aim to construct a linear order $\leq$ that is 'compatible' with the partial order $\preceq$.

15 Definition. (Compatibility, Topological Sorting):
Let $\preceq$ be a partial order on a set A. Another partial order $\leq$ on A is said to be compatible *with $\preceq$ if $a \leq b$ whenever $a \preceq b$. (Thus, as sets of ordered pairs, $\preceq$ is a subset of $\leq$.) In this case $\leq$ is called an* extension *of $\preceq$. Constructing a* ***linear*** *extension of a partial order is called* topological sorting.

Suppose that a development project at a computer company requires the completion of six tasks A,B,C,D,E,F. Some of these tasks can only be started after other tasks are finished. For example task C requires the prior completion of task A. A partial order on tasks is set up by considering task $X \prec Y$ if task Y cannot be started until task X has been completed (*e.g.* $\mathsf{A} \prec \mathsf{C}$). The Hasse diagram for for the six tasks, with respect to this partial order, is shown in Figure 1.19 below.

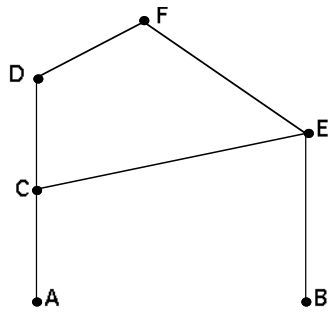


Figure 1.19: Partial order for six tasks

A possible order in which these tasks can be completed is given by

$$\mathsf{A},\ \mathsf{B},\ \mathsf{C},\ \mathsf{E},\ \mathsf{D},\ \mathsf{F}.$$

So we have imposed a total order on the six tasks, *viz.*: $\mathsf{A} < \mathsf{B} < \mathsf{C} < \mathsf{E} < \mathsf{D} < \mathsf{F}$.
This total order is compatible with the original partial order shown in the Hasse diagram. For example, in the diagram $\mathsf{A} \prec \mathsf{C}$, $\mathsf{A} \prec \mathsf{E}$, $\mathsf{A} \prec \mathsf{D}$ and $\mathsf{A} \prec \mathsf{F}$, and in the linear order A comes before all of C, E, D and F. Similarly, from the diagram $B \prec E$ and $B \prec F$ and B comes before both E and F in the linear order. We could complete this check for all tasks in the diagram. So our linear order is a linear extension of the original partial ordering.
However, this is not the only linear extension. Another is: $\mathsf{B} < \mathsf{A} < \mathsf{C} < \mathsf{D} < \mathsf{E} < \mathsf{F}$.

You may be surprised to learn that **any** partial order on a finite set has a compatible linear order. In other words:

16 Theorem. *Every finite poset can be topologically sorted.*

We will prove this theorem constructively by providing an algorithm. That the algorithm always works depends very much on the following result, whose proof is left as a little challenge.

17 Lemma. *Every finite non-empty poset has a minimal element.*

Topological Sorting Algorithm

We start with a finite non-empty poset $(A, \preceq)$. (Empty posets are vacuously linearly ordered, so can be ignored.) Lemma 17 ensures that every such poset has a minimal element. So we choose a minimal element a_1. Then $(A \setminus \{a_1\}, \preceq)$ is also a poset. If it is non empty we choose a minimal element a_2. If there are further elements in A we choose a minimal element in $A \setminus \{a_1, a_2\}$. We continue this process choosing a_{k+1} to be minimal in $A \setminus \{a_1, a_2, \ldots, a_k\}$ as long as elements remain. Since A is a finite set, this procedure must terminate. The result is a sequence of elements $a_1, a_2, \ldots, a_n$. The desired total order (*i.e.* a topological sort) is defined by

$$a_1 \leq a_2 \leq \cdots \leq a_n$$

because this total order is automatically compatible with the original partial order.

Example

Table 1.2 below lists the tasks required to assemble a bicycle. You must write a list of sequential instructions for the buyer to follow. Will the order in which the tasks are listed in the table be suitable? Give another sequence that could be used.

Table 1.2: Assembling a Bicycle

Task	Prerequisite Tasks
1. Tightening frame fittings	None
2. Attaching handle bars to frame	1
3. Attaching gear mechanism	1
4. Mounting tire on wheel assembly	None
5. Attaching wheel assembly to frame	1, 4
6. Installing brake mechanism	2, 3, 5
7. Adding pedals	6
8. Attaching seat	4
9. Adjusting seat height	7, 8

To help answer these questions first construct a Hasse diagram showing the partial order defined by the list of prerequisites. The diagram is shown below (Figure 1.20.)

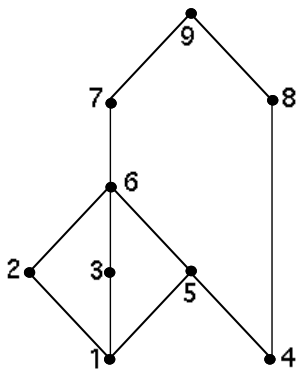
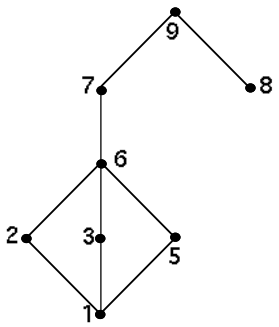
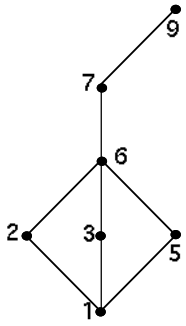


Figure 1.20: Hasse diagram for bicycle instructions

It is easy to see that the total order $1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9$ is compatible with the given partial order. To find an alternative partial order, let us use the algorithm we have described above. The first step is to choose a minimal element. Note that there are two minimal elements 1 and 4, so we can choose either. To ensure that this new total order is different from the first, let us choose 4. We then remove the vertex 4 from the diagram, and find a minimal element from the new diagram.

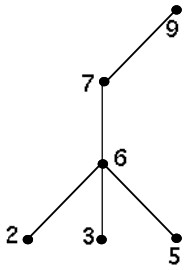


After Step 1

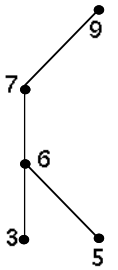


After Step 2

When vertex 4 has been removed the minimal elements are 1 and 8, so for step 2 let us choose 8, and then remove vertex 8. The total order at this stage is: $4 < 8$. After Step 2 the only minimal element is 1, so for step 3 we must chose 1, giving a total order so far: $4 < 8 < 1$. We then remove vertex 1 from the diagram.



After Step 3



After Step 4

After Step 3 there are 3 minimal elements, 2, 3 and 5. So for Step 4 let us choose the task 2, and remove vertex 2 from the diagram. The total order so far is: $4 < 8 < 1 < 2$. After Step 4 there are two minimal elements 3 and 5. For Step 5 let us chose task 5 and remove vertex 5 from the diagram. The total order so far is: $4 < 8 < 1 < 2 < 5$. After Step 5 the only minimal element is 3, so that must be our choice and the total order becomes: $4 < 8 < 1 < 2 < 5 < 3$. For the remaining steps there are no choices and the final total order is: $4 < 8 < 1 < 2 < 5 < 3 < 6 < 7 < 9$.

Realisers

We have seen that it is possible to construct several different linear orders that are compatible with a given partial order. We now take up the question of whether knowing

some or all of the linear orders compatible with a partial order will allow us to retrieve the original partial order. First consider an example. Let P be the poset defined by the Hasse diagram Figure 1.21.

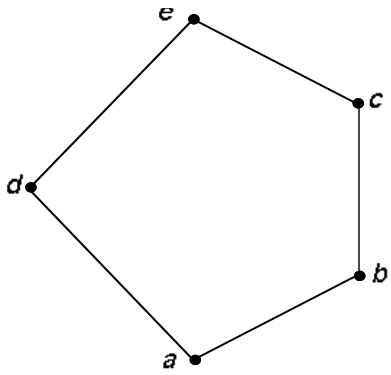


Figure 1.21: Hasse a diagram for a partial order to be 'retrieved'

There are three linear orders that are compatible with this partial order. They are

$$a < b < c < d < e$$
$$a < b < d < c < e$$
$$a < d < b < c < e.$$

With these three linear orders (or extensions) for P we have enough information to reconstruct the poset. Firstly, consider the elements c and e. Notice that $c < e$ in all the linear extensions. This happens only when $c \preceq e$ in the poset itself. Consider now the elements b and d. In the first linear extension we have $b < d$ and in the third we have $d < b$. If it were the case that $b \prec d$ in P, then we would have $b < d$ in all the linear extensions. Since this is not the case we deduce that b and d are incomparable in P. We summarise these remarks.

18 Lemma. *Let P be a finite partially ordered set and let x and y be distinct elements of P. If $x < y$ in all linear extensions of P (total orders compatible with P) then $x \prec y$ in P. Conversely, if $x < y$ in one linear extension, and $y < x$ in another, then x and y are incomparable in P.*

This observation gives us a way to store a partially ordered set P in a computer. We can save, as lists, the linear extensions of P. To see if $x \prec y$ in P, we simply check whether $x < y$ in all of the linear extensions.
However, we might run into problems as some partially ordered sets have a large number of linear extensions. Consider for example the 'antichain' of 7 elements whose

Hasse digram is displayed in Figure 1.22 below. (An antichain is a partial order in which no two distinct elements are comparable.)

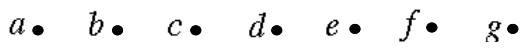


Figure 1.22: A 7-element antichain

There are 7! linear extensions of this poset, and if we had an anti-chain of 10 elements, there would be 10! (over 3 million) linear extensions. However, we don't need all these linear extensions to represent an anti-chain in our computer. In fact for any antichain we only need to use two. For our example these linear orders could be:

$$a < b < c < d < e < f < g$$
$$g < f < e < d < c < b < a$$

This works because for any two elements x and y in the antichain, if $x < y$ in one of the linear orders, then $x > y$ in the other.
The same idea works for the 5-element set we considered earlier (see Figure 1.21 on page 36). In fact, we can use just the first and third of the linear extensions that we listed for this poset, *i.e.*

$$a < b < c < d < e \qquad \text{and} \qquad a < d < b < c < e.$$

These two linear orders provide sufficient information to retrieve the original partial order. We have $b < d$ in the first and $d < b$ in the second, indicating that b and d are incomparable in the partial. Similarly we have $c < d$ in the first extension and $d < c$ in the second. So c and d are also incomparable in the partial order. On the other hand in both linear orders $a < d < e$ and $a < b < c < e$ (making use of transitivity). This allows us to reconstruct the partial order, and it would be sufficient to store these two extensions in a computer.

19 Definition. (Realiser): *Let $P = (X, \preceq)$ be a partially ordered set. Let $\mathcal{R}$ be a set of linear extensions of P. We call $\mathcal{R}$ a* realiser *of P if, for all $x, y \in X$ we have $x \preceq y$ in P if and only if $x \leq y$ in all linear extensions in $\mathcal{R}$. We say that $\mathcal{R}$* realises *P.*

The following result clarifies this definition.

20 Lemma. *Let P be a poset and let $\mathcal{R} = \{L_1, L_2, \ldots, L_t\}$ be a set of linear extensions of P. Then $\mathcal{R}$ is a realiser of P if and only if for all pairs of incomparable elements of P, x and y, there are indices i and j such that $x < y$ in L_i and $y < x$ in L_j.*

Two More Realiser Examples

1. Let P be a poset with realiser $\mathcal{R} = \{L_1, L_2\}$ where L_1, L_2 are given by:

$$L_1 : a < c < b < d < e < h < f < g,$$
$$L_2 : h < b < f < g < a < d < c < e.$$

We can reconstruct the partial order of P by listing the ordered pairs in the partial order. We recall that a partial order is reflexive, so we include (x,x) for all elements x in P. For $x \neq y$ in P we include (x,y) in the partial order if $x < y$ in both L_1 and L_2. Thus the partial order consists of the following ordered pairs:

$$\{(a,a),(a,c),(a,d),(a,e),(b,b),(b,d),(b,e),(b,f),(b,g),(c,c),$$
$$(c,e),(d,d),(d,e),(e,e),(f,f),(f,g),(g,g),(h,h),(h,f),(h,g)\}$$

The Hasse diagram for this relation is shown as Figure 1.23 below.

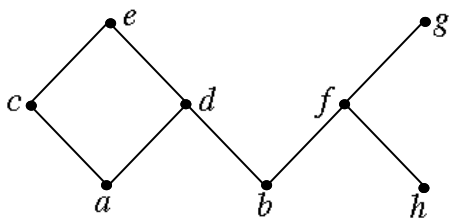


Figure 1.23: Hasse diagram for example 1

2. Let $P = (A, \preceq)$ be the poset with Hasse diagram shown in Figure 1.24 below.

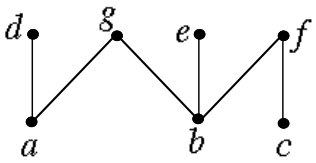


Figure 1.24: Hasse diagram for example 2

Let L_1 and L_2 be the following linear orders on A.

$$L_1 : a < d < b < g < e < c < f,$$
$$L_2 : c < b < f < e < a < g < d.$$

Let $\mathcal{R} = \{L_1, L_2\}$. We claim that $\mathcal{R}$ is a realiser for P. To see this we first check that L_1 and L_2 are linear extensions of P. For this we need to check that if $u \prec v$ in

P, then $u < v$ in both L_1 and L_2. Observe that $a < d$ and $a < g$ in both L_1 and L_2, $b < g$, $b < e$ and $b < f$ in both L_1 and L_2, and finally $c < f$ in both L_1 and L_2.

We now check that if u and v are incomparable in P, then $u < v$ in one of L_1 and L_2, and $v < u$ in the other. There are several cases and we aim to check these systematically. First consider a, b and c which are all incomparable with each other. We have (i) $a < b$ in L_1 and $b < a$ in L_2, (ii) $a < c$ in L_1 and $c < a$ in L_2, and (iii) $b < c$ in L_2 and $c < b$ in L_1. We check in the same manner the incomparabilities in $\{d, g, e, f\}$, in $\{a, e, f\}$, in $\{c, d, e, g\}$ and between b and d. Once we have done this we can claim that $\mathcal{R}$ is a realiser for P.

Dimension

In the second of the two previous examples we found a realiser for P comprising just 2 linear extensions L_1 and L_2. However it is clear that P has many other linear extensions, and that any set of linear extensions that contains L_1 and L_2 is a realiser for P. In particular, any poset is realised by the set of *all* its linear extensions. For an arbitrary poset P an interesting question is "what is the minimum number of linear extensions needed to make up a realiserfor P?' This minimum number is called the 'dimension' of P.

21 Definition. (Dimension): *Let P be a finite poset. The smallest size of a realiser of P is called the* dimension *of P.*

Each of the previous two examples had dimension 2. We can be sure of this because

1. For each we exhibited a realiser of size 2, so their dimensions are *at most* 2.
2. Neither is totally ordered, so their dimensions are *at least* 2. (A realiser of just one linear extension is only possible if the poset is a linearly ordered to start with.)

We finish this section with an example of a poset whose dimension is more than 2.

Example

Let P_3 be the poset with the Hasse diagram dispayed as Figure 1.25 below.

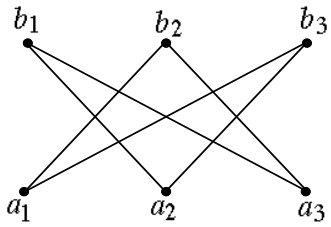


Figure 1.25: Hasse diagram for a poset of dimension 3

We show that P_3 has dimension 3. To do this we first show that P_3 has a realiser of size 3. Then we show that P_3 cannot have a realiser with less than 3 linear extensions.

Consider the following linear orders.

$$L_1 : a_2 < a_3 < b_1 < a_1 < b_2 < b_3,$$
$$L_2 : a_1 < a_3 < b_2 < a_2 < b_1 < b_3,$$
$$L_3 : a_1 < a_2 < b_3 < a_3 < b_1 < b_2.$$

To check that each of these is an extension of P_3 we need to check that $a_1 < b_2$ and $a_1 < b_3$ in each L_i, $(i = 1,2,3)$. We also check that $a_2 < b_1, a_2 < b_3$ and finally $a_3 < b_1, a_3 < b_2$ in each L_i. as this is true, each L_i is a linear extension of P_3.
Put $\mathcal{R} = \{L_1, L_2, L_3\}$. We claim that $\mathcal{R}$ is a realiser for P_3. There are three types of incomparable pairs in P_3:

- Pairs of the form (a_i, a_j). Note that $a_i < a_j$ in L_j and $a_i > a_j$ in L_i.
- Pairs of the form (b_i, b_j). Note that $b_i < b_j$ in L_i and $b_i > b_j$ in L_j.
- Pairs of the form (a_i, b_i). Note that $a_i > b_i$ in L_i, but $a_i < b_i$ in L_k $(k \neq i)$.

This is enough to show that $\mathcal{R}$ is a realiser for P_3.

The hardest step is to show that P_3 cannot have a realiser with less than 3 linear extensions. We argue by contradiction. Suppose that there is a realiser $\mathcal{R}$ of P_3 comprising only two linear extensions. (Again, it is not possible to have a realiser with 1 linear extension, since P_3 is not linear.) For $k = 1,2,3$ there must be an extension $L \in \mathcal{R}$ in which $a_k > b_k$, because (a_k, b_k) is an incomparable pair. But there are 3 such incomparable pairs and only two linear extensions in $\mathcal{R}$. Therefore by the pigeonhole principle there must be a linear extension L and two distinct indices i and j with $a_i > b_i$ and

$a_j > b_j$ in L. Since $b_j \succ a_i$ and $b_i \succ a_j$ in P_3, we must also have $b_j > a_i$ and $b_i > a_j$ in L. So in L we have

$$b_j > a_i > b_i > a_j > b_j$$

which implies that $b_j > b_j$, an impossibility. Therefore $\mathcal{R}$ is not a realiser of P_3, and so we cannot realise P_3 with fewer that 3 linear extensions. Thus $\dim P_3 = 3$ as claimed.

This example is a special case of a more general example. For $n \geq 2$, define $P_n = (S_n, \preceq)$ as follows:

$$\begin{aligned} S_n &= \{a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n\} \\ \preceq &= \{(a_i, a_i) : i = 1, \ldots, n\} \cup \{(b_i, b_i) : i = 1, \ldots, n\} \cup \{(a_i, b_j) : i, j = 1, \ldots, n, i \neq j\}. \end{aligned}$$

Using a proof similar to that for the case of P_3 we can show that P_n has dimension n.

1.4 Closure of Relations

There are times when we work with relations that are not transitive or not reflexive, perhaps, and it is important that we make a new relation containing the old one, that has the properties we require. Consider the relation $\boldsymbol{R}$ defined on the set $\mathcal{H}$ of humans by

$$x\boldsymbol{R}y \leftrightarrow x \text{ is a parent of } y.$$

Then this relation is not transitive since if $(x, y) \in \boldsymbol{R}$ and $(y, z) \in \boldsymbol{R}$ (*i.e.* x is a parent of y and y is a parent of z), then it is not usually the case that x is a parent of z. However, we might be interested in a more extended relation that does connect x to z in the situation described above. If we describe the relation $\mathbf{R}$ as the 'PARENT' relation, we might describe the extended relation, $\boldsymbol{R}'$ say, as the 'ANCESTOR' relation. A formal definition of $\boldsymbol{R}'$ could be

$$\begin{aligned} x\boldsymbol{R}'y \quad \leftrightarrow \quad & x\boldsymbol{R}y \quad \text{OR} \quad \exists n \in \mathbb{N}\ \exists x_1, x_2, \ldots, x_n \in \mathcal{H} \\ & x\boldsymbol{R}x_1 \text{ and } x_1\boldsymbol{R}x_2 \text{ and } \ldots \text{ and } x_{n-1}\boldsymbol{R}x_n \text{ and } x_n\boldsymbol{R}y \end{aligned}$$

This new relation $\boldsymbol{R}'$ is called the *transitive closure* of $\boldsymbol{R}$. In this section we construct several different closures for relations.

Reflexive and Symmetric Closures

Let $A = \{1,2,3,4\}$. Consider the relation $\boldsymbol{R} = \{(1,1),(1,4),(2,2),(2,3),(3,2),(4,3)\}$ which has matrix

$$M_R = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Since $\boldsymbol{R}$ is not reflexive, we aim to construct a reflexive relation containing $\boldsymbol{R}$ that is as small as possible. We denote this new relation by $r\boldsymbol{R}$. In this case it is quite easy, and we need only add the pairs (3, 3) and (4, 4) to make a reflexive relation

$$r\boldsymbol{R} = \{(1,1),(1,4),(2,2),(2,3),(3,2),(3,3),(4,3),(4,4)\}.$$

The matrix for $r\boldsymbol{R}$ is

$$rM_R = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

The matrix is obtained by adding 1's to the main diagonal, if they are not already there. This example illustrates a more general principle. Let $\boldsymbol{R}$ be a relation on the set A. The reflexive closure of $\boldsymbol{R}$ which we denote by $r\boldsymbol{R}$ can be formed by adding to $\boldsymbol{R}$ all the pairs of the form (a,a), where $a \in A$, that are not already in $\boldsymbol{R}$. Adding these pairs forms a new relation that is reflexive, contains $\boldsymbol{R}$, and is contained in any reflexive relation that contains $\boldsymbol{R}$.

The relation considered above, $\boldsymbol{R} = \{(1,1),(1,4),(2,2),(2,3),(3,2),(4,3)\}$, is not symmetric. In a similar way to what we have done above we can construct a new relation $s\boldsymbol{R}$ which is the smallest relation containing $\boldsymbol{R}$ and which is symmetric. In this example we need to add the pairs $(4,1)$ and $(3,4)$ since these are the only pairs of the form (b,a) that are not in $\boldsymbol{R}$ when $(a,b) \in \boldsymbol{R}$. Thus

$$s\boldsymbol{R} = \{(1,1),(1,4)(2,2),(2,3),(3,2),(3,4),(4,1),(4,3)\}$$

is symmetric and contains $\boldsymbol{R}$, and any symmetric relation that contains $\boldsymbol{R}$ must contain $s\boldsymbol{R}$. The matrix for $s\boldsymbol{R}$ is given by

$$sM_R = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}.$$

Again this example illustrates a more general principle. The symmetric closure of a relation $\boldsymbol{R}$ on a set A can always be constructed by adding to $\boldsymbol{R}$ all the pairs (b,a) where $(a,b) \in \boldsymbol{R}$. This produces a new relation that is symmetric, contains $\boldsymbol{R}$ and is contained in any symmetric relation that contains $\boldsymbol{R}$.

22 Definition. The Inverse of a Relation.
Let $\boldsymbol{R}$ be a relation on a set A. The inverse *of $\boldsymbol{R}$, denoted by $\boldsymbol{R}^{-1}$, is given by*

$$\boldsymbol{R}^{-1} = \{(b,a) \in A \times A : (a,b) \in \boldsymbol{R}\}.$$

This definition allows us to summarise this discussion with the following theorem.

23 Theorem. *If $\boldsymbol{R}$ is a relation on a set A and if $\Delta = \{(a,a) : a \in A\}$, then*

(i) *$r\boldsymbol{R} = \boldsymbol{R} \cup \Delta$, and*

(ii) *$s\boldsymbol{R} = \boldsymbol{R} \cup \boldsymbol{R}^{-1}$*

Two More Examples

1. Let $\boldsymbol{R}$ be the relation on the set $A = \{1,2,3,4\}$ with matrix

$$M_R = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Then the matrices for $r\boldsymbol{R}$ and $s\boldsymbol{R}$ are given by

$$rM_R = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix} \quad \text{and} \quad sM_R = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}.$$

Further, the matrix for $sr\boldsymbol{R}$ is the same as the matrix for $rs\boldsymbol{R}$ and is given by

$$srM_R = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}.$$

2. Let $\boldsymbol{R} = \{(m,n) \in \mathbb{N} \times \mathbb{N} : m > n\}$. Then

   (i) $r\boldsymbol{R} = \boldsymbol{R} \cup \Delta = \boldsymbol{R} \cup \{(m,m : m \in \mathbb{N}\} = \{(m,n) \in \mathbb{N} \times \mathbb{N} : m \geq n\}$.

   (ii) $s\boldsymbol{R} = \boldsymbol{R} \cup \boldsymbol{R}^{-1} = \{(m,n) \in \mathbb{N} \times \mathbb{N} : m > n\} \cup \{(m,n) \in \mathbb{N} \times \mathbb{N} : m < n\}$
   $= \{(m,n) \in \mathbb{N} \times \mathbb{N} : m \neq n\}$.

   (iii) $rs\boldsymbol{R} = sr\boldsymbol{R} = \mathbb{N} \times \mathbb{N}$.

Transitive Closure

The final situation we want to consider is the case when a relation is not transitive. We use the same example $\boldsymbol{R} = \{(1,1),(1,4),(2,2),(2,3),(3,2),(4,3)\}$. We examine precisely why this relation is not transitive. We observe that $(1,4)$ and $(4,3)$ are in $\boldsymbol{R}$, but $(1,3) \notin \boldsymbol{R}$. Also $(3,2)$ and $(2,3)$ are in $\boldsymbol{R}$, but $(3,3) \notin \boldsymbol{R}$, and $(4,3)$ and $(3,2)$ are in $\boldsymbol{R}$, but $(3,3) \notin \boldsymbol{R}$. If we add $(1,3)$, $(3,3)$ and $(4,2)$ to $\boldsymbol{R}$ we get

$$\boldsymbol{R}_1 = \{(1,1),(1,3),(1,4),(2,2),(2,3),(3,2),(3,3),(4,2),(4,3)\}.$$

Now we notice that $\boldsymbol{R}_1$ is not transitive since $(1,3)$ and $(3,2)$ are in $\boldsymbol{R}$, but $(1,2) \notin \boldsymbol{R}$. We now add $(1,2)$ to $\boldsymbol{R}_1$ to get

$$\boldsymbol{R}_2 = \{(1,1),(1,2),(1,3),(1,4),(2,2),(2,3),(3,2),(3,3),(4,2),(4,3)\}.$$

A careful check show that $\boldsymbol{R}_2$ is transitive. We illustrate this procedure using the digraphs of $\boldsymbol{R}, \boldsymbol{R}_1$ and $\boldsymbol{R}_2$ in Figure 1.26.

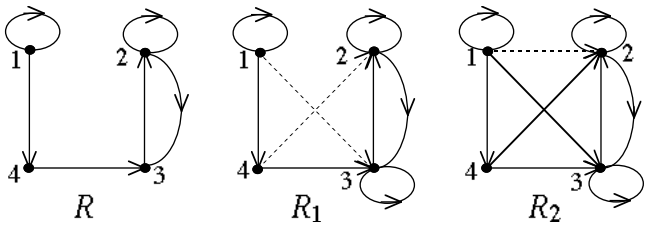


Figure 1.26: Forming the transitive closure of a relation

From the graphs we see that when we have two sides of a triangle we need to add the third side, if it is not there already. Alternatively we can think of it as adding "shortcuts". We need to continue this process until we can't add any more shortcuts. Also if we have (a,b) and (b,a) in the relation we need to add (a,a) if it is not already there. This example shows that constructing the transitive closure of a relation is more complicated than constructing the reflexive or symmetric closure of the relation. Apart from perhaps requiring several steps, it may also be difficult to decide when we have completed the task. In the following sections we examine this construction in more detail.

Paths in Directed Graphs

We have used directed graphs (digraphs) to illustrate the construction of a transitive closure. It is often helpful to do this, but we first recall some definitions.

24 Definition. *Let G be a directed graph, and let a and b be vertices of G. A* walk *from a to b is a sequence of one or more edges $(x_0,x_1),(x_1,x_2),\ldots,(x_{n-1},x_n)$ of G, where $x_0 = a$ and $x_n = b$. The walk is denoted by $x_0,x_1,\ldots,x_n$, and has length n. A walk that begins and ends at the same vertex is called a* cycle.
A walk with no repeated vertices is called a path *and a cycle with no repeated vertices (except first and last) is called a* circuit.

In the digraph shown below (Figure 1.27) some walks are a,b,e,d (path length 3); a,a,f,b,c (walk length 4); e,d,f,b,e,d,f,f,a,b,e (cycle length 10); a,b,e,d,f,a (circuit length 5).

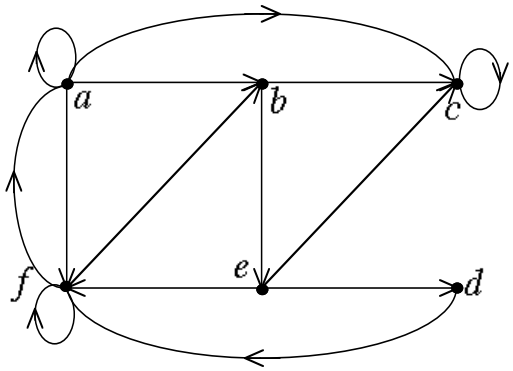


Figure 1.27: A directed graph

Note that some authors do not distinguish between 'walks' and 'paths'. This is partly because if there is a walk from a to $b \neq a$ there must also be a path from a to b, obtained by omitting any cycles which form part of the walk. A similar issue relates to 'cycles' and 'circuits'.

The term *walk* can also be applied to relations.

25 Definition. *Let $\boldsymbol{R}$ be a relation on the set A. Consider $a,b \in A$. We say that there is a* walk *from a to b in the relation $\boldsymbol{R}$ if there is a sequence of elements $a = x_0, x_1, x_2, \ldots, x_{n-1}, b = x_n$ with $(a,x_1) \in \boldsymbol{R}$, $(x_1,x_2) \in \boldsymbol{R}, \ldots, (x_{n-1},b) \in \boldsymbol{R}$.*

26 Definition. *Let $\boldsymbol{R}$ be a relation on the set A. Then for $n \in \mathbb{N}$, $\boldsymbol{R}^n$ contains all pairs (a,b) where there is a walk from a to b in $\boldsymbol{R}$ of length n.*

Three Examples illustrating $\boldsymbol{R}^n$

1. Let $\boldsymbol{R} = \{(1,1),(2,3),(3,1),(4,2)\}$. The digraph of $\boldsymbol{R}$ is shown as Figure 1.28 below.

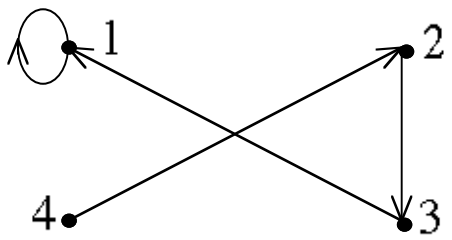


Figure 1.28: Digraph for $\boldsymbol{R} = \{(1,1),(2,3),(3,1),(4,2)\}$

(i) $\boldsymbol{R}^1 = \boldsymbol{R}$. This contains all walks of length 1.

(ii) $\boldsymbol{R}^2 = \{(1,1),(2,1),(3,1),(4,3)\}$. This contains all the walks in $\boldsymbol{R}$ of length 2. The pair $(1,1) \in \boldsymbol{R}^2$ because we can go around the loop at 1 twice, making a walk of length 2 from 1 to 1. The walk of length 2 from 2 to 1 follows the edges from 2 to 3 and from 3 to 1, and the walk from 4 to 3 follows the edges from 4 to 2 and from 2 to 3. The walk from 3 to 1 of length 2 goes from 3 to 1, and then goes around the loop at 1.

(iii) $\boldsymbol{R}^3 = \{(1,1),(2,1),(3,1),(4,1)\}$. This contains the walks in $\boldsymbol{R}$ of length 3. The walk from 1 to 1 of length 3 just goes around the loop at 1 three times. The walk from 2 to 1 goes from vertex 2 to vertex 3, from vertex 3 to vertex 1, and then around the loop at vertex 1. There are similar explanations for (3, 1) and (4, 1).

(iv) $\boldsymbol{R}^4 = \{(1,1),(2,1),(3,1),(4,1)\}$. This is the same as $\boldsymbol{R}^3$. Since all the length 3 walks end at 1, we get a length 4 walk by going around the loop at 1 an extra time.

From this we can see that for $n \geq 3$ the walks of length n are exactly the same as the walks of length 3, and are constructed by going around the loop at 1 the appropriate number of times. So $\boldsymbol{R}^n = \boldsymbol{R}^3$ for $n \geq 3$.

2. Let $\boldsymbol{R} = \{(1,1),(2,1),(2,3),(3,1),(4,1),(4,2),(4,3)\}$ The digraph of $\boldsymbol{R}$ is shown as Figure 1.29 below.

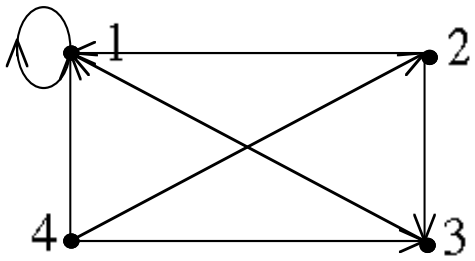


Figure 1.29: Digraph for $\boldsymbol{R} = \{(1,1),(2,1),(2,3),(3,1),(4,1),(4,2),(4,3)\}$

(i) $\boldsymbol{R}^1 = \boldsymbol{R}$.

(ii) $\boldsymbol{R}^2 = \{(1,1),(2,1),(3,1),(4,1),(4,3)\}$. Note there is a walk of length 1 from 4 to 3, but there is also a walk of length 2 that goes from 4 to 2 and from 2 to 3. There are 2 walks of length 2 from 4 to 1, one from 4 to 1 and around the loop at 1, and another from 4 to 2 and from 2 to 1.

(iii) $\boldsymbol{R}^3 = \{(1,1),(2,1),(3,1),(4,1)\}$. The reasoning is much the same as the reasoning in the example 1 above.

(iv) $\boldsymbol{R}^4 = \boldsymbol{R}^3$.

Again we have the situation that for $n \geq 3$, $\boldsymbol{R}^n = \boldsymbol{R}^3$. In this example we also have that $\boldsymbol{R}^n \subseteq \boldsymbol{R}$ for all $n \geq 1$. This indicates that $\boldsymbol{R}$ is a transitive relation.

3. Let $\boldsymbol{R}$ be the relation on the set $\mathcal{H}$ of all humans that contains (a,b) if person a has met person b. We may well be interested in $\boldsymbol{R}^n$, where n is an integer, $n \geq 1$. We note that the relation $\boldsymbol{R}^2$ contains (a,b) if there is a person c such that $(a,c) \in \boldsymbol{R}$ and $(c,b) \in \boldsymbol{R}$. This means that $(a,b) \in \boldsymbol{R}^2$ if a and b have a mutual acquaintance. Similarly, $\boldsymbol{R}^3$ contains (a,b) if there are people c and d such that $(a,c),(c,d),(d,b) \in \boldsymbol{R}$. For $n > 3$ we just get a longer chain. Many conjectures have been made about this relation, and how large n needs to be before any individual is connected to any other individual. See for example "six degrees of separation": http://whatis.techtarget.com/definition/six-degrees-of-separation

It is also helpful to find the matrices for $\boldsymbol{R}^2, \boldsymbol{R}^3, \ldots$. But we first define a different product of matrices. All the matrices describing a relation have entries which consist of 0's and 1's and we want to define a product that gives us again a matrix whose entries have only 0's and 1's. Such matrices are called *Boolean matrices*.
For example let A and B be the boolean matrices below:

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad \text{and} \qquad B = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

Since A is a 3×4 matrix and B is a 4×2 matrix we can multiply A and B with the usual matrix multiplication to get the 3×2 matrix

$$AB = \begin{bmatrix} 0 & 2 \\ 1 & 2 \\ 1 & 1 \end{bmatrix}.$$

We define the Boolean product of A and B, written $A \odot B$, to be the matrix derived from the product matrix AB by leaving the 0 entries unchanged, and replacing any positive integer with a 1. Hence for our example

$$A \odot B = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

We will use this product mainly to find the Boolean product of a matrix with itself, or with repeated Boolean products of itself. In Example 1 above the matrix associated with the relation $\boldsymbol{R}$ is

$$M_R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

If we put $M^{[2]} = M \odot M$, and $M^{[3]} = M^{[2]} \odot M$, then

$$M_R^{[2]} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \text{and} \quad M_R^{[3]} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

What we need to notice here is that $M_R^{[2]}$ is exactly the matrix associated with $\boldsymbol{R}^2$ and $M_R^{[3]}$ is the matrix associated with $\boldsymbol{R}^3$.

27 Definition. *Let M be a Boolean matrix. Put $M^{[1]} = M$, $M^{[2]} = M \odot M$ and $M^{[n]} = M^{[n-1]} \odot M$, for $n \geq 2$.*

It is then always true that if M is the matrix associated with a relation $\boldsymbol{R}$, then $M^{[n]}$ is the matrix associated with $\boldsymbol{R}^n$, the paths of length n in $\boldsymbol{R}$. This means that by finding the repeated Boolean product of a matrix with itself, we can find the walks of any give length in the relation.

Finding the Transitive Closure of a Relation

We now combine the ideas of a path in a relation, and the powers of a relation and its associated matrix to determine the transitive closure of a relation.

28 Definition. *Let $\boldsymbol{R}$ be a relation on a set A. The relation given by*

$$R^\star = \{(a,b) \in A \times A : \textit{ there is a walk in } \boldsymbol{R} \textit{ from } a \textit{ to } b\}$$

is called the connectivity relation *on A with respect to $\boldsymbol{R}$.*

Note that if $a \neq b$ 'walk' can be replaced by 'path' in definition 28 above, since any walk from a to b can be refined to a path from a to b. If $a = b$ then 'walk' can be replaced by 'circuit' for similar reasons.

Example

Let $A = \{1,2,3,4,5\}$, and let $\boldsymbol{R} = \{(1,2),(1,5),(2,3),(2,4),(4,3),(4,5),(5,4)\}$. Then

$$\boldsymbol{R}^\star = \{(1,2),(1,3),(1,4),(1,5),(2,3),(2,4),(2,5),\\ (4,3),(4,4),(4,5),(5,3),(5,4),(5,5)\}.$$

Note that $\boldsymbol{R}^\star$ is transitive. Check it!

The fact that $\boldsymbol{R}^\star$ is transitive in the example above is not by chance, because walks in $\boldsymbol{R}$ comprise precisely the members of $\boldsymbol{R}$ (walks of length 1) and the shortcuts (walks of length > 1) needed for transitivity. So:

29 Theorem. *Let $\boldsymbol{R}$ be a relation on a set A. Then the transitive closure of $\boldsymbol{R}$ is $\boldsymbol{R}^\star$.*

Since $\boldsymbol{R}^n$ comprises all walks in $\boldsymbol{R}$ of length n we have that

30 Theorem. *If $\boldsymbol{R}$ is a relation on a set A, then* $\boldsymbol{R}^\star = \bigcup_{n=1}^{\infty} \boldsymbol{R}^n$.

Theorems 29 and 30 above give us a formula for $t\boldsymbol{R}$ but it is not practical as it involves an infinite union. But for finite sets we can replace this infinite union with a finite one by observing that any walk in a digraph can be refined to a path or circuit by omitting all sections that 'double back'. Since paths and circuits have no repeated vertices (other than first and last vertices of a circuit) we get

31 Lemma. *Let A be a finite set with n elements, and let $\boldsymbol{R}$ be a relation of A.*
If $a, b \in A$ with $a \neq b$ and there is a walk in $\boldsymbol{R}$ from a to b then there is a path in $\boldsymbol{R}$ of length at most $n-1$ from a to b.
If there is a cycle in $\boldsymbol{R}$ from a back to a, then there is a circuit in $\boldsymbol{R}$ of length at most n from a back to a.

The importance of this result is that if $\boldsymbol{R}$ is a relation on a finite set A and A has n elements, then $\boldsymbol{R}^\star = \cup_{i=1}^{n} \boldsymbol{R}^i$. This is a finite union of sets and makes calculation possible. One approach is to make use of the matrices $M_R^{[i]}$ introduced in Definition 27. We first need a further operation on such boolean matrices:

32 Definition. *Let $A = [a_{ij}]$ and $B = [b_{ij}]$ be Boolean matrices of the same size. Then the* join *of A and B, written $A \vee B$, is the Boolean matrix whose (i, j) entry is $a_{ij} \vee b_{ij}$, where $0 \vee 0 = 0$, and $1 \vee 0 = 0 \vee 1 = 1 \vee 1 = 1$.*

For example let A and B be the following matrices.

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

Then

$$A \vee B = \begin{bmatrix} 1 \vee 0 & 0 \vee 0 & 0 \vee 1 \\ 0 \vee 0 & 1 \vee 1 & 1 \vee 0 \\ 1 \vee 1 & 0 \vee 0 & 0 \vee 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}.$$

For a relation $\boldsymbol{R}$ on a set A with n elements, we have $\boldsymbol{R}^\star = \boldsymbol{R} \cup \boldsymbol{R}^2 \cup \boldsymbol{R}^3 \cup \cdots \cup \boldsymbol{R}^n$. We have also noted that the matrix associated with the relation $\boldsymbol{R}^i$ of path of length i in $\boldsymbol{R}$ is $M_R^{[i]}$. To use this correspondence between the relations and the matrices we need an operation on the matrices that mirrors the union operation on sets. The join operation that we have just described is exactly what we need, leading to

33 Theorem. *Let M_R be the Boolean matrix of the relation $\boldsymbol{R}$ on a set A with n elements. Then the Boolean matrix of the transitive closure of $\boldsymbol{R}$ is given by*

$$M^\star = M \vee M^{[2]} \vee M^{[3]} \vee \cdots \vee M^{[n]}.$$

Two Examples

1. Let $\boldsymbol{R}$ be the relation on the set $A = \{1,2,3\}$ given by $\boldsymbol{R} = \{(1,1),(2,3),(3,1),(3,2)\}$. The matrix of $\boldsymbol{R}$ is given by

$$M_R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

From this we can calculate that

$$M_R^{[2]} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad M_R^{[3]} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

Since A has 3 elements, we use the theorem to find $M_R^\star$ by finding the join of M_R, $M_R^{[2]}$ and $M_R^{[3]}$. Thus

$$M_R^\star = M_R \vee M_R^{[2]} \vee M_R^{[3]} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \vee \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \vee \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

From the matrix $M_R^\star$ we find $\boldsymbol{R}^\star = \{(1,1),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)\}$.

2. Let $\boldsymbol{R}$ be the relation on $\mathbb{N}$, the set of natural numbers given by

$$x\boldsymbol{R}y \leftrightarrow y = x+1.$$

We investigate the closures on $\boldsymbol{R}$. It is perhaps helpful to write

$$\boldsymbol{R} = \{(1,2),(2,3),(3,4),(4,5),\ldots\}.$$

Then $r\boldsymbol{R}$, the reflexive closure of $\boldsymbol{R}$ is given by

$$r\boldsymbol{R} = \{(1,1),(1,2),(2,2),(2,3),(3,3),(3,4)\ldots\}$$

or

$$x\ r\boldsymbol{R}\ y \leftrightarrow y = x+1 \text{ or } y = x.$$

The symmetric closure $s\boldsymbol{R}$ is given by

$$s\boldsymbol{R} = \{(1,2),(2,1),(2,3),(3,2),(3,4),(4,3),\ldots\}$$

or alternatively

$$x\ s\boldsymbol{R}\ y \leftrightarrow y = x+1 \text{ or } x = y+1 \leftrightarrow |x-y| = 1.$$

Since $\mathbb{N}$ is not a finite set we can't use the methods described above to calculate the transitive closure of $\boldsymbol{R}$. However, it is clear that the transitive closure $t\boldsymbol{R}$ of $\boldsymbol{R}$ contains the pair (x,y) if there is a chain $x = x_0, x_1 = x_0+1, x_2 = x_1+1, \ldots, y = x_n = x_{n-1}+1$. This is equivalent to saying that $(x,y) \in t\boldsymbol{R}$ if and only if $x < y$.

From here we can calculate that $(x,y) \in st\boldsymbol{R}$ if and only if $x < y$ or $y < x$. On the other hand, $(x,y) \in ts\boldsymbol{R}$ if and only if $x < y, y < x$ or $x = y$. That is, $ts\boldsymbol{R} = \mathbf{N} \times \mathbf{N}$. It is a good exercise to investigate $rt\boldsymbol{R}, tr\boldsymbol{R}, rst\boldsymbol{R}, trs\boldsymbol{R}$ and other combinations of r, s and t.

Theorem 33 leads to an algorithm for computing the matrix of the relation $t\boldsymbol{R} = \boldsymbol{R}^\star$. If M_R is the $n \times n$ matrix associated with $\boldsymbol{R}$. To find $M_R^\star$ the successive Boolean powers of M_R are calculated. As each power is calculated, its join with the join of all smaller powers is computed. When this is done with the nth power, the matrix $\boldsymbol{R}^\star$ has been computed.

Algorithm – Transitive Closure

Procedure *transitive closure* (M, a zero-one $n \times n$ matrix.)
$A := M$
$B := A$
for $i = 2$ **to** n
begin
  $A := A \odot M$
  $B := B \vee A$
end $\{B$ is the Boolean matrix for $\boldsymbol{R}^\star.\}$

Finding each Boolean product requires n^3 bit operations. To find the Boolean powers we must find $n-1$ Boolean products, so the number of bit operations to find the Boolean powers is $n^3(n-1)$. To find the Boolean matrix of $\boldsymbol{R}^\star$, we must calculate $n-1$

joins of Boolean matrices. To find each of these joins requires n^2 operations. Thus the total number of bit operations to find the Boolean matrix of $\boldsymbol{R}^\star$ is

$$n^3(n-1)+n^2(n-1)=n^4-n^2=O(n^4).$$

There is a more efficient method, known as Warshalls Algorithm, for computing the transitive closure of a relation on a set with n members. Warshalls algorithm uses $2n^3$ bit operations.

For some further reading:

http://www.cs.hut.fi/ enu/tc.html,
http://www.informatik.uni-trier.de/ ley/db/deductive/closure.html,
http://www.acm.org/pubs/toc/Abstracts/tods/155273.html,
http://www.cs.hut.fi/ psu/VK94/node28.html,
http://epubs.siam.org/sam-bin/dbq/article/22530,
http://citeseer.nj.nec.com/3813.html,
http://www.cs.sunysb.edu/ algorith/files/transitive-closure.shtml,
http://ftp.cs.rochester.edu/u/leblanc/csc173/graphs/tc.html,
http://cse.hanyang.ac.kr/ jmchoi/class/1998-1/disc-math/note12/node2.html,
http://www.psrg.lcs.mit.edu/ bvelez/std-colls/cacm/cacm-2769.html,
http://www.cs.oberlin.edu/classes/dragn/lectures/graphalg9.html,
http://www.cs.byu.edu/courses/cs236/javalect/GRAPHS8/Graphs8.html,
http://www.csc.tntech.edu/ srini/DM/chapters/review5.3.html.

1.5 Equivalence Relations

One of the most important relations that we study in this chapter is the equivalence relation. An equivalence relation seems similar to a partial ordering, because the only difference is that we require an equivalence relation to be symmetric (rather than anti-symmetric), as well as reflexive and transitive.

34 Definition. *A binary relation on a set A that is reflexive, symmetric and transitive is called an* equivalence relation *on A. An equivalence relation is often denoted by "$\sim$".*

Some equivalence relations that we have already met are

on any set S: $x \sim y \leftrightarrow x = y$,

on $\mathbb{N}$: $x \sim y \leftrightarrow x+y$ is even,

on the set of all lines in the plane: $x \sim y \leftrightarrow x$ is parallel to y, or x coincides with y,

on $\{0,1\}$: $x \sim y \leftrightarrow x = y^2$,

on $A = \{1,2,3\}$: $\boldsymbol{R} = \{(1,1),(1,3),(2,2),(3,1),(3,3)\}$.

Three Further Examples

1. Let $\boldsymbol{R}$ be the relation on $\mathbf{Z} \times \mathbf{N}$ defined by

$$(a,b)\boldsymbol{R}(c,d) \leftrightarrow ad = bc.$$

We show that $\boldsymbol{R}$ is an equivalence relation.
$\boldsymbol{R}$ is *reflexive*: $(a,b)\boldsymbol{R}(a,b)$ because $ab = ba$ by the commutativity of multiplication.
$\boldsymbol{R}$ is *symmetric*: Suppose that $(a,b)\boldsymbol{R}(c,d)$. Then $ad = bc$. By the commutativity of multiplication this is equivalent to $cb = da$ which implies that $(c,d)\boldsymbol{R}(a,b)$.
$\boldsymbol{R}$ is *transitive*: Suppose that $(a,b)\boldsymbol{R}(c,d)$ and $(c,d)\boldsymbol{R}(e,f)$. Then $ad = bc$ and $cf = de$. We want to show that $(a,b)\boldsymbol{R}(e,f)$, or equivalently, $af = be$. From $ad = bc$ we get $adf = bcf$ and from $cf = de$ we get $bcf = bde$. Combining these equalities gives us that $adf = bde$. Since $d \in \mathbb{N}$ and hence $d \neq 0$ we can divide this equality by d to get $af = be$. This is what we wanted.
Note: Members (a,b) of $\mathbb{Z} \times \mathbb{N}$ can be viewed as fractions $\frac{a}{b}$, and in this setting the relation $\boldsymbol{R}$ on $\mathbb{Z} \times \mathbb{N}$ represents equality ($\frac{a}{b} = \frac{c}{d}$ if and only if $ad = bc$).

2. Let $A = \{a,b,c,d,e\}$, and let $\boldsymbol{R} = \{(a,a),(a,b),(b,a),(b,b),(c,c),(c,d),(d,c),(d,d),(e,e)\}$. It is easily checked that $\boldsymbol{R}$ is reflexive, symmetric and transitive, and therefore an equivalence relation. The digraph of this relation is shown as Figure 1.30 below.

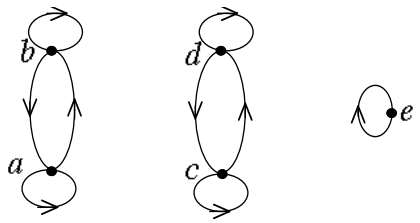


Figure 1.30: Digraph of a relation on $\{a,b,c,d,e\}$

We can see that $\boldsymbol{R}$ separates or partitions A into three subsets or classes $\{a,b\},\{c,d\}$ and $\{e\}$. This partitioning is a characteristic of equivalence relations.

3. Let $\boldsymbol{R}$ be the relation on $\mathbb{Z}$, the set of all integers, defined by

$$x\boldsymbol{R}y \leftrightarrow x-y \text{ is divisible by } 3.$$

We show that $\boldsymbol{R}$ is an equivalence relation.
$\boldsymbol{R}$ is *reflexive*, since for any integer x, $x-x=0$, which is divisible by 3. Hence $x\boldsymbol{R}x$.
$\boldsymbol{R}$ is *symmetric*, since for any integers x and y, if $x-y$ is divisible by 3, then so is $y-x$. So $x\boldsymbol{R}y$ implies $y\boldsymbol{R}x$.
$\boldsymbol{R}$ is *transitive*, for suppose that x,y and z are integers for which $x\boldsymbol{R}y$ and $y\boldsymbol{R}z$. This means that $x-y$ is divisible by 3, so (i) $x-y=3k$ for some integer k, and $y-z$ is divisible by 3, so (ii) $y-z=3j$ for some integer j. We want to show that $x\boldsymbol{R}z$, or equivalently that $x-z$ is divisible by 3. From (i) we get $x=3k+y$ and from (ii) we get $z=y-3j$, and hence

$$x-z=(3k+y)-(y-3j)=3k+3j=3(k+j).$$

Thus $x-z$ is indeed divisible by 3 and so $x\boldsymbol{R}z$, which shows that $\boldsymbol{R}$ is transitive.

Equivalence Classes

In example 3 above let us now group together all the integers that are related to each other. Consider all the integers related to 3. They include $0,3,6,9\ldots,-3,-6,-9,\ldots$ and any integer that is divisible by 3. Notice that all these integers are related to each other. Now consider all the integers related to 4. They include $1,4,7,10,\ldots,-2,-5,-8\ldots$ and any integer that has a remainder of 1 when divided by 3. Again, all these integers are related to each other. Finally, consider all the integers related to 2. They include $2,5,8,11,\ldots,-1,-4,-7\ldots$ and any integer that has a remainder of 2 when divided by 3. These integers are also all related to each other. Using the notation $[a]$ to denote the set of integers related to a, we have

$$\begin{aligned}
[3] &= \{\ldots,-6,-3,0,3,6,9,12,\ldots\} = [0] = [6] = [-15] = \cdots \\
[4] &= \{\ldots,-5,-2,1,4,7,10,13,\ldots\} = [1] = [25] = [-2] = \cdots \\
[2] &= \{\ldots,-4,-1,2,5,8,11,14,\ldots\} = [5] = [-4] = [-7] = \cdots
\end{aligned}$$

Looking at these sets or classes, we see that no two of the classes overlap, and every integer belongs to exactly one of the classes. If we combine all the classes we get all of the integers. We have made a *partition* of the integers.

35 Definition. *Let $\boldsymbol{R}$ be an equivalence relation on a set A. For $a \in A$, the* equivalence class *of a, denoted by $[a]$, is the set of all elements in A that are related to a and is given by*

$$[a] = \{x \in A : a\boldsymbol{R}x\}.$$

Since an equivalence relation is reflexive, it is always true that a is in its own equivalence class, that is, $a \in [a]$. We can also show that if x and y belong in the same equivalence class then $[x] = [y]$. This is illustrated in the last example where [0] = [6] = [-3], and so on.

36 Definition. *A* partition *of a set A is a collection of non-empty disjoint subsets of A whose union is equal to A. More formally, a* partition *of A is a collection of subsets $X_1, X_2, \ldots, X_k$ where $X_i \neq \emptyset$ for $i = 1, \ldots k$, $X_i \cap X_j = \emptyset$ for $1 \leq i, j \leq k$ and $A = \cup_{i=1}^{k} X_i$.*

There are two important results related to equivalence classes. The first generalises the picture we saw in examples 2 and 3 above:

37 Theorem. *Let $\boldsymbol{R}$ be an equivalence relation on the set A. Then the equivalence classes of $\boldsymbol{R}$ form a partition of A.*

The proof of theorem 37 relies heavily on the reflexivity, symmetry and transitivity properties of any equivalence relation. The converse of Theorem 37 is also true; if we start with a partition we can make an equivalence relation from it.

As an example take $A = \{1,2,3,4,5,6,7,8\}$. Then a partition of A is given by $X_1 = \{1,2,5\}, X_2 = \{3,4,8\}$ and $X_3 = \{6,7\}$. Consider X_1. We want it to be an equivalence class for a relation $\boldsymbol{R}$, and so we want each element in X_1 to be related to itself and to every other element in X_1. So $\boldsymbol{R}$ must contain $(1,1), (1,2), (1,5), (2,1), (2,2), (2,5), (5,1)$, $(5,2)$ and $(5,5)$. Similarly for X_2 to be an equivalence class, $\boldsymbol{R}$ must contain $(3,3), (3,4)$, $(3,8), (4,3), (4,4), (4,8), (8,3), (8,4)$ and $(8,8)$. Finally for X_3 to be an equivalence class $\boldsymbol{R}$ must contain $(6,6), (6,7), (7,6)$ and $(7,7)$. If we now put

$$\boldsymbol{R} = \{(1,1),(1,2),(1,5),(2,1),(2,2),(2,5),(5,1),(5,2),(5,5)\} \cup$$
$$\{(3,3),(3,4),(3,8),(4,3),(4,4),(4,8),(8,3),(8,4),(8,8)\} \cup \{(6,6),(6,7),(7,6),(7,7)\},$$

that is $\boldsymbol{R} = \{(1,1),(1,2),(1,5),(2,1),(2,2),(2,5),(5,1),(5,2),(5,5,),(3,3),(3,4),$ $(3,8),(4,3),(4,4),(4,8),(8,3),(8,4),(8,8),(6,6),(6,7),(7,6),(7,7)\}$ we find that $\boldsymbol{R}$ is reflexive, symmetric and transitive, and is therefore an equivalence relation. The graph for $\boldsymbol{R}$ is shown in Figure 1.31 below.

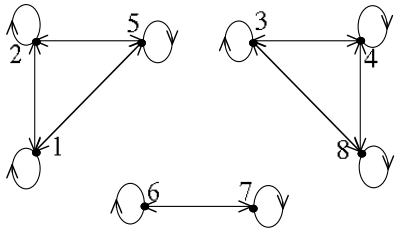


Figure 1.31: Digraph of a relation induce by a partition

Notice that the edges between the vertices have arrows at both ends indicating the symmetry of the relation. The following theorem confirms that it is always possible to construct an equivalence relation from a partition of a set, as in this example.

38 Theorem. *Let A be a finite set with a partition $A = X_1 \cup X_2 \cup \cdots \cup X_n$. Then there is an equivalence relation on A whose equivalence classes are the sets $X_1, X_2, \ldots, X_n$.*

We can use the connection between equivalence relations and partitions to count the number of equivalence relations on a set. No two partitions on a set define the same equivalence relation, and no two equivalence relations on a set define the same partition, so instead of counting equivalence relations we count the number of partitions of a set.

For example, let $A = \{a, b, c\}$ be a set with three elements. The partitions of A and the corresponding equivalence relations are given in Table 1.3 below.

Table 1.3: Partitions and Equivalence Relations

Partition	Equivalence Relation
$X_1 = \{a, b, c\}$	$\boldsymbol{R} = \{(a,a),(a,b),(a,c),(b,a),(b,b),(b,c),(c,a),(c,b),(c,c)\}$
$X_1 = \{a, b\}, X_2 = \{c\}$	$\boldsymbol{R} = \{(a,a),(a,b),(b,a),(b,b),(c,c)\}$
$X_1 = \{a, c\}, X_2 = \{(b)\}$	$\boldsymbol{R} = \{(a,a),(a,c),(b,b),(c,a),(c,c)\}$
$X_1 = \{a\}, X_2 = \{b, c\}$	$\boldsymbol{R} = \{(a,a),(b,b),(b,c),(c,b),(c,c)\}$
$X_1 = \{a\}, X_2 = \{b\}, X_3 = \{c\}$	$\boldsymbol{R} = \{(a,a),(b,b),(c,c)\}$

Since there are 5 partitions of A there are 5 equivalence relations on A.

The task of counting the number of partitions of a set has been studied extensively, and the number of partitions of a set with n elements is called the *Bell number* B_n. It is easy to see that $B_1 = 1, B_2 = 2$ and we have $B_3 = 5$. The size of the remaining numbers grows very quickly. For further reading on the Bell Numbers see the following references:

http://mathworld.wolfram.com/BellNumber.html
http://en.wikipedia.org/wiki/Bell_numbers

1.6 n-ary Relations and Relational Databases

Relationships between elements of more than two sets often arise. When n sets are involved (counting copies, if present) such relationships are called ***n*-ary relations** (2-ary= binary; 3-ary = ternary; 4-ary = quaternary; etc.). These relations are used in computer databases. They help us to answer questions about the information stored in databases.

n-ary Relations

The definition of an n-ary relation is an extension of the definition of a binary relation.

39 Definition. *Let $A_1, A_2, \ldots, A_n$ be sets. An **n-ary** relation on these sets is a subset of $A_1 \times A_2 \times \ldots \times A_n$. The sets $A_1, A_2, \ldots, A_n$ are called the **domains** of the relation, and n is called its **degree**. The members of an n-ary relation are called **n-tuples**. (2-tuple = ordered pair; 3-tuple = ordered triple; 4-tuple = ordered quadruple; etc.)*

Examples

1. Let $\boldsymbol{R}$ be the relation consisting of triples (a, b, c) where a, b, c are integers and $a > b > c$. Then $(5, 3, 2) \in \boldsymbol{R}$, but $(1, 3, 2) \notin \boldsymbol{R}$. This $\boldsymbol{R}$ is a ternary relation with all three domains equal to $\mathbb{Z}$.

2. Let H be the set of all humans. Define $\boldsymbol{B} \subseteq H \times H \times H$ to be the set of all ordered triples (f, m, c), where f is the biological father of c, m is the biological mother of c, and f and m are currently married. Since it is possible for two parents to have more than one child, this relation can't be interpreted as a function of two variables.Let $\boldsymbol{A} \subseteq H \times H \times H$ to be the set of all ordered triples (f, m, c), where f is the adoptive father of c, m is the adoptive mother of c, and f and m are currently married. The relations $\boldsymbol{B}$ and $\boldsymbol{A}$ are disjoint.

3. Suppose that an airline *APA*, say, has flight *AP*003 from Dhaka to Khulna at 1530 hours. In this and similar statements for other flights we have several bits of information:

   - The airline's name *(A)*
   - Flight number *(F)*
   - Starting point of the flight *(S)*
   - Destination of the flight *(D)*
   - Time for departure (T)

   For each flight this information can be put into a 5-tuple *(A, F, S, D, T)* and the set of all these 5-tuples is a 5-*ary relation* $\boldsymbol{R}$ that can be used as a database. Then (*APA, AP*003, Dhaka, Khulna, 1530) $\in \boldsymbol{R}$. The degree of this relation is 5, and the domains are the set of all airlines, the set of flight numbers, the set of originating cities, the set of destination cities (this is likely to be the same as the set of originating cities) and the set of times.

Relational Databases

One of the most significant applications of n-ary relations is to provide the theoretical model for relational databases. A database is a collection of related information, generally managed through the use of computers. It is a convenient way to store data where one can add new records, update existing records, delete a record, search for information, combine some information for specific requirements, and do various other similar manipulations. EF Codd originally introduced the relational database model in 1970. The theoretical model is seen in the name *relational database*. However, much of the terminology used in the discussion of relational databases includes words that are influenced by the need to use computers in the implementation of the data base model. Using traditional terminology, a database consists of **files** of **records**. Each record is a list of a fixed number, say n, of **entries**. Each entry comes from a specific *field*. The names of these fields are called *attributes*. So the records may be considered as n-tuples in an n-ary relation (one relation per file) and the fields are the domains of this relation. There is another view that emphasises the matrix-like structure that can be easily imposed on the relation: a two-dimensional table with the ordered n-tuples as *rows* and with the coordinate positions (fields) aligned as *columns*.
As an example, a database file of student records may have attributes name, student

number, course and grade for that course. The student records are represented as 4-tuples of the form (*STUDENT NAME, STUDENT ID, COURSE, GRADE)*. A sample of 5 such records is

(Anderson, u23145, MATH2301, P)

(Arnold, u31567, MATH2303, CR)

(Connor, u54367, MATH1013, HD)

(Lamont, u233148, MATH2301, N)

(Skinner, u23148, MATH2303, P)

This same information could be displayed in the table (matrix) form shown in Table 1.4 below.

Table 1.4:

Student Name	**Student ID**	**Course**	**Grade**
Anderson	U23145	MATH2301	P
Arnold	U31567	MATH2303	CR
Connor	U54367	MATH1013	HD
Lamont	U67321	MATH2301	N
Skinner	U23148	MATH2303	P

A domain of an n-ary relation is called a **primary key** when the value of the n-tuple from this domain determines the n-tuple. So a domain is a **primary key** when no two n-tuples in the relation have the same value in this domain. In the example above, the domain of student names is a primary key, as is the domain of student ids. The domain of courses and the domain of grades are not primary keys. If more records are added in the future the situation about primary keys may change — for example the database may contain then multiple records for the same student (who has taken several courses). In fact records are often added or deleted from databases. Because of this the property that a domain be a primary key is time dependent. Consequently,

a primary key should be chosen that remains one even if the database is changed. For the example above, Table 1.5 shows the result of adding another record: (Anderson, U23145, MATH2303, CR).

Table 1.5:

Student Name	**Student ID**	**Course**	**Grade**
Anderson	U23145	MATH2301	P
Anderson	U23145	MATH2303	CR
Arnold	U31567	MATH2303	CR
Connor	U54367	MATH1013	HD
Lamont	U67321	MATH2301	N
Skinner	U23148	MATH2303	P

We now have a problem in determining a primary key. In this case we need a combination of domains to uniquely identify 4-tuples in this relation. Here, we could choose student name and course, or student id and course. The Cartesian product of these domains is called a **composite key**. Since primary and composite keys are used to identify records uniquely in a database, it is important that the keys remain valid when new records are added to the database. Hence every new record should be checked to ensure that it has values that are different in the appropriate field, or fields, from all other records in this table. In the example above, it would be best to choose student id and course as the composite key, as it is possible that two different students could have the same family name.

Operations on Databases

There are a variety of operations on n-ary relations that can be used to form new relations. We discuss here two operations, the *projection* and the *join* operation.

40 Definition. *The* projection $P_{i_1,i_2,\ldots,i_m}$ *maps the n-tuple* $(a_1,a_2,\ldots,a_n)$ *to the m-tuple* $(a_{i_1},a_{i_2},\ldots,a_{i_m})$*, where* $m \leq n$*.*

The projection $P_{1,3,5}$ applied to the 6-tuple $(1,4,2,5,2,1)$ deletes three of the entries (the second, fourth and sixth) leaving (1, 2, 2), the first, third and fifth entries.
For a database example consider Table 1.6 below.

Table 1.6:

Airline	**Flight Number**	**Gate**	**Destination**	**Departure Time**
Virgin	122	4	Sydney	06:30
Qantas	221	3	Brisbane	06:40
Qantas	123	2	Sydney	07:10
Qantas	323	3	Adelaide	07:30
Virgin	199	4	Sydney	10:25
Qantas	324	5	Brisbane	11:10
Virgin	331	4	Sydney	12:15

By applying the projection $P_{1,2,4}$ to Table 1.6 we get Table 1.7.

Table 1.7:

Airline	**Flight Number**	**Destination**
Virgin	122	Sydney
Qantas	221	Brisbane
Qantas	123	Sydney
Qantas	323	Adelaide
Virgin	199	Sydney
Qantas	324	Brisbane
Virgin	331	Sydney

When a projection is applied to a table (relation, database file), it is possible that fewer rows (n-tuples, records) will result than in the original. This happens when some of the rows in the table have identical values in each of the columns retained by the projection. For example, when the projection $P_{1,2}$ is applied to Table 1.8, the result is Table 1.9.

Table 1.8:

Student	**Major**	**Course**
Brown	Physics	MATH1116
Brown	Physics	PHYS1101
Jones	Mathematics	MATH1116
Jones	Mathematics	SRES1001

Table 1.9:

Student	**Major**
Brown	Physics
Jones	Mathematics

The *join* operation is used to combine two tables into one when these tables share some attributes. For example, a database containing attributes airline, flight number, and gate, and another containing flight number, gate and departure time could be combined using the *join* operation to give a table with attributes airline, flight number, gate and departure time.

As another example consider Tables 1.10 and 1.11 below relating to biological and adoptive children of a few parents.

Table 1.10: Biological

Father	**Mother**	**Biological Child**
John Smith	Jane Smith	William Smith
John Smith	Jane Smith	Susan Smith
Esteban Rodriguez	Anita Rodriguez	Pablo Rodriguez
Walter Leblanc	Miranda Leblanc	Wanda Leblanc
Robin Westlund	Virginia Westlund	Derwin Westlund
Robin Westlund	Virginia Westlund	Darwin Westlund

Table 1.11: Adoptive

Father	**Mother**	**Adoptive Child**
John Smith	Jane Smith	Polly Smith
John Smith	Jane Smith	Carmen Smith
Esteban Rodriguez	Anita Rodriguez	Tran-minh Rodriguez
Isaac Levitz	Helen Levitz	Aaron Levitz
Isaac Levitz	Helen Levitz	Hanna Levitz
Bob Jones	Betty Jones	Samantha Jones

The join of the Biological and Adoptive Tables is essentially the relation containing all pairs of biological and adoptive children who have the same parents. More precisely, the join is the set of all 4-tuples in the set "Father $\times$ Mother $\times$ Biological Child $\times$ Adoptive Child" for which the projection onto {Father, Mother, Biological Child} is a 3-tuple in Biological, and the projection onto {Father, Mother, Adoptive Child} is a 3-tuple in Adoptive. This rules out 4-tuples like (Isaac Levitz, Helen Levitz, Wanda

Lablanc, Aaron Levitz) because the projection onto{Father, Mother, Biological Child} is given by (Isaac Levitz, Helen Levitz, Wanda Lablanc) which is not in Biological. This rule also rules out any 4-tuple containing Robert Westlund because Robert has no adopted child. This means that the only sets of parents in the join of Biological and Adoptive are John and Jane Smith and Esteban and Anita Rodriguez. The entries for John and Jane Smith are quite interesting. They have two biological children and two adoptive children. The join requires that we include all possible ordered pairs of biological and adoptive children who have the same parents. Thus the join Biological $\star$ Adoptive is given by Table 1.12.

Table 1.12: Biological $\star$ Adoptive

Father	**Mother**	**Biological Child**	**Adoptive Child**
John Smith	Jane Smith	William Smith	Polly Smith
John Smith	Jane Smith	William Smith	Carmen Smith
John Smith	Jane Smith	Susan Smith	Polly Smith
John Smith	Jane Smith	Susan Smith	Carmen Smith
Esteban Rodriguez	Anita Rodriguez	Pablo Rodriguez	Tran-minh Rodriguez

The definition below goes some way towards a formal, mathematical, definition of the join operation for relations. It is restricted to a case in which the domains of the relations are in very convenient orders. For example, to apply the definition to the preceding example about biological and adoptive children it would be necessary to move the "Biological Child" column to the front in both Tables 1.10 and 1.12. The definition would then apply with $m = n = 3$ and $p = 2$.

41 Definition. *Let R be a relation of degree m and S a relation of degree n. The* ***join*** *$J_p(R,S)$, where $p \leq m$ and $p \leq n$, is a relation of degree $m+n-p$, that consists of all $(m+n-p)$-tuples $(a_1,a_2,\ldots,a_{m-p},c_1,c_2,\ldots,c_p,b_1,b_2,\ldots,b_{n-p})$ where the m-tuple $(a_1,a_2,\ldots,a_{m-p},c_1,c_2,\ldots,c_p)$ belongs to R and the n-tuple $(c_1,c_2,\ldots,c_p,b_1,b_2,\ldots,b_{n-p})$ belongs to S.*

In practice, we decide which fields are common to both relations. The fields in the relation produced by the join of relations R and S, say, consist of the fields that belong to the relation R only, the fields that belong to both R and S and the fields that belong to S only. The index p in the notation $J_p(R,S)$ refers to the number of fields that are common to both relations. The entries in the new relation must all be valid (suitably reordered as necessary) when restricted to either R or S.
As a final example consider the join of the two relations specified by Tables 1.13 and 1.14.

Table 1.13: Yearbook

Student	**Task**	**Homeroom**
Joe	Advertisements	H4
Martha	Activities	G3
Kim	Teacher Photos	H4
Rosa	Student Photos	D5

Table 1.14: Newspaper

Student	**Feature**	**Cohort**
Amelia	News	1st Year
Wesley	City Page	2nd Year
Kim	Photos	3rd Year
Rosa	Editorials	1st Year
Bob	Sports	2nd Year

The join of these relations will contain the domains (fields) Student (which is common to both relations), Task, Homeroom, Feature and Cohort. The only students who have the four attributes Task, Homeroom, Feature and Homeroom are Kim and Rosa, so

they are the only ones who will appear in the join of the two relations. The table representing the join of the two relations is Table 1.15.

Table 1.15: Yearbook $\star$ Newspaper

Student	**Task**	**Homeroom**	**Feature**	**Cohort**
Kim	Teacher Photos	H4	Photos	3rd Year
Rosa	Student Photos	D5	Editorials	1st Year

There are other operations that are useful on a relational data base. Of particular importance are the design of a database, and the normal forms which aim to ensure consistency and stability, to minimise data redundancy to ensure consistent updatability and maintability of the database and to avoid update and delete anomalies that result in ambiguous or inconsistent data. You will need to do extra reading to learn about these forms.

Database References

Mannila, H &, Räihä, K 1992, *The design of relational databases*, Addison-Wesley, Reading Mass.
Carter, J 1995, *The relational database*, International Thomson Computer Press, London.

END OF CHAPTER ONE

2. FOUNDATIONS

This chapter contains two topics, functions and modular arithmetic, that belong very much to the foundations of the mathematics we are using. In some ways they will be disconnected, but they should be useful in all the mathematics that you do.

2.1 Functions

Recall that a function with 'signature' $f : A \to B$ is a (binary) relation f on $A \times B$ with the property that for each $a \in A$ there is exactly one $b \in B$ for which $(a,b) \in f$. We write $f(a) = b$. We call A the **domain** of f and B the **codomain** of f.
In this section we revise some of the properties of functions and study some special functions. The main properties that we will be concerned with are that of being *one-to-one* and that of being *onto*. To help discuss these properties, consider all the functions $\{0,1,2\} \to \{a,b\}$. There are eight such functions:

Table 2.1: All Functions from $\{0,1,2\}$ to $\{a,b\}$

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$	$f_7(x)$	$f_8(x)$
0	a	a	a	a	b	b	b	b
1	a	a	b	b	a	a	b	b
2	a	b	a	b	a	b	a	b

A function f is called **one-to-one** or **injective** when, for every x and y in the domain of f, $x \neq y$ implies $f(x) \neq f(y)$. Equivalently, we may say that a function f is one-to-one

provided that $f(x) = f(y)$ always implies that $x = y$. (This is the 'contrapositive' of the first statement.)
Consider f_1 and f_2, say, as arrow diagrams. (See Figure 2.1.)

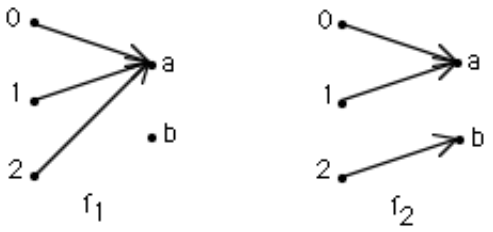


Figure 2.1: Arrow Diagrams for f_1 and f_2.

Neither f_1 nor f_2 is one-to-one since there is more than one arrow going to the point a in both cases. For a function to be one-to-one there can be at most *one* arrow going to any point in the codomain. In fact none of the functions $f_1, f_2, \ldots, f_8$ are one-to-one.

For a function from $f : A \to B$ we say that f is **onto** or **surjective** if for every $b \in B$, there is an element $a \in A$ such that $f(a) = b$. In terms of the arrow diagrams there must be an arrow to every point in the codomain. In the diagram above f_1 is not onto, while f_2 is onto. Also f_3, $f_4 \ldots$, f_7 are onto, but f_8 is not.
Take another simple example. Let $A = \{0,1\}$ and $B = \{a,b,c\}$, and consider all the functions g from A to B. There are nine such functions:

Table 2.2: All functions from $\{0,1\}$ to $\{a,b,c\}$.

x	$g_1(x)$	$g_2(x)$	$g_3(x)$	$g_4(x)$	$g_5(x)$	$g_6(x)$	$g_7(x)$	$g_8(x)$	$g_9(x)$
0	a	a	a	b	b	b	c	c	c
1	a	b	c	a	b	c	a	b	c

This time we can see that none of the functions are onto, while all except g_1, g_5, and g_9 are one-to-one.

In the examples mentioned so far we haven't found any functions that are both one-to-one and onto. However, we can draw some further conclusions from these examples. Let A and B be finite sets, $A = \{a_1, a_2, \ldots, a_m\}$, and $B = \{b_1, b_2, \ldots, b_n\}$, so that $|A| = m$ and $|B| = n$. If $m > n$, then no function from A to B can be one-to-one. If $m < n$ then no function from A to B can be onto. So the only way that we can have functions from A to B that are one-to-one and onto is to have $m = n$. For example take $A = \{0,1,2\}$ and $B = \{a,b,c\}$, and let us find all the one-to-one functions from A to B. There are exactly six of these. They are shown in Table 2.3.

Table 2.3: All one-to-one functions from $\{0,1,2\}$ to $\{a,b,c\}$

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$
0	a	a	b	b	c	c
1	b	c	a	c	a	b
2	c	b	c	a	b	a

What we find is that all these functions are also onto. In fact, they are the only onto functions from A to B. Further, when $m = n$ it is always true that a function $f : A \to B$ is one-to-one if and only if it is onto. (*Warning: This result is not true if A and B are infinite sets.*) A function that is one-to-one and onto is called a **one-to-one correspondence** or **bijection**.

If $f : A \to B$ is a one-to-one correspondence we can define an **inverse function** $f^{-1} : B \to A$. The inverse function is a way of "going backwards". If $f(a) = b$, where $a \in A$ and $b \in B$, we define $f^{-1}(b) = a$. For example let $A = \{1,2,3,4,5,6\} = B$ and let f be the function from A to A given by Table 2.4.
Clearly, f *is* one-to-one since there is no repeated entry in the second row. It is therefore also onto since $|A| = |B|$. Writing f as a set of ordered pairs we have

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

We find the inverse of f simply by reversing each of the ordered pairs in f:

$$f^{-1} = \{(2,1),(3,2),(5,3),(1,4),(6,5),(4,6)\} = \{(1,4),(2,1),(3,2),(4,6),(5,3),(6,5)\}.$$

In table form we can write f^{-1} as in Table 2.5.

Table 2.4: A one-to-one correspondence

x	1	2	3	4	5	6
$f(x)$	2	3	5	1	6	4

Table 2.5: The Inverse of the function defined by Table 2.4

x	1	2	3	4	5	6
$f^{-1}(x)$	4	1	2	6	3	5

We next define composition of functions. For functions, $f : A \to B$ and $g : B \to C$ the **composite** function $g \circ f : A \to C$ is defined by

$$\forall x \in A (g \circ f)(x) = g(f(x)).$$

For example let $A = \{0,1,2\}$, $B = \{a,b,c,d\}$ and $C = \{7,8,9,10\}$, and let $f : A \to B$ be defined by

$$f(0) = a, \quad f(1) = c, \quad f(2) = a$$

and $g : B \to C$ be defined by

$$g(a) = 8, \quad g(b) = 7, \quad g(c) = 10, \quad g(d) = 9.$$

Then

$$\begin{aligned}(g \circ f)(0) &= g(f(0) = g(a) = 8,\\ (g \circ f)(1) &= g(f(1) = g(c) = 10,\\ (g \circ f)(2) &= g(f(2) = g(a) = 8.\end{aligned}$$

Figure 2.2 shows an arrow diagram for $g \circ f$.

Take care! For the composition of f and g to be defined, the domain of g must be the same as the codomain of f. Also the composite of f and g is denoted by $g \circ f$, not $f \circ g$ as you might expect.

Some important results about function composition are listed as theorem 42 below, but first we must define 'identity' functions:

For any set A the **identity function**, $\iota_A : A \to A$ is defined by $\iota_A(x) = x$, for all $x \in A$.

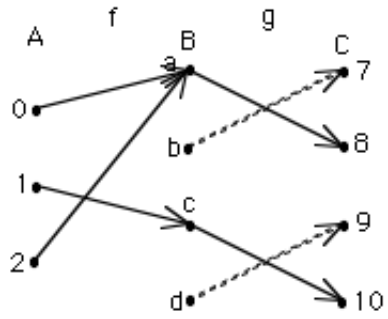


Figure 2.2: Composition of Functions.

42 Theorem. *Let $f : A \to B$, $g : B \to C$, and $h : C \to D$. Then*

1. $(h \circ g) \circ f = h \circ (g \circ f)$,
2. $\iota_B \circ f = f$, $f \circ \iota_A = f$,
3. *If f is a one-to-one correspondence (so that $f^{-1} : B \to A$ exists), then $f \circ f^{-1} = \iota_B$, and $f^{-1} \circ f = \iota_A$.*
4. *If f^{-1} and g^{-1} both exist, then $(g \circ f)^{-1}$ exists, $(g \circ f)^{-1} : C \to A$ and $(g \circ f)^{-1} = f^{-1} \circ g^{-1}$.*

Permutations

Let $X_n = \{1, 2, \ldots, n\}$ be the set of integers from 1 to n. We shall study some special functions $X_n \to X_n$. These functions, known as permutations, are the one-to-one functions from X_n to itself. Because X_n is finite, they are automatically also onto functions and hence are one-to-one correspondences. They are important in group theory and beyond. They are easy to calculate with, and give an example where the multiplication is not commutative.

43 Definition. *A **permutation** of X_n is a one-to-one function $\sigma : X_n \to X_n$. The set of all such permutations is denoted by S_n.*

There are several different ways of representing permutations.
Take, for example, $n = 6$ and let σ be the permutaion given by:

$$\sigma(1) = 2, \quad \sigma(2) = 5, \quad \sigma(3) = 4, \quad \sigma(4) = 3, \quad \sigma(5) = 6, \quad \sigma(6) = 1.$$

Here the permutation is given in **function form**. We may also give it in **arrow form**, as shown below.

$$\sigma: \quad 1 \to 2, \quad 2 \to 5, \quad 3 \to 4, \quad 4 \to 3, \quad 5 \to 6, \quad 6 \to 1.$$

A very helpful way of representing such a permutation is as a **two row matrix**.

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 5 & 4 & 3 & 6 & 1 \end{pmatrix}$$

We can also use a digraph to represent a permutation. We make a vertex for each element $x \in X_n$ and make a directed edge from x to $\sigma(x)$. The digraph for σ is shown in Figure 2.3. We can also use a 'shorthand' notation that follows directly from the

Figure 2.3: Digraph for a permutation σ.

diagram above. We start at the vertex 1 and follow the arrows around until we get back to the 1, but we don't repeat the 1. From the first part of the diagram we get the **cycle** (1256). Then we go to the next part of the diagram and start at either of the vertices. Again we follow the arrows around until we get back to where we started, but we don't repeat the initial vertex. We repeat this process as often as necessary. In this case the second cycle is (34), and we write $\sigma = (1256)(34)$. We say that σ has been decomposed into disjoint cycles (because the cycles have no vertex in common). For another example take

$$\tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 2 & 6 & 5 & 4 & 1 \end{pmatrix}$$

The digraph for τ is shown in Figure 2.4. In cycle notation then, $\tau = (136)(2)(45)$. However, we usually omit cycles of length 1, and so we write $\tau = (136)(45)$. Since 2 doesn't show in this representation we assume that 2 is fixed, that is, $\tau(2) = 2$. Cycle notation is a compact way of writing a permutation, and it gives nearly as much information as any other way of writing a permutation. However, be warned that the

Figure 2.4: Digraph for another permutation, τ.

cycle (123), for example, could represent any of the following permutations, or one like them.

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}, \quad \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 4 \end{pmatrix}, \quad \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 1 & 4 & 5 \end{pmatrix}, \quad \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 3 & 1 & 4 & 5 & 6 \end{pmatrix}.$$

We summarise this discussion with the following definitions.

44 Definition. *Let a_1, $a_2, \ldots, a_k$ be **distinct** elements of X_n, where $k \leq n$. The **cycle** $\rho = (a_1 a_2 \ldots a_k)$ is defined as follows:*

$\rho(a_1) = a_2$,

$\rho(a_2) = a_3$, *and in general,*

$\rho(a_j) = a_{j+1}$ *for* $j = 1, 2, \ldots, k-1$,

$\rho(a_k) = a_1$ *and*

$\rho(x) = x$ *for* $x \in X_n \setminus \{a_1, a_2, \ldots, a_k\}$.

*The set $\{a_1, a_2, \ldots, a_k\}$ is called the **orbit** of the cycle ρ. In this case, the cycle ρ is said to have **length** k.*

Note that the cycle $(a_1 a_2 \ldots a_k)$ is exactly the same as $(a_2 a_3 \ldots a_k a_1)$. As an example, (1347) = (3471).

45 Definition. *Cycles ρ_1, ρ_2, ... are **disjoint** if they have disjoint orbits.*

The cycles $\sigma = (1456)$ and $\tau = (2378)$ are disjoint cycles, since the orbit of σ is $\{1,4,5,6\}$ and the orbit of τ is $\{2,3,7,8\}$ and these sets are disjoint.

Multiplication of Permutations

If σ and τ are permutations on the same set X_n then $\sigma : X_n \to X_n$ and $\tau : X_n \to X_n$, so we can form the composites $\sigma \circ \tau$ and $\tau \circ \sigma$. When working with permutations these

composites are viewed as products, and written $\sigma\tau$ and $\tau\sigma$ respectively. The two-line matrix is possibly the most useful notation to use when we want to find the product of two permutations. So as an example on S_6 consider the two permutations σ and τ shown below.

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 2 & 4 & 5 & 1 & 6 \end{pmatrix}, \qquad \tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 1 & 4 & 5 & 3 & 2 \end{pmatrix}.$$

To find the product $\sigma\tau$, we can do it point by point. We get

$$\sigma\tau(1) = \sigma(\tau(1)) = \sigma(6) = 6,$$

$$\sigma\tau(2) = \sigma(\tau(2)) = \sigma(1) = 3,$$

$$\sigma\tau(3) = \sigma(\tau(3)) = \sigma(4) = 5,$$

$$\sigma\tau(4) = \sigma(\tau(4)) = \sigma(5) = 1,$$

$$\sigma\tau(5) = \sigma(\tau(5)) = \sigma(3) = 4,$$

$$\sigma\tau(6) = \sigma(\tau(6)) = \sigma(2) = 2.$$

The result is

$$\sigma\tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 3 & 5 & 1 & 4 & 2 \end{pmatrix}.$$

This method is okay, but perhaps a little clumsy. Let's try to make it simpler. We interchange columns in σ so that the first line is in the same order as the second line of τ. This gives an alternative way of writing σ:

$$\sigma = \begin{pmatrix} 6 & 1 & 4 & 5 & 3 & 2 \\ 6 & 3 & 5 & 1 & 4 & 2 \end{pmatrix}.$$

It is now easy to read the product:

$$\sigma\tau = \begin{pmatrix} 6 & 1 & 4 & 5 & 3 & 2 \\ 6 & 3 & 5 & 1 & 4 & 2 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 1 & 4 & 5 & 3 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 3 & 5 & 1 & 4 & 2 \end{pmatrix}.$$

The resulting product of $\sigma\tau$ takes the first line of τ and the second line of σ. The line that is identical in both permutations disappears. It is because we are using function notation that we read from right to left.

In cycle notation,

$$\sigma = (1345), \tau = (162)(345)$$

and we want to find

$$\sigma\tau = (1345)(162)(345).$$

We aim to write the composition of σ and τ as a product of disjoint cycles. We start with the right most occurrence of 1. We simply track what happens to each of the numbers:

$$1 \to_\tau 6 \to_\sigma 6, \quad 6 \to_\tau 2 \to_\sigma 2, \quad 2 \to_\tau 1 \to_\sigma 3,$$
$$3 \to_\tau 4 \to_\sigma 5, \quad 5 \to_\tau 3 \to_\sigma 4, \quad 4 \to_\tau 5 \to_\sigma 1.$$

The result is the cycle $\sigma\tau = (162354)$.

Further Examples of Calculating Products of Permutations

Let $\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 4 & 5 & 6 & 3 & 1 & 7 \end{pmatrix}$ and $\tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 4 & 1 & 3 & 7 & 2 \end{pmatrix}$.
To calculate $\sigma\tau$ we interchange the columns in σ so that the first line in σ is the same as the second line of τ. We get

$$\sigma = \begin{pmatrix} 5 & 6 & 4 & 1 & 3 & 7 & 2 \\ 3 & 1 & 6 & 2 & 5 & 7 & 4 \end{pmatrix}.$$

So we find

$$\sigma\tau = \begin{pmatrix} 5 & 6 & 4 & 1 & 3 & 7 & 2 \\ 3 & 1 & 6 & 2 & 5 & 7 & 4 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 4 & 1 & 3 & 7 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 1 & 6 & 2 & 5 & 7 & 4 \end{pmatrix}$$

Alternatively, we can write $\sigma = (1246)(35)$ and $\tau = (1534)(267)$ and find $\sigma\tau$.

$$1 \to_\tau 5 \to_\sigma 3, \quad 3 \to_\tau 4 \to_\sigma 6, \quad 6 \to_\tau 7 \to_\sigma 7,$$
$$7 \to_\tau 2 \to_\sigma 4, \quad 4 \to_\tau 1 \to_\sigma 2, \quad 2 \to_\tau 6 \to_\sigma 1.$$

This gives us the cycle (136742). As this doesn't include all the numbers, we need to check what happens to those missing. The only one is 5, and we find $5 \to_\tau 3 \to_\sigma 5$. This is what we would expect if the cycles stay disjoint, so it is a good test of what we have already done. So we have shown that $\sigma\tau = (136742)$, which is exactly the same as the result above.

Let's also calculate $\tau\sigma$, σ^2, σ^3, and σ^4.

To find $\tau\sigma$, we first interchange the columns of τ so that the first line of τ is the same as the second line of σ. In this form τ becomes

$$\tau = \begin{pmatrix} 2 & 4 & 5 & 6 & 3 & 1 & 7 \\ 6 & 1 & 3 & 7 & 4 & 5 & 2 \end{pmatrix}$$

and we conclude that

$$\tau\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 6 & 1 & 3 & 7 & 4 & 5 & 2 \end{pmatrix}$$

In cycle form this is (165472). Note that $\tau\sigma$ is different from $\sigma\tau$.

To find the powers of σ it can be quite helpful to use its digraph, shown in Figure 2.5. To find σ^2 we trace two steps around the diagram, rather than 1. Thus $1 \to 4$, $4 \to 1$ by σ^2. We find both 3 and 5 are fixed by σ^2. To find σ^3 we go round the digraph in steps of 3, and we follow a similar pattern for higher powers.

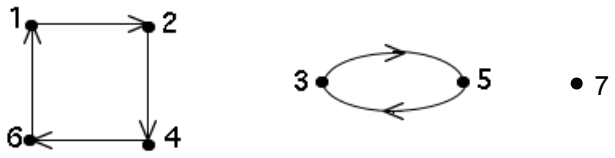


Figure 2.5: Digraph for σ.

We find

$$\sigma^2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 6 & 3 & 1 & 5 & 2 & 7 \end{pmatrix} = (14)(26),$$

$$\sigma^3 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 6 & 1 & 5 & 2 & 3 & 4 & 7 \end{pmatrix} = (1642)(35),$$

and

$$\sigma^4 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{pmatrix}.$$

It is rather difficult to write this last permutation in cycle notation as it is made up of cycles of length 1. In fact, we recognise it as the identity function because it maps every element to itself. We often represent the identity permutation by ε.

Inverses of Permutations

Since a permutation is a one-to-one correspondence, every permutation has an **inverse** and the inverse is also a permutation. Let us take σ from the above example, so

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 4 & 5 & 6 & 3 & 1 & 7 \end{pmatrix}.$$

To find σ^{-1} we simply swap the rows, and then rearrange the columns so that the first row is in the usual order.

$$\sigma^{-1} = \begin{pmatrix} 2 & 4 & 5 & 6 & 3 & 1 & 7 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 6 & 1 & 5 & 2 & 3 & 4 & 7 \end{pmatrix}.$$

It is easy to check that

$$\sigma\sigma^{-1} = \sigma^{-1}\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{pmatrix}.$$

We can also do this calculation in cycle form. We have $\sigma = (1246)(35)$. To find σ^{-1} we just reverse the cycles. So $\sigma^{-1} = (6421)(53) = (1642)(35)$. Again we have $\sigma\sigma^{-1} = (1246)(35)(1642)(35) = \varepsilon$.

2.2 Modular Arithmetic

We return to an important equivalence relation on the set of integers. This relation leads to a study of a number system containing only a finite number of elements. This system of numbers has many applications, for example, in computers, cryptography, error correcting codes and signal processing. It is also the basis for some important areas of mathematics.

46 Definition. *Let $a, b \in \mathbb{Z}$ and $m \in \mathbb{N}$. We say that a **is congruent to** b **modulo** m, if and only if $a - b$ is a multiple of m. We write $a \equiv b \pmod{m}$.*

Some easy examples are:
$17 \equiv 5 \pmod 4$, since 4 divides $17 - 5 = 12$.
$43 \equiv 98 \pmod{11}$ since 11 divides $43 - 98 = -55$.

ISBNs

Congruences often occur in applications involving error correcting codes. As you will know all books published since 1972 carry an ISBN (International Standard Book Number) code. *Discrete Mathematics with Applications* (Second Edition) by Susanna Epp carries the code 0–534–94446–9. This code consists of four parts: a group code, a publisher code, an identifying number assigned by the publisher and a check digit. In the ISBN 0–534–94446–9, the group code (0) denotes that the book was published in an English speaking country. The next group of digits (534) identifies the publisher, and the third group of digits (94446) designates this particular book among all the books published by the publisher 534. The final digit (9) is a check digit that is used to detect errors in copying or transmitting the ISBN.
The check digit has eleven possible values: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, X. (The check digit X represents 10.) The check digit is determined by multiplying the first nine digits of the ISBN by 10, 9, 8, 7, 6, 5, 4, 3 and 2 respectively to obtain a number y. The check digit d is chosen so that $y+d \equiv 0 \pmod{11}$. For the ISBN 0–534–94446–9,

$$\begin{aligned} y &= 10(0)+9(5)+8(3)+7(4)+6(9)+5(4)+4(4)+3(4)+2(6) \\ &= 0+45+24+28+54+20+16+12+12 \\ &= 211. \end{aligned}$$

So $y + d = 211 + 9 = 220 = 0 \pmod{11}$.

In 2007 a modified ISBN system was introduced. An extra three digits (currently either 978 or 979) are appended to the left and the check digit is calculated differently. To distinguish between the old and new systems, they are known as ISBN-10 and ISBN-13 respectively, reflecting the number of digits used. Books may now show numbers under both systems. For example, the current (fourth) edition of Epp (published 2010) shows:

ISBN–13: 978–0–495–39132–6
ISBN–10: 0–495–39132–8

Under ISBN–13 the check digit (the rightmost digit of the thirteen) is always a regular decimal digit 0, 1, 2,...,8 or 9 (X is no longer used) and is chosen so that $s \equiv 0 \pmod{10}$ where s is the sum of the odd-placed digits (the 1st, 3rd, 5th, ..., 13th digits) plus three times the sum of the even-placed digits. For example, for Epp 4ed

$$(9+8+4+5+9+3+6)+3(7+0+9+3+1+2) = 44+3\times 22 = 110 \equiv 0 \pmod{10}.$$

Returning now to congruences in general, the following result is crucial for further development:

47 Theorem. *Congruence modulo m is an equivalence relation on the set $\mathbb{Z}$ of integers.*

Since "congruence modulo m" is an equivalence relation, it partitions the integers into equivalence classes. These classes are often called the **congruence classes modulo** m, and the set of all equivalence classes modulo m is denoted by $\mathbb{Z}_m$. We recall that any two congruence classes are either equal or disjoint. In $\mathbb{Z}_m$, $[x] = [y]$ if and only if $x \equiv y \pmod{m}$. So if r is the remainder when we divide x by m, then $[x] = [r]$ in $\mathbb{Z}_m$. Since the possible values for the remainder r are $0, 1, 2, \ldots, m-1$, there are m distinct congruence classes in $\mathbb{Z}_m$. They are $[0], [1], [2], \ldots, [m-1]$. The remainders $0, 1, 2, \ldots, m-1$ crop up in modular arithmetic sufficiently often to warrant a special notation.

48 Definition. *When any integer $a \in \mathbb{Z}$ is divided by a positive integer $m \in \mathbb{N}$ to leave a non-negative **remainder** r less than m we write $r = a \bmod m$. So*

$$a = qm + r \quad \text{with} \quad 0 \leq r < m$$

*for a unique integer q called the **quotient** of a divided by m, written $q = a \operatorname{div} m$.*

Simple examples are:

$$\begin{array}{llll} 21 \bmod 4 = 1, & 21 \operatorname{div} 4 = 5 & \text{since} & 21 = 5 \times 4 + 1 \\ -21 \bmod 4 = 3, & -21 \operatorname{div} 4 = -6 & \text{since} & -21 = -6 \times 4 + 3 \end{array}$$

Example: Z_5

The possible remainders mod 5 are 0, 1, 2, 3 and 4. So the distinct congruence classes are in $\mathbb{Z}_5$ are:

$$\begin{aligned} [0] &= \{\ldots, -10, -5, 0, 5, 10, 15, 20 \ldots\}, \\ [1] &= \{\ldots, -9, -4, 1, 6, 11, 16, 21, \ldots\}, \\ [2] &= \{\ldots, -8, -3, 2, 7, 12, 17, 22, \ldots\}, \\ [3] &= \{\ldots, -7, -2, 3, 8, 13, 18, 23, \ldots\}, \\ [4] &= \{\ldots, -6, -1, 4, 9, 14, 19, 24, \ldots\}. \end{aligned}$$

We can represent each congruence class in $\mathbb{Z}_5$ in many different ways. For example [1] = [6] = [21] and [4] = [–1] = [19]. The number in the square brackets is called the representative of the congruence class.

49 Definition. Addition and Multiplication in $\mathbb{Z}_n$. *Let* $[x]$ *and* $[y]$ *be equivalence classes in* $\mathbb{Z}_n$*, for some integer n. We define* $[x]+[y]=[x+y]$ *and* $[x][y]=[xy]$.

This says that to add two congruence classes represented by x and y, find $x+y$. The answer is the congruence class containing $x+y$. Also, to multiply the classes represented by x and y, multiply x and y. The answer is the class containingxy. In $\mathbb{Z}_5$, $[3]+[4]=[3+4]=[7]=[2]$, and $[3][4]=[3\times 4]=[12]=[2]$. The following theorem ensures that these rules for addition and multiplication are well defined.

50 Theorem. *If* $x\equiv x'(mod\ m)$ *and* $y\equiv y'(mod\ m)$*, then*
(a) $x+y\equiv x'+y'(mod\ m)$ *and*
(b) $xy\equiv x'y'(mod\ m)$.

Example: Addition and Multiplication in $\mathbb{Z}_6$

Usin Definition 49 it is easy to make tables for these operations. For convenience, we leave out the square brackets around the numbers in the table.

$+$	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

$\times$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

These tables indicate some of the important properties of addition and multiplication in $\mathbb{Z}_m$. The properties are listed below.

Some Properties of Modular Arithmetic

1. Addition and multiplication are both **commutative**. This means that $[x]+[y]=[y]+[x]$, and $[x][y]=[y][x]$.

2. Addition and multiplication are both **associative**. This means that

$$([x]+[y])+[z]=[x]+([y]+[z]), \quad \text{and} \quad ([x][y])[z]=[x]([y][z]).$$

This property is not easily seen from tables like those shown above. However, it follows easily from associativity of addition and multiplication of the integers.

3. The class [0] is an **identity for addition**, and the class [1] is an **identity for multiplication**. This says that

$$[x]+[0]=[0]+[x]=[x], \quad \text{and} \quad [x][1]=[1][x]=[x],$$

for all $[x] \in \mathbb{Z}_m$.

4. Every element in $\mathbb{Z}_m$ has an **inverse** under the addition operation. That is, for every $[x] \in \mathbb{Z}_m$, there is an element $[y] \in \mathbb{Z}_m$ such that $[x]+[y]=[0]$. We often write $[y]=-[x]$.

5. Multiplication is **distributive** over addition in $\mathbb{Z}_m$. This means that

$$[x]([y]+[z])=[x][y]+[x][z].$$

Again this property is not evident from tables like those shown above. It follows easily from the distributivity property in the integers.

Warning! Some properties that are true for real numbers are not always true for operations in $\mathbb{Z}_n$.

Property 4 above says that there is always an inverse for addition. If we work in the real numbers, we also find that every non-zero real number has an inverse under multiplication. In general this is not true in $\mathbb{Z}_m$. It is sufficient to look at the multiplication table for $\mathbb{Z}_6$ to show that not every non-zero element of $\mathbb{Z}_6$ has an inverse under multiplication. For example, if we take $[x]=[4]$, there is no $[y]$ such that $[4][y]=1$. On the other hand $[5][5]=[1]$, so we can write $[5]^{-1}=[5]$.

Another unusual property is evident when we look at the multiplication table for $\mathbb{Z}_6$. We note that [4][3] = [0]. We have the product of two non-zero elements of $\mathbb{Z}_6$ giving a zero product. This is quite different from multiplication in the real numbers, where if $xy = 0$, we may assume that either $x=0$, or $y=0$.

It is good to compare addition and multiplication in $\mathbb{Z}_m$ with addition and multiplication in the real numbers and to see which other properties hold, and which ones don't. It is also good to see what difference different values of m might make.

Modular arithmetic can sometimes be used to draw conclusions about ordinary arithmetic. On a Sharp model EL–506S calculator, the value of 2^{30} is given as 1, 073, 741, 820. If this value were correct then the last digit of $2^{28} = 2^{30}/4$ would be 5. Clearly no power of 2 can be odd, so the last digit of 2^{30} must be wrong. What is the correct last digit of 2^{30}?

We note that two positive integers have the same last digit if and only if they are congruent modulo 10. In $\mathbb{Z}_{10}$,

$$[2^{30}] = [(2^5)^6] = [2^5]^6 = [32]^6 = [2]^6 = [2^6] = [64] = [4].$$

Hence $2^{30} \equiv 4 \pmod{10}$ and so the last digit of 2^{30} is 4, and $2^{30} = 1,073,741,824$.

Properties of Congruences mod m

We have already established some important properties of the congruence relation. But there are many others that lead to useful applications.

51 Theorem. *Let $m \geq 1$, and let a and b be any integers such that $a \equiv b \pmod{m}$.*

(i) If $d \mid m$, then $a \equiv b \pmod{d}$.

(ii) If e is a common factor of a, b and m, then $\frac{a}{e} \equiv \frac{b}{e} \pmod{\frac{m}{e}}$.

(iii) For all $n \geq 0$, $a^n \equiv b^n \pmod{m}$.

(iv) For all integers k, $ka \equiv kb \pmod{m}$.

We haven't been concentrating on proofs, but we shall give some proofs here to indicate what the proofs are like.

Proof. (i) Since $a \equiv b \bmod m$, there is an integer k such that $a - b = km$. But $m = dm'$ for some integer m'. So $a - b = kdm'$. That is, $a - b$ is a multiple of d, and so $a \equiv b \pmod{d}$. □

Proof. (ii) Put $\frac{a}{e} = a'$, $\frac{b}{e} = b'$ and $\frac{m}{e} = m'$. Since $a \equiv b \bmod m$, we an write $a - b = km$, for some integer k. This implies that $e(a' - b') = em'k$. Using cancellation in $\mathbb{Z}$, we conclude that $a' - b' = m'k$, or $a' \equiv b' \bmod m'$. This is what we wanted to show. □

The other proofs follow similar lines.

Examples Using Congruence Properties

1. Since $85 \equiv 37 \pmod 6$ and $3 \mid 6$, $\quad 85 \equiv 37 \pmod 3$.

2. Since $60 \equiv 42 \pmod{18}$, we may cancel the common divisor of 6 from all three numbers to get $10 \equiv 7 \pmod 3$.

3. We can use the properties, particularly (iii), to calculate $795^{25} \bmod 11$.
   First note that $795 \bmod 11 = 3$. So $(795)^{25} \equiv 3^{25} \pmod{11}$. We need a bit more effort to reduce 3^{25} modulo 11. The following lines follow progressively.

$$\begin{aligned} 3^2 &= 9 \equiv -2 \pmod{11} \\ 3^4 &= (3^2)^2 \equiv (-2)^2 = 4 \pmod{11} \\ 3^8 &= (3^4)^2 \equiv 4^2 = 16 \equiv 5 \pmod{11} \\ 3^{16} &= (3^8)^2 \equiv 5^2 = 25 \equiv 3 \pmod{11} \end{aligned}$$

   Since $25 = 16+8+1$, by the law of exponents, $3^{25} = 3^{16} \times 3^8 \times 3^1$.

$$\text{Hence } 3^{25} \equiv 3 \times 5 \times 3 = 15 \times 3 \equiv 4 \times 3 = 12 \equiv 1 \pmod{11}.$$

   We conclude that $795^{25} \bmod 11 = 1$.

Finally in this subsection a couple results about multiple moduli that will be useful later on.

52 Theorem. *Let m_1 and m_2 be positive integers, and suppose that $a \equiv b \pmod{m_1}$ and $a \equiv b \pmod{m_2}$. Let $m = \operatorname{lcm}(m_1, m_2)$. Then $a \equiv b \pmod m$.*

53 Corollary. *Let $m_1, m_2, \ldots, m_r$ be positive integers, and assume that $\gcd(m_i, m_j) = 1$ for all $i \neq j$. If $a \equiv b \pmod{m_i}$ for all $i = 1, 2, \ldots, r$, then $a \equiv b \pmod{m_1 m_2 \cdots m_r}$.*

Remark. *The corollary follows directly from the theorem because $\gcd(m_i, m_j) = 1$ for all $i \neq j$ implies that $\operatorname{lcm}(m_1, m_2, \ldots, m_r) = m_1 m_2 \cdots m_r$.*

Solving Linear Congruences

Next we use the results of the previous sections to solve, if possible, congruences of the form $ax \equiv b \pmod m$, where $a, b \in \mathbb{Z}$ and $m \in \mathbb{N}$ are given and $x \in \mathbb{Z}$ is to be found. These are called *linear* congruences. First, some exploratory examples:

$4x \equiv 9 \pmod{12}$. We look for x such that $4x-9$ is divisible by 12. However, it is not possible to find an integer x that will satisfy this relationship, since $4x-9$ will always be odd and, therefore, not divisible by 12.

$3x \equiv 23 \pmod{16}$. This has a solution $x=-3$, since $3(-3)-23=-32\equiv 0 \pmod{16}$. Further, $-3+16k$ is also a solution for any $k\in\mathbb{Z}$ since if -3 is a solution, so is every member of the congruence class $[-3]_{16}$. The smallest positive integer satisfying the congruence is $x=13$. Possibly the question that is more important is: where did $x=-3$ come from? The answer is simply by trial and error, noting that it is easier to try smaller numbers first (both positive and negative).

$21x \equiv 18 \pmod{12}$. This has a solution$x=2$, since $21\equiv 9 \pmod{12}$ and so an equivalent congruence is $9x\equiv 18 \pmod{12}$, and this certainly has $x=2$ as a solution. Another solution is $x=-2$, since $-18\equiv 18 \pmod{12}$. Thus $x=2$ and $x=10$ are both (positive) solutions to this linear congruence.

$17x \equiv 45 \pmod{11}$. To simplify (mental) arithmetic we'll first replace 17 and 45 by their remainders modulo 11 to get the equivalent congruence $6x\equiv 1 \pmod{11}$. Now we can use trial and error trying $x=1$, $x=-1$ without success but then $x=2$ which works. Alternatively, we can see that $6x\equiv 1 \pmod{11}$ is equivalent to the congruence $6x=12 \pmod{11}$ (since $12\equiv 1 \pmod{11}$), and this last congruence has a solution $x=2$. Other solutions are $x=2+11k$ for $k\in\mathbb{Z}$.

Having found *some* solutions to a linear congruence, we need to ask if we have found *all* solutions. Theorem 54 deals with this:

54 Theorem. *Let $m\in\mathbb{N}$ and $a,b\in\mathbb{Z}$ and let $d=gcd\,(a,m)$. The congruence*

$$ax\equiv b \pmod{m}$$

has a solution if and only if d divides b.
If d does divide b, put $a=da'$, $b=db'$, $m=dm'$. Then the modified congruence

$$a'x\equiv b' \pmod{m'}$$

has a unique solution r modulo m'. The congruence class $[r]_{m'}$ is the complete solution set to the original linear congruence, corresponding to d distinct solutions modulo m.

We will now review the some of the exploratory examples in light of the Theorem 54 and also work a couple more examples:

$4x \equiv 9 \pmod{12}$. Here $a=4$, $b=9$, $m=12$ and $d=$ gcd (4, 12) = 4. Since 4 does not divide 9 the theorem confirms that there is no solution to the congruence.

3x $\equiv$ 23 (mod 16). Here $d = \gcd(3, 16) = 1$. So d divides $b = 23$, and the congruence has a solution. Further, since $d = 1$, $m' = m = 16$ and so the solution set is a single congruence class modulo 16. So the solution set $[-3]_{16} = [13]_{16}$ we found is complete.

21x $\equiv$ 18 (mod 12). In this case $d = \gcd(21, 12) = 3$. Hence d divides $b = 18$. The modified congruence is

$$7x \equiv 6 \pmod 4.$$

This can be rewritten as $(-1)x \equiv 2 \pmod 4$ leading immediately to $x \equiv -2 \equiv 2 \pmod 4$. The solution is unique modulo 4; any x in $[2]_4$, and only such integers, solves the modified equation, and, more importantly, the original equation. We could leave it at that, but it is sometimes desirable to express the solution in terms of the original modulus, in this case 12. Because $d = 3$ there will be three different solutions modulo 12, namely 2, $2+4 = 6$ and $2+2\times 4 = 10$. We have already found the solutions $x = 2$ and $x = 10$, but did not observe the solution$x = 6$. A quick check does confirm that $x = 6$ is also a solution. Note that going beyond $2+8$ to $2+12 = 14$, $2+16 = 18$ etc. produces nothing new, since $14 \equiv 2 \pmod{12}$, $18 \equiv 6 \pmod{12}$ and so on.

15x $\equiv$ 9 (mod 36). Here $d = \gcd(15, 36) = 3$, and 3 divides $b = 9$, so there will be three distinct solutions modulo 36. To find them we must solve the modified congruence

$$5x \equiv 3 \pmod{12}.$$

Since $3 \equiv 15 \pmod{12}$, we can rewrite this as $5x \equiv 15 \pmod{12}$ leading to $x \equiv 3 \pmod{12}$. Hence our solutions are $x = 3$, 3+12=15 and $3+2\times 12 = 27$, all mod 36.

495x $\equiv$ 120 (mod 615). Since $495 = 15 \cdot 33$ and $615 = 15 \cdot 41$ we get $d = 15$. As $15 \mid 120$ there are 15 distinct solutions mod 615. The modified equation is

$$33x \equiv 8 \pmod{41}.$$

To find a solution to this congruence we note that $8 \equiv -33 \pmod{41}$, so we can rewrite the congruence as $33x \equiv -33 \pmod{41}$, giving the solution $x \equiv -1 \equiv 40 \pmod{41}$. We can now find all the solutions to the original problem, $495x \equiv 120 \pmod{615}$. They are $x = 40$, $x = 40+41 = 81$, $x = 40+2\cdot 41 = 122$, $x = 40+3\cdot 41 = 163, \ldots$ and $x = 40+14\cdot 41 = 614$ (all mod 615).

Modular Inverses

One of the questions that often arises in applications of modular arithmetic is to find a *multiplicative inverse* of one integer modulo another. Any solution s to the linear congruence $ax \equiv 1 \pmod{m}$ is called an **inverse of a** $\pmod{\mathbf{m}}$. We write $s \equiv a^{-1} \pmod{m}$ and, if $0 \leq s < m$, $s = x^{-1} \bmod m$.

For example, if we want to solve the linear congruence

$$3x \equiv 7 \pmod{10},$$

a natural approach is to find $3^{-1} \bmod 10$ and multiply both sides of the congruence by 3^{-1}. In this case it is fairly easy to determine that $3^{-1} \bmod 10 = 7$ and so we get

$$7 \cdot 3x \equiv 7 \cdot 7 \pmod{10} \quad \text{or} \quad x \equiv 49 \equiv 9 \pmod{10}.$$

However, when the numbers are larger it is not always so easy to spot an inverse — assuming one exists. Later in this chapter we will investigate an algorithm for finding inverses, but for now we just discuss *when* they exist. Recall from Theorem 54 that the linear congruence $ax \equiv b \pmod{m}$ has a solution if and only if $gcd(a,m) \,|\, b$. When $b = 1$ this means $gcd(a,m) = 1$. When any pair of integers x, y have greatest common divisor 1 we say they are *relatively prime*. Thus

$\mathbf{a^{-1}}$ mod m exists if and only if a and m are relatively prime.

For example, if we looked for the inverse of 5 modulo 10, we can find no integer x so that $5x \equiv 1$ modulo 10. To see that this is not possible, recall that solving the problem is equivalent to finding integers x and y such that $5x = 10y + 1$. We cannot do this because 5 and 10 have a common divisor bigger than 1. They are not relatively prime. Here 5 divides $5x$, and 5 divides $10y$ and for the above equality to hold we would require that 5 divide 1, which is clearly not the case.

On the other hand, for example, $10^{-1} \bmod 21$ does exist because 10 and 21 are relatively prime. In fact $10 \times -2 = -20 \equiv 1 \pmod{21}$ and so $-2 \equiv 10^{-1} \pmod{21}$. Since $-2 \equiv 19 \pmod{21}$ we could also write $10^{-1} \bmod 21 = 19$.

The Chinese Remainder Theorem

In an old party game, Alice calls for Bob to think of a number n, between 1 and 60. Alice then asks Bob to tell the remainder of n on division by 3, 4, and 5 respectively. For example, these remainders might be 2, 1, and 2. Then Alice tells Bob what the number is. How? One possibility is that Alice uses the Chinese Remainder Theorem.

The number n is a solution to the *simultaneous* congruences:

$$\begin{aligned} n &\equiv 2 \pmod 3, \\ n &\equiv 1 \pmod 4, \\ n &\equiv 2 \pmod 5. \end{aligned}$$

We can take a fairly laborious approach. The most general solution to the first congruence is $n = 2+3u$, for any $u \in \mathbb{Z}$. We substitute u into the second congruence and solve for u:

$$\begin{aligned} 2+3u &\equiv 1 \pmod 4 \\ -u &\equiv -1 \pmod 4 \\ u &\equiv 1 \pmod 4 \end{aligned}$$

Thus $u = 1+4v$, for any $v \in \mathbb{Z}$, making $n = 2+3u = 2+3(1+4v) = 5+12v$. We now put this value for n into the last congruence and solve for v.

$$\begin{aligned} 5+12v &\equiv 2 \pmod 5 \\ 12v &\equiv 2 \pmod 5 \\ 2v &\equiv 2 \pmod 5 \\ v &\equiv 1 \pmod 5 \end{aligned}$$

Hence $v = 1+5w$ for any $w \in \mathbb{Z}$. Thus $n = 5+12v = 5+12(1+5w) = 17+60w$. The only solution between 1 and 60 is when $w = 0$ and then $n = 17$. Notice that any two solutions are congruent mod 60 and that 60 = 3.4.5, the product of the moduli.
Another way of solving this problem is to say that, from the last congruence, $n \equiv 2 \pmod 5$, n must be one of the 11 numbers

$$2,\ 7,\ 12,\ \underline{17},\ 22,\ 27,\ 32,\ \underline{37},\ 42,\ 47,\ 52,\ \underline{57}.$$

The next congruence, $n \equiv 1 \pmod 4$, indicates that n is among every fourth number of this list, starting with 17. These numbers have been underlined. These are the only ones that are congruent to 1 (mod 4). Finally we find (from the underlined numbers) any that are congruent to 2 mod 3. The only one satisfying this requirement is 17.

The general question here is that of solving simultaneous congruences:

$$\begin{aligned} x &\equiv b_1 \pmod{m_1} \\ x &\equiv b_2 \pmod{m_2} \\ &\vdots \\ x &\equiv b_n \pmod{m_n} \end{aligned}$$

We assume that the moduli are relatively prime in pairs. That is, for all $i \neq j$, $\gcd(m_i, m_j) = 1$. The b's can be any integers.

55 Theorem. The Chinese Remainder Theorem *The simultaneous congruences shown above, with moduli relatively prime in pairs, always have a solution. The solution is unique modulo* $M = m_1 m_2 \cdots m_n$.

This is an interesting theorem but it tells us little about how to find the solution. If we worked through a proof, we might have some idea about finding a solution. Also, we could generalise the method we used in solving the initial part of the game problem. Better still, we could also use the following algorithm.

The CRT Algorithm

(CRT stands for "Chinese Remainder Theorem".)

Input: A positive integer n, integers $b_1, b_2, \ldots, b_n$ and positive integers $m_1, m_2, \ldots, m_n$ which are relatively prime in pairs.
Output: An integer x satisfying the simultaneous congruences given above.

Step 0. Set $M = m_1 m_2 \cdots m_n$.

Step 1. For all $j = 1, 2, \ldots, n$ find $c_j \equiv (\frac{M}{m_j})^{-1} \pmod{m_j}$.

Step 2. For all $j = 1, 2, \ldots, n$ set $\alpha_j = c_j(\frac{M}{m_j})$.

Step 3. Set $x = \sum_j \alpha_j b_j$.

56 Remarks.

1. The least non-negative solution will be $x \bmod M$.

2. The size of x *can be reduced by choosing some of the* c_j*'s to be negative.*

3. The algorithm works because $\alpha_j \bmod m_k = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{if } j \neq k \end{cases}$.

Let us redo the party problem, using this algorithm.
We have $x \equiv 2 \pmod 3$, $x \equiv 1 \pmod 4$, and $x \equiv 2 \pmod 5$. So $M = 3 \cdot 4 \cdot 5 = 60$, with $m_1 = 3$, $m_2 = 4$ and $m_3 = 5$. Also $b_1 = 2$, $b_2 = 1$ and $b_3 = 2$. Therefore

$$\begin{aligned} c_1 \tfrac{60}{3} = 20c_1 &\equiv 1 \pmod 3. \\ \text{So } 2c_1 &\equiv 1 \pmod 3 \\ \text{and hence we can take } c_1 &\equiv 2 \text{ or } -1 \pmod 3. \\ \text{Choose } \alpha_1 &= (-1) \cdot 20 = -20. \end{aligned}$$

$$\begin{aligned} c_2 \tfrac{60}{4} = 15c_2 &\equiv 1 \pmod 4. \\ \text{So } 3c_2 &\equiv 1 \pmod 4, \\ \text{and hence we can take } c_2 &\equiv 3 \text{ or } -1 \pmod 4. \\ \text{Choose } \alpha_2 &= (-1) \cdot 15 = -15. \end{aligned}$$

$$\begin{aligned} c_3 \tfrac{60}{5} = 12c_3 &\equiv 1 \pmod 5. \\ \text{So } 2c_3 &\equiv 1 \pmod 5, \\ \text{and hence we can take } c_3 &\equiv 3 \text{ or } -2 \pmod 5. \\ \text{Choose } \alpha_3 &= (3) \cdot 12 = 36. \end{aligned}$$

Now the algorithm tells us that

$$\begin{aligned} x &= \alpha_1 b_1 + \alpha_2 b_2 + \alpha_3 b_3 \\ &= (-20)2 + (-15)1 + (36)2 \\ &= -40 - 15 + 72 = 17. \end{aligned}$$

Choosing $c_1 = 2$ and $c_2 = 3$ gives $\alpha_1 = 40$ and $\alpha_2 = 45$, leading instead to

$$\begin{aligned} x &= (40)2 + (45)1 + (36)2 \\ &= 80 + 45 + 72 = 197, \end{aligned}$$

but note that $197 \bmod 60 = 17$, as before. This is more typical; the least positive answer to the problem is usually $x \bmod M$ rather than plain x.

A convenient compact way to set out the calculations required for the CRT algorithm is to use a table like Table 2.6 below, dealing with the same 'party problem' as above and using all positive inverses.

Table 2.6: Using the CRT algorithm to solve $x \bmod 3 = 2$, $x \bmod 4 = 1$, $x \bmod 5 = 2$.

m_i	$\frac{M}{m_i}$	$r_i = \frac{M}{m_i} \bmod m_i$	$c_i = r_i^{-1} \bmod m_i$	$\alpha_i = c_i\left(\frac{M}{m_i}\right)$	b_i	$\alpha_i b_i$
3	20	2	2	40	2	80
4	15	3	3	45	1	45
5	12	2	3	36	2	72
$M = 60$						$197 \bmod 60 = 17$

Notice that the α_j's in the CRT algorithm don't depend on the b_j's. So, if the m_j's remain fixed but the b_j's change, only the last step needs re-doing. This is handy for the party game example above. For example, if the remainders had been 2,3,4 instead of 2,1,2 then the 'mystery' number was

$$\sum_j \alpha_j b_j \bmod 60 = (40)2 + 45(3) + (36)4 \bmod 60 = 359 \bmod 60 = 59.$$

Modular coordinates

One consequence of the Chinese Remainder Theorem is that a sequence of non-negative integers $b_1, b_2, \ldots, b_n$ may be replaced by *one* integer x. The number x may be regarded as the **code number** for the sequence. Equivalently, we may think of the b_j as the *coordinates* of the number x. Here the b_j are given modulo m_j, and x is determined modulo their product.

57 Definition. *Suppose that the positive integers $m_1, m_2, \ldots, m_n$ are relatively prime in pairs. Then x has coordinates $(b_1, b_2, \ldots, b_n)$ with respect to the integers $m_1, m_2, \ldots, m_n$, provided $x \equiv b_j \pmod{m_j}$, with $b_j \in \mathbb{Z}_{m_j}$, for $j = 1, 2, \ldots, n$.*

Note that if $M = m_1 m_2 \cdots m_n$, then by the Chinese Remainder Theorem the value of the integer x is uniquely determined mod M from its coordinates.

58 Theorem. *Let x and y have coordinates $(a_1, a_2, \ldots, a_n)$ and $(b_1, b_2, \ldots, b_n)$, respectively. Then the coordinates of $x + y$ are $(a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n)$, and the coordinates of xy are $(a_1 b_1, a_2 b_2, \ldots, a_n b_n)$. The latter computations are made modulo m_j for $j = 1, 2, \ldots, n$.*

Examples

1. Choosing the moduli 9, 10, 11, we find the coordinates of the numbers 735 and 543, and the coordinates of their product.

   We can actually find a number modulo 9 by taking the sum of its digits. So $735 \equiv 7+3+5 = 15 \equiv 6 \pmod 9$ and $543 \equiv 5+4+3 = 12 \equiv 3 \pmod 9$.

   By using the last digit we can find the number modulo 10. So $735 \equiv 5 \pmod{10}$ and $543 \equiv 3 \pmod{10}$.

   We can find a number modulo 11 by taking the alternating sum of its digits. So $735 \equiv 5-3+7 = 9 \pmod{11}$ and $543 \equiv 3-4+5 = 4 \pmod{11}$.

   Hence, the coordinates of 735 are (6, 5, 9) and the coordinates of 543 are (3, 3, 4). The coordinates of 735×543 are found by taking the product, coordinate by coordinate. Hence they are

   $$\begin{aligned}(6\times 3 \bmod 9, 5\times 3 \bmod 10, 9\times 4 \bmod 11)& \\ = (18 \bmod 9, 15 \bmod 10, 36 \bmod 11) = (0,5,3)&\end{aligned}$$

2. *Parallel Computation* The techniques in the previous example with moduli 9, 10 and 11 can be used to do computations with very large numbers. The idea is that we do the computation on each coordinate, which is reasonable in size, and at the end put the answers together. Unlike working in a base b no carrying is involved, so each computation can be done independently. The technique is useful for parallel computation, where many smaller computations can be done at the same time.

   In summary: Suppose that we wish to do many computations on very large integers, and these computations involve addition and multiplication only. Choose integers $m_1, m_2, \ldots, m_n$, that are relatively prime in pairs, and set $M = m_1 m_2 \cdots m_n$. The computation proceeds as follows.

   (a) Replace each integer r in the computation with its coordinates $(r_1, r_2, \ldots, r_n)$. This amounts to finding $r \bmod m_j$ for $j = 1, 2, \ldots, n$.

   (b) Independently do the computations on the jth coordinate, working modulo m_j. In this way, the answer A_j can be found mod m_j for $j = 1, \ldots, n$. Thus, the final answer A has coordinates $(A_1, A_2, \ldots, A_n)$.

(c) Finally, reconstitute the answer A by solving the congruences

$$A = A_j \pmod{m_j} \text{ for } j = 1, 2, \ldots, n.$$

This can be done by the CRT algorithm. Note that this requires calculating the values of $\alpha_1, \ldots, \alpha_n$, but this could be done in advance, or in parallel with step (b).

Step (b) breaks the computation into simple, more manageable computations that can be done independently on different machines, or by different people, and at different times. Step (a) takes some time, but is relatively straightforward.

Even though the answer is found mod M, it is possible to arrange for M to be much larger than the expected answer, so that A mod M will be the same as A.

3. This example uses small numbers to illustrate the technique described above.

Using moduli 7, 9, 10 and 11 we compute $A = 35 \times 73$. Since $35 \equiv 0 \pmod 7$, $35 \equiv 8 \pmod 9$ $35 \equiv 5 \pmod{10}$, and $35 \equiv 2 \pmod{11}$, the coordinates of 35 are (0, 8, 5, 2). Similarly, the coordinates of 73 are (3, 1, 3, 7). We calculate the coordinates of A, as follows:

$$\begin{aligned} A_1 &= 0 \times 3 \bmod 7 &&= 0, \\ A_2 &= 8 \times 1 \bmod 9 &&= 8, \\ A_3 &= 5 \times 3 \bmod 10 &&= 5, \\ A_4 &= 2 \times 7 \bmod 11 &&= 3. \end{aligned}$$

Thus A has coordinates (0, 8, 5, 3) with respect to these moduli.We use the CRT algorithm to solve these congruences, obtaining $A \equiv 2555 \pmod{6930}$. Since this answer is calculated modulo 6930, we need some check to decide whether 2555 is a sensible result for finding the product of 35 and 73. A rough estimate shows that$A$ is no larger than $40 \times 80 = 3200$. So we conclude that $A = 2555$, since a larger A modulo 6930 would be well over this estimate.

As previously noted, (Remarks 56) each of the numbers $\alpha_1, \alpha_2, \ldots, \alpha_n$ in the CRT algorithm is defined so that $\alpha_j \bmod m_j = 1$ and $\alpha_j \bmod m_k = 0$ when $k \neq j$. This means, for example for $n = 3$, that the α's have coordinates

$$\alpha_1 = (1,0,0),\ \alpha_2 = (0,1,0),\ \alpha_3 = (0,0,1).$$

For a number x with coordinates (b_1, b_2, b_3) we know from the CRT algorithm that $x \equiv \alpha_1 b_1 + \alpha_2 b_2 + \alpha_2 b_3 \pmod{m_1 m_2 m_3}$ so, a little informally, we can write

$$(b_1, b_2, b_3) = b_1(1,0,0) + b_2(0,1,0) + b_3(0,0,1).$$

By analogy with vectors and linear spaces, we therefore call $(\alpha_1, \alpha_2, \alpha_3)$ a *basis for solutions modulo* (m_1, m_2, m_3):

59 Definition. *For moduli (m_1, m_2, m_3), a **basis** consists of three integers α_1, α_2, and α_3 with components $(1,0,0), (0,1,0)$ and $(0,0,1)$ respectively. The definition generalises to any number of moduli.*

We can now summarise all of our work on the Chinese Remainder Theorem as follows:

60 Theorem. *Let $(\alpha_1, \ldots, \alpha_n)$ be a basis for the coprime moduli $(m_1, \ldots, m_n)$. Then the system of simultaneous congruences $x \equiv b_j \pmod{m_j}$, $j = 1, \ldots, n$ has the solution*

$$x \equiv \alpha_1 b_1 + \alpha_2 b_2 + \cdots + \alpha_n b_n \pmod{M}$$

where $M = m_1 m_2 \ldots m_n$.

Two More Examples

1. We find a basis for the moduli $(9, 10)$. For this, α_1 must satisfy $\alpha_1 \equiv 1 \pmod 9$ and $\alpha_1 \equiv 0 \pmod{10}$. From the CRT algorithm this is achieved by taking

   $$\alpha_1 = (10^{-1} \bmod 9) \cdot 10 = (1^{-1} \bmod 9) \cdot 10 = 1 \cdot 10 = 10.$$

   Similarly, α_2 must satisfy $\alpha_2 \equiv 0 \pmod 9$, and $\alpha_2 \equiv 1 \pmod{10}$. From the CRT algorithm this is achieved by taking

   $$\alpha_2 = (9^{-1} \bmod 10) \cdot 9 = (-1^{-1} \bmod 10) \cdot 9 \equiv -1 \cdot 9 = -9.$$

   Hence a basis for this system is (10, -9).

   Now suppose we want to solve

   $$x \equiv 7 \pmod 9 \qquad x \equiv 6 \pmod{10}.$$

   From Theorem 60 we put

   $$x = \alpha_1 \cdot 7 + \alpha_2 \cdot 6 = 10 \cdot 7 + (-9) \cdot 6 = 70 - 54 = 16$$

   and since $M = 9 \cdot 10$ the solution set is $\{x : x \equiv 16 \pmod{90}\}$.

2. We find a basis for the moduli 2, 3, 5, and 7.

$$\begin{aligned}
\alpha_1 &= ((3\cdot 5\cdot 7)^{-1} \bmod 2)(3\cdot 5\cdot 7) = (1^{-1} \bmod 2)105 = (1)105 = 105\\
\alpha_2 &= ((2\cdot 5\cdot 7)^{-1} \bmod 3)(2\cdot 5\cdot 7) = (1^{-1} \bmod 3)70 = (1)70 = 70\\
\alpha_3 &= ((2\cdot 3\cdot 7)^{-1} \bmod 5)(2\cdot 3\cdot 7) = (2^{-1} \bmod 5)42 = (3)42 = 126\\
\alpha_4 &= ((2\cdot 3\cdot 5)^{-1} \bmod 7)(2\cdot 3\cdot 5) = (2^{-1} \bmod 7)30 = (4)30 = 120.
\end{aligned}$$

Thus a basis for the moduli 2, 3, 5 and 7 is (105, 70, 126, 120).

Consider a number x with components (1, 2, 1, 0). Since $2\cdot 3\cdot 5\cdot 7 = 210$, to retrieve the number we do the following calculation:

$$\begin{aligned}
x &\equiv 1\cdot\alpha_1 + 2\alpha_2 + 1\cdot\alpha_3 + 0\cdot\alpha_4\\
&= 1\cdot 105 + 2\cdot 70 + 1\cdot 126 + 0\cdot 120\\
&= 371\\
&\equiv 161 \pmod{210}.
\end{aligned}$$

Similarly if a number y has the components (0, 1, 3, 2),

$$\begin{aligned}
y &\equiv 0\cdot\alpha_1 + 1\cdot\alpha_2 + 3\alpha_3 + 2\alpha_4\\
&= 0\cdot 105 + 1\cdot 70 + 3\cdot 126 + 2\cdot 120\\
&= 688\\
&\equiv 58 \pmod{210}.
\end{aligned}$$

Euclid's Algorithm for GCD

(c.f. Epp 4ed Section 4.8)

As we know the greatest common divisor (gcd, also know as 'highest common factor', hcf) of two integers a and b is the largest positive integer that divides both a and b, *i.e.* greatest integer that is common to the sets of divisors of a and b. For example if $a = 20$ and $b = 16$, then the largest positive integer that divides them both is 4, and we write gcd(20,16) = 4. In making this definition we usually assume that a and b aren't both zero. It is not necessary that a and b be positive integers, however, in the context in which we will be using them they will usually be positive. So the discussion will be restricted to positive integers.

There are several ways of finding the gcd of two integers. One way is just to find both sets of divisors.

Examples

1. We find the greatest common divisor of 24 and 30.

   The (positive) divisors of 24 are 1, 2, 3, 4, 6, 8, 12 and 24 and the divisors of 30 are 1, 2, 3, 5, 6, 10, 15, and 30, so the common divisors of 24 and 30 are 1, 2, 3, and 6. Hence gcd(24, 30) = 6.

2. We find the greatest common divisors of 15 and 8.

   The divisors of 15 are 1, 3, 5, and 15 and the divisors of 8 are 1, 2, 4, and 8. The only common divisor of 15 and 8 is 1, so we have gcd(15, 8) = 1.

3. We find the greatest common divisor of 0 and 10.
   The divisors of 0 are 1, 2, 3, 4, ..., and the divisors of 10 are 1, 2, 5 and 10, so the common divisors of 0 and 10 are 1, 2, 5 and 10. Hence gcd(0, 10) = 10.

This illustrates for us that two integers a and b (not both zero) will always have a common divisor, since 1 will always be a divisor of both numbers. Hence common divisors always exist and since the set of common divisors is always finite (why?) there will always be a largest element in the set of common divisors. You will notice that the above method for finding common divisors is not very efficient.

In the Example 2 above the only common divisor is 1. Recall that in this case we say that the integers are *relatively prime* or *coprime*.

A more efficient method for computing the greatest common divisor of two integers is the **Euclidean Algorithm**. Before describing this algorithm we will use it to find the gcd(54, 153). As a first step we divide the larger of the two integers by the smaller and find the remainder:

$$153 = 54 \cdot 2 + 45.$$

Any divisor of 153 and 54 must also be a divisor o $45 = 135 - 54 \cdot 2$. Hence the greatest common divisor of 153 and 54 is also the greatest common divisor of 54 and 45. This reduces the problem of finding gcd(153,54) to finding gcd(54, 45). We proceed as before and divide the larger number by the smaller:

$$54 = 45 \cdot 1 + 9.$$

As above, any common divisor of 54 and 45 is also a common divisor of 9 and the gcd(54, 45) is the same as the gcd(45, 9). But

$$45 = 9 \cdot 5.$$

Since 9 divides 45, it follows that gcd(45, 9) = 9 and gcd(45, 9) = gcd(54, 45) = gcd(54, 153) = 9.
We formalise these arguments as theorems that we can prove.

61 Theorem. *Let a and b be any integers with $b \neq 0$, and suppose that q and r are non-negative integers that satisfy*

$$a = b \cdot q + r.$$

Then gcd (a,b) =gcd (b,r).

Proof. If we can show that the set of common divisors of a and b is the same as the set of common divisors of b and r, we will have shown that gcd $(a,b) =$ gcd (b,r), since both pairs must then have the same greatest common divisor.
So suppose that d divides both a and b (or d is a common divisor of a and b). Then because $r = a - b \cdot q$ it follows that d divides r and is therefore a common divisor of b and r. From this we can conclude that any common divisor of a and b is also a common divisor of b and r.
We also need to show that any common divisor of b and r is also a common divisor of a and b. So suppose that d divides both b and r. Then, since $a = b \cdot q + r$ it follows that d divides a. This is enough to show that any common divisor of b and r is also a common divisor of a and b.
Since the set of common divisors of a and b is the same as the set of common divisors of b and r it follows that gcd $(a,b) =$ gcd (b,r). □

The **Euclidean algorithm** can now be described as follows:

1. Let a and b be integers with $a > b \geq 0$.

2. First check whether $b = 0$. If it is then gcd $(a,b) = a$ and the process is complete. If $b \neq 0$, then $b > 0$ and we can find q and r such that

$$a = b \cdot q + r \qquad \text{where } 0 \leq r < b.$$

   Note that in this calculation $r = a \bmod b$ and from the theorem above, gcd $(a,b) =$ gcd (b,r). Hence we have changed the problem of finding gcd (a,b) to the problem of finding gcd (b,r). This is helpful since these numbers are smaller than the original numbers. We started with the assumption that $a > b \geq 0$ and the

remainder r that we have found satisfies $0 \leq r < b$. Combining these inequalities gives

$$a > b > r \geq 0.$$

So the largest number in the pair (b, r) is smaller than the largest number in the pair (a, b).

3. We now repeat the process going back to step 2, using b instead of a and r instead of b. The repetitions will eventually terminate when $r = 0$ since each new remainder is less than the previous one. The gcd is last non-zero remainder.

On paper, work using Euclid's algorithm can be set out very simply:

Table 2.7: Using Euclid's Algorithm to find gcd(721,136)

Recipe:	Example
1. Write down the larger number. 2. Underneath, write down the smaller number. 3. Under that, write down the remainder when the number two places above is divided by the number one place above. 4. Repeat step 3 until you write down 0. 5. Your gcd is the number above 0.	721 136 41 13 2 → 1 ← 0

For programming, the Euclidean algorithm can be expressed in pseudocode as below, where ":=" means "is assigned the value of".

procedure gcd (a, b: integers, $a > b \geq 0$)
  $x := a$
  $y := b$
  while $y \neq 0$
  begin
    $r := x \bmod y$
    $x := y$
    $y := r$
  end {gcd (a, b) is x}.

Table 2.8 below traces the values of the variables x, y and r as the above procedure is applied to again finding gcd(721,136). Values given are for at the end of each pass through the 'while' loop, where loop 0 means everything before the first loop starts (often called the initialisation phase).

Table 2.8: Using procedure gcd to find gcd(721,136)

Step	**r**	**x**	**y**
0.	—	721	136
1.	$721 \bmod 136 = 41$	136	$41 \neq 0$
2.	$136 \bmod 41 = 13$	41	$13 \neq 0$
3.	$41 \bmod 13 = 2$	13	$2 \neq 0$
4.	$13 \bmod 2 = 1$	2	$1 \neq 0$
5.	$2 \bmod 1 = 0$	1	0

The process terminates after step 5 since at this point $y = 0$. We then conclude that gcd (721, 136) is the current value of x, which is 1.

Extending Euclid's Algorithm: Solving gcd$(a,b) = ua+vb$

Having found d = gcd (a,b) we now want to find, for a reasons to be explained shortly, integers u and v such that $d = ua + vb$. It is always possible to do this, as the following theorem claims:

62 Theorem. *Let a and b be integers, $a > b \geq 0$. Then there are integers u and v such that*

$$d = gcd\ (a,b) = ua + vb.$$

This result can be proved by investigating the set S of *all* linear combinations of $ua+vb$ of a and b where u and v are integers.

$$S = \{au + bv\ :\ u,v \in \mathbb{Z}\}.$$

Since d divides both a and b (from its definition) it follows that d divides every member of the set S. So to prove the result we show that d itself is in the set S. We choose s the smallest positive integer in S, and by investigating the properties of s we show that in fact $s = d$ (we omit the details of this last part).

This theorem gives us no indication as to how we might find the integers u and v. However we can modify the Euclidean algorithm to do this, at the same time obtaining a constructive proof of Theorem 62. Actually there are two rather different ways of modifying Euclid's algorithm to achieve our goal. Each has its merits, so we will discuss them both, continuing with the example $a = 721$, $b = 136$.

Method 1: Back Substitution

Given that 1 = gcd (721, 136) we find integers u and v such that $1 = u \cdot 721 + v \cdot 136$. The steps we use correspond to the steps in the previous example, but include a little more information; in particular x div y as well as x mod y.

Step 1. $721 = 5 \cdot 136 + 41$,

Step 2. $136 = 3 \cdot 41 + 13$,

Step 3. $41 = 3 \cdot 13 + 2$,

Stpe 4. $13 = 6 \cdot 2 + 1$,

Step 5. $2 = 2 \cdot 1 + 0$,

Step 6. gcd (721, 136) = 1.

We now work backwards from step 4 (where the gcd 1 first appears) to progressively express the gcd as a linear combination of larger and larger x and y values until it is expressed as a linear combination of $x = a$ and $y = b$. The process is set out in Table 2.9 below.

Table 2.9: Expressing gcd(721,136) as $u \cdot 721 + v \cdot 136$ using back substitution

From Step	We get	and hence
4.	$1 = 13 - 6 \cdot 2$	
3.	$2 = 41 - 3 \cdot 13$	$1 = 13 - 6 \cdot (41 - 3 \cdot 13) = 19 \cdot 13 - 6 \cdot 41$
2.	$13 = 136 - 3 \cdot 41$	$1 = 19 \cdot (136 - 3 \cdot 41) - 6 \cdot 41 = 19 \cdot 136 - 63 \cdot 41$
1.	$41 = 721 - 5 \cdot 136$	$1 = 19 \cdot 136 - 63 \cdot (721 - 5 \cdot 136) = (-63) \cdot 721 + 334 \cdot 136$

From the table $u = -63$ and $v = 334$ is a solution to the problem.

As we have seen, the back substitution method requires us know x div y as well as x mod y for each step of Euclid's algorithm, so it would make sense to keep track of

these values as we go. In the second method that is exactly what we do, but it turns out that at the same time we can progressively build up to the values of u and v; *i.e.* instead of stepping forward through Euclid's algorithm and then stepping all the way back again we can do everything in the forward direction only.

Method 2: 'Magic Box' or Forward Method

The forward method works by expressing *each* remainder in the Euclidean algorithm in the form $ua+vb$, with the values of u and v changing at each step and culminating in the values we seek when the last non-zero remainder is reached. Let us denote by r_i the number in row i of the column of numbers that results from applying the Euclidean algorithm to a,b. So, in our running example (see Table 2.7) $r_1 = a = 721$, $r_2 = b = 136$ and for $i \geq 2$

$$r_{i+1} = r_{i-1} \bmod r_i = r_{i-1} - q_i r_i \quad \text{where} \quad q_i = r_{i-1} \operatorname{div} r_i.$$

$$\begin{aligned} \text{Then if} \quad r_{i-1} &= u_{i-1}a + v_{i-1}b \\ \text{and} \quad r_i &= u_i a + v_i b \\ \text{it follows that} \quad r_{i+1} &= (u_{i-1} - q_i u_i)a + (v_{i-1} - q_i v_i)b. \end{aligned}$$

Our magic box table has columns for the q's, r's u's and v's. To start the ball rolling row 1 has blank (q_1 is not defined), $r_1 = a$, 1, 0 (since $a = 1(a) + 0(b)$) and row 2 has $q_2 = a$ div b, $r_2 = b$, 0, 1 (since $b = 0(a) + 1(b)$), The full table for our (721,136) example is given as Table 2.10 at right. Note that in each row the first entry (q_i) is the 'multiplier' factor; it is multiplied the u and v values *in the same row* and the products subtracted from the u and v values in the previous row to get the u and v values in the next row. For example in row 4 the first entry is 41 div 13 = 3 and the last entry is $1 - (3)(-5) = 16$.

Table 2.10: Magic box for (721,136)

q	r	u	v
	721	1	0
5	136	0	1
3	41	1	−5
3	13	−3	16
6	2	10	−53
2	1	−63	334
	0		

Finding Inverses in Modular Arithmetic

One of the most important uses of the extended Euclidean algorithm is for finding modular inverses. Recall that given an integer a and a modulus $m \in \mathbb{N}$, an inverse of a modulo m is any solution s to the linear recurrence $ax \equiv 1 \pmod{m}$ and we then write $s = a^{-1} \pmod{m}$. Also recall that inverses exist if and only if a and m are coprime, and in that case the inverses form a congruence class modulo m. The 'standard' representative of this class, the integer b for which $0 \leq b < m$, is called the

preferred inverse of a modulo m and we write $b = a^{-1} \bmod m$.

From Theorem 62 we know that there exist integers u and v such that $gcd(m,a) = u \cdot m + v \cdot a$ and the extended Euclidean algorithm provides the means to find such u and v. In particular if $a^{-1} \bmod m$ exists then we can find u and v such that $u \cdot m + v \cdot a = 1$. Rearranging, we have

$$\begin{aligned} a \cdot v &= 1 + (-u) \cdot m \\ \text{and so } a \cdot v &\equiv 1 \pmod{m}, \\ \text{i.e. } v &\equiv a^{-1} \pmod{m}. \end{aligned}$$

The preferred inverse will just be $v \bmod m$.

Although perhaps the most straightforward and universal method, using the extended Euclidean algorithm is not the only way of computing modular inverses. A couple of other methods, one involving Chinese remainders, are demonstrated in the examples that follow.

Examples

18^{-1} mod 37 We are looking for an integer x that solves the linear congruence

$$18x \equiv 1 \pmod{37}.$$

Since 18 and 37 are coprime, we know that a solution exists. In fact we could just experiment using the integers between 1 and 37 to find an appropriate integer x, but this is not very efficient. A better alternative is just to try using the sorts of *ad hoc* methods we have used on other linear congruences. In this case we note that $2 \times 18 = 36 = 37 - 1$, so if we multiply the congruence by 2 we get

$$-x \equiv 2 \pmod{37}.$$

Thus $18^{-1} \equiv -2 \pmod{37}$ and hence $18^{-1} \bmod 37 = 35$. We say that -2 is *an* inverse of 18 modulo 37 but 35 is the *preferred* inverse because $0 \leq 35 < 37$.

Now let's solve the problem again using the **extended Euclidean algorithm method**. Remember that this method is completely *algorithmic*; just follow the recipe — no inspiration (like deciding to multiply by 2) is required. From the (very short) Table 2.11 we read off that

$$1 \cdot 37 - 2 \cdot 18 = 1.$$

Table 2.11: Magic box for (37,18)

q	r	u	v
	37	1	0
2	18	0	1
	1	1	-2

So $(-2)(18) = 1 - 37 \equiv 1 \pmod{37}$ taking us back to $18^{-1} \equiv -2 \pmod{37}$ and hence $18^{-1} \bmod 37 = 35$.

Another method for solving this inverse question makes use of Chinese remainders. Although in this case it probably takes a bit more work than either of the two previous methods, it is worth demonstrating here where large numbers don't obscure the basic idea. Let us reconsider the equation we we want to solve. We can write it in the form:

$$18x = 37n + 1$$

and try to first find n. We observe that the left hand side of this equation is divisible by both 9 and 2. (I have used 9 and 2 here rather than say 3 and 6, because 9 and 2 are relatively prime.) So we could actually consider the equation as two congruences:

$$\begin{aligned} 37n+1 \equiv 0 \pmod 2 \quad &\Leftrightarrow \quad 37n \equiv -1 \pmod 2 \quad \Leftrightarrow \quad n \equiv 1 \pmod 2, \\ 37n+1 \equiv 0 \pmod 9 \quad &\Leftrightarrow \quad 37n \equiv -1 \pmod 9 \quad \Leftrightarrow \quad n \equiv 8 \pmod 9. \end{aligned}$$

To find the solution for n now requires an application of the Chinese remainder theorem. The smallest positive integer satisfying these two congruences is $n = 17$. Hence $x = \dfrac{37 \cdot 17 + 1}{18} = 35$ as expected.

19^{-1} mod 141:

First note that this inverse exists since 19 is relatively prime to 141. For a first solution we use the standard method of employing the extended Euclidean algorithm on the pair $(141, 19)$. This is done in table 2.12. From the last line of this table we get

$$\begin{aligned} 19 \cdot 52 &= 141 \cdot 7 + 1 \\ &\equiv 1 \pmod{141} \end{aligned}$$

which tells us that an inverse of 19 modulo 141 is 52. Since, as it happens, $0 \leq 52 < 141$. 52 is in fact the preferred inverse of 19 modulo 141, *i.e.* $19^{-1} \bmod 141 = 52$.

Table 2.12: Magic box for (141,19)

q	r	u	v
	141	1	0
7	19	0	1
2	8	1	−7
2	3	−2	15
1	2	5	−37
2	1	−7	52
	0		

Now let's try the problem using a modification to the **Chinese remainders method** introduced in the previous example. We want to solve

$$19x = 141n + 1.$$

Since 19 is prime we use instead the factors of $141 = 3 \times 47$ to derive two congruences from the equation:

$$\begin{aligned} 19x \equiv 1 \pmod 3 \quad &\Leftrightarrow \quad x \equiv 1 \pmod 3 \\ 19x \equiv 1 \pmod{47} \quad &\Leftrightarrow \quad x \equiv ?? \pmod{47} \end{aligned}$$

The second of these congruences is still difficult to solve, since to solve it we need the inverse of 19 modulo 47. As both 19 and 47 are prime numbers, the only way we can easily solve the problem is to use the extended Euclidean algorithm. Since the numbers are a little smaller than the original numbers it is easier than using Table 2.12 above. The calculation is set out in Table 2.13, the last line of which gives

$$19 \cdot 5 = 1 + 47 \cdot 2$$

Table 2.13: Magic box for (47,19)

q	r	u	v
	47	1	0
2	19	0	1
2	9	1	−2
	1	−2	5

Returning to the original congruences we now have

$$x \equiv 1 \pmod 3 \quad \text{and} \quad x \equiv 5 \pmod{47}.$$

This is a Chinese remainder problem and we could use the CRT algorithm to solve it. However in this case it is easy to spot that $m = 5 + 47 = 52$ is a solution. This is the required inverse of 19 modulo 141.

Finally we demonstrate **an *ad hoc* method** which, for this example, is the simplest of all. Again we want to solve $19x = 141n + 1$. This time we consider the equation modulo 19 obtaining

$$0 \equiv 141n + 1 \equiv 8n + 1 \pmod{19} \quad \Leftrightarrow \quad 8n \equiv -1 \pmod{19} \quad \Leftrightarrow \quad 8n \equiv 18 \pmod{19}.$$

We can divide both sides of the last congruence by 2 to get

$$4n \equiv 9 \equiv 28 \pmod{19} \quad \Leftrightarrow \quad n \equiv 7 \pmod{19}.$$

When we put $n = 7$ in the original equation we get:

$$19x = 141 \cdot 7 + 1 = 988,$$

giving $x = 988/19 = 52$.

Encryption schemes using modular arithmetic

Cryptography is the study of coding and decoding messages with the aim of keeping the information in the message secure. With the increased use of electronic equipment in banking, for example, where large amounts of data ares kept in files or transmitted over communication lines, the security of this information is very important, and at present there is a keen interest in cryptography.
In this context a *plaintext* is an uncoded message and a *ciphertext* is a coded message. The process of transforming plaintext to ciphertext is called *encoding* or *encryption*, and the reverse process, that of transforming ciphertext to plaintext, is called *decoding* or *decryption*.
There are many methods of encryption and decryption. Since the advent of computers, cryptographic systems have become more and more mathematical, and in particular use modular arithmetic extensively.

A polygraphic System of Encoding

We first introduce a *polygraphic* system introduced by Hill [2], in which the plaintext is divided into groups of size $m \geq 2$ and transformed by a matrix. Let A denote the $m \times m$ invertible matrix with elements in $\mathbb{Z}_n$. Then the vector $y = Ax$ is the ciphertext of the plaintext x. We describe the process for the simplest case $m = 2$.

1. If the message is already a sequence of numbers, this step is unnecessary. But if the message is a sequence of letters we first need to convert it to numbers. To do this we can assume a one to one correspondence

A	B	C	D	$\ldots$	Z
00	01	02	03	$\ldots$	25

2. We need to choose a matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ where the entries$a, b, c, d \in \mathbb{Z}_{26}$, and the determinant $D = ad - bc$ satisfies the condition $\gcd(D, 26) = 1$. This ensures that the matrix A has an inverse (modulo 26).

3. We divide the plaintext into pairs of letters (because $m = 2$). If there is an odd number of letters we can add a dummy letter to the last one. We replace each letter with its numerical value.

4. Write each group in the plaintext as a column vector $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and form the product $y = Ax \bmod 26$.

5. Transform $y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$ into its letter equivalents.

To decode the received message, we use A^{-1} and compute $x = A^{-1}y \bmod 26$.

Example

As above we choose $m = 2$ and we work modulo 26. Choose $A = \begin{bmatrix} 4 & 5 \\ 3 & 8 \end{bmatrix}$.

To find A^{-1} recall that if a matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ and has determinant $D = ab - cd$, then $A^{-1} = D^{-1} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$. If working modulo 26 all these numbers are taken modulo 26. So in order to be able to find D^{-1}, D must be relatively prime to 26. The determinant of our A is 17 and gcd(17, 26) = 1. So A has an inverse in $\mathbb{Z}_{26}$. To compute A^{-1} we need to know 17^{-1} modulo 26. We can use one of the methods described earlier to find that $17^{-1} \equiv -3 \bmod 26$. Consequently

$$A^{-1} = \begin{bmatrix} 8 & -5 \\ -3 & 4 \end{bmatrix} \cdot -3 \mod 26$$

$$\text{and so } A^{-1} = \begin{bmatrix} 2 & 15 \\ 9 & 14 \end{bmatrix}.$$

Suppose we want to encrypt the message *HELP*. We split the text into two blocks *HE* and *LP*, and after substitution of their numerical values from $\mathbb{Z}_{26}$ we get

$$HE \leftrightarrow \begin{bmatrix} 07 \\ 04 \end{bmatrix} = x \quad \text{and} \quad LP \leftrightarrow \begin{bmatrix} 11 \\ 15 \end{bmatrix} = x'.$$

We compute Ax and Ax' in $\mathbb{Z}_{26}$ to get the ciphertexts:

$$y = Ax = \begin{bmatrix} 4 & 5 \\ 3 & 8 \end{bmatrix} \begin{bmatrix} 07 \\ 04 \end{bmatrix} \bmod 26 = \begin{bmatrix} 22 \\ 01 \end{bmatrix} \leftrightarrow WB.$$

Similarly,

$$y' = Ax' = \begin{bmatrix} 4 & 5 \\ 3 & 8 \end{bmatrix} \begin{bmatrix} 11 \\ 15 \end{bmatrix} \bmod 26 = \begin{bmatrix} 15 \\ 23 \end{bmatrix} \leftrightarrow PX.$$

Hence the coded text of *HELP* is *WBPX*.

How secure is this system? Because we encode blocks of letters rather than single letters, a simple count of the frequency of the occurrences of letters in the ciphertext will not help in decoding a message. However, because the transformation is linear, it is usually possible to recover the matrix A from a very small amount of deciphered text, and hence to decipher the rest of the document.

Suppose the only information that we know is that *HELP* is encoded to *WBPX*, or, in terms of numbers, $\begin{bmatrix} 07 \\ 04 \end{bmatrix} \to \begin{bmatrix} 22 \\ 01 \end{bmatrix}$ and $\begin{bmatrix} 11 \\ 15 \end{bmatrix} \to \begin{bmatrix} 15 \\ 23 \end{bmatrix}$.

We assume that we don't know what the encoding matrix A is. Let us make a matrix M, the message matrix, from the two message vectors: $M = \begin{bmatrix} 07 & 11 \\ 04 & 15 \end{bmatrix}$, and a similar matrix C from the encoded messages: $C = \begin{bmatrix} 22 & 15 \\ 01 & 23 \end{bmatrix}$. These matrices satisfy the relationship: $AM = C$, and if M^{-1} exists, then we can recover A from: $A = CM^{-1}$. In this case the determinant of M is 9, and so the matrix M has an inverse modulo 26.

$$M^{-1} = \begin{bmatrix} 15 & -11 \\ -4 & 7 \end{bmatrix} \cdot 9^{-1} \bmod 26 = \begin{bmatrix} 15 & 15 \\ 22 & 7 \end{bmatrix} \cdot 3 \bmod 26 = \begin{bmatrix} 19 & 19 \\ 14 & 21 \end{bmatrix}.$$

From here we can calculate

$$A = CM^{-1} = \begin{bmatrix} 22 & 15 \\ 01 & 23 \end{bmatrix} \begin{bmatrix} 19 & 19 \\ 14 & 21 \end{bmatrix} \mod 26 = \begin{bmatrix} 4 & 5 \\ 3 & 8 \end{bmatrix}.$$

This shows that this method of encryption is not secure. However, it has other interesting properties and a generalisation of it may be appropriate in some applications.

The polygraphic system we have considered is an example of a **private key cryptosystem.** If the encryption key is known it is easy to find the decryption key. When a private key system is used a pair of people who wish to communicate must have a separate key. The key must be exchanged securely since anyone knowing the key can encrypt and decrypt easily.

Public Key Cryptosystems: RSA

In the mid 1970's cryptologists introduced the concept of **public key encryption**. When such systems are used, knowing how to encrypt a message and send it to someone does not help to decrypt the message. In such a system every person can have a publicly known encryption key. Only the decryption keys are kept secret, and only the intended recipient of the message can decrypt it, for even if someone knows the encryption key it would take an enormous amount of work (such as 2 billion years of computer time) to find the decryption key.

In 1976 three researchers at M.I.T. – Ron Rivest, Adi Shamir and Len Adleman – introduced a public key encryption scheme, known as the **RSA** system, from the initials of the inventors. The RSA cryptosystem is based on exponentiation modulo the product of two large prime numbers. Each individual has an encryption key consisting of a modulus $n = pq$, where p and q are large primes, say with 100 to 200 digits each, and an exponent e that is relatively prime to $(p-1)(q-1)$. These are the public elements of the key.

The secret elements of the key, necessary for decryption, are the individual primes p and q (rather than their product), and an integer d which satisfies $ed \equiv 1$ modulo $(p-1)(q-1)$. (So $d = e^{-1} \mod (p-1)(q-1)$.) To send a message we translate the message to an integer M, the plaintext. We transform M to another integer C, the ciphertext, using the transformation

$$C = M^e \mod n.$$

The receiver decodes the message C by calculating $C^d \bmod pq$. The result is the original message M since

$$C^d = (M^e)^d = M^{ed} = M^{1+k(p-1)(q-1)}.$$

By a theorem, known as Fermat's Little Theorem, (assuming that gcd (M,p) = gcd$(M,q) =$ 1, which holds except in rare cases, especially when p and$q$ are very large prime numbers), it follows that $M^{p-1} \equiv 1 \bmod p$ and $M^{q-1} \equiv 1 \bmod q$. Consequently

$$C^d \equiv M \cdot (M^{p-1})^{k(q-1)} \equiv M \pmod{p}$$

and

$$C^d \equiv M \cdot (M^{q-1})^{k(p-1)} \equiv M \pmod{q}$$

Since gcd $(p,q) = 1$ it follows by the properties of congruences that

$$C^d \equiv M \bmod pq.$$

The belief that the RSA scheme is cryptographically strong (not vulnerable to attempts to discover the decryption key) is based on the presumed difficulty of factoring large numbers. Mathematicians have tried unsuccessfully for centuries to find efficient algorithms to factor integers. In 1988, however, a group of about 10 mathematicians using several hundred computers succeeded in factoring the 100-digit number $(11^{104}+1)/(11^8+1)$. They found that it is the product of two primes, one of 41 digits, one of 60 digits. They were clever in the use of computers, as they had to be, because standard methods would take an incredibly long time. In 1990 several hundred mathematicians, using about1000 computers, were able to factorise a certain number of 155 digits. It was the largest on a list called *The 10 Most Wanted Numbers*, which was set up to challenge experts. This feat will tend to make RSA users choose larger primes, and research other methods of encryption.

Example

Creating an RSA system: For demonstration purposes we create a 'toy' *RSA* system using tiny primes, say $p = 41$ and $q = 53$. (For a real, secure, the primes have to be vastly bigger than this — several hundred digits long in fact.) Then the public key $n = pq = 2173$. Also $(p-1)(q-1) = 40 \times 52 = 2080 = 32 \times 5 \times 13$. The other public key e must be relatively prime to $(p-1)(q-1)$, so let us choose $e = 49$. As creators of this system we need to calculate d satisfying $ed \equiv 1 \bmod 2080$, that is $49d \equiv 1 \bmod 2080$. Clearly d is the inverse, modulo 2080 of 49.

Possibly the easiest way to find d, in this situation, is to use the extended Euclidean algorithm, as in Table 2.14 at right.

Table 2.14: Finding $49^{-1} \bmod 2080$

q	r	u	v
	2080	1	0
42	49	0	1
2	22	1	−42
4	5	−2	85
2	2	9	−382
	1	−20	849

From the last line of the table we read off that

$$1 = (-20) \cdot 2080 + 849 \cdot 49$$

and hence that $49^{-1} \bmod 2080 = 849$.

So $d = 849$.

An alternative method to find d would be to use the Chinese Remainder Theorem. We need to solve $49d \equiv 1 \bmod 2080$. Using the coprime factors of 2080, this is equivalent to solving the simultaneous congruences:

$$\begin{aligned} 49d &\equiv 1 \pmod{32} \\ 49d &\equiv 1 \pmod{5} \\ 49d &\equiv 1 \pmod{13}. \end{aligned}$$

These simplify to

$$\begin{aligned} 17d &\equiv 1 \pmod{32} \\ 4d &\equiv 1 \pmod{5} \\ 10d &\equiv 1 \pmod{13}. \end{aligned}$$

Each of these amounts to finding an inverse. Some simple modular arithmetic leads to

$$\begin{aligned} d &\equiv 17 \pmod{32} \\ d &\equiv 4 \pmod{5} \\ d &\equiv 4 \pmod{13}. \end{aligned}$$

We then use the CRT algorithm to find that $d \equiv 849$ modulo 2080.

So we have set up an RSA system. The public keys are $n = 2173$ and $e = 49$ and we can send these to anyone who needs to send us a message. We keep to ourselves the identity of the primes p and q, and the integer d.

Encrypting a message using 'square and multiply':

We consider the process of sending a message using the toy RSA cryptosystem

just described. We know that $n = 2173$ and $e = 49$, and we want to send the message HELP. We first translate the letters of HELP into plaintext using the correspondence introduced earlier. We obtain

$$\text{HELP} \quad \Leftrightarrow \quad 0704 \quad 1115$$

We **encrypt** each block separately using the mapping $C = M^{49} \bmod 2173$. To calculate a high power like this efficiently we use the 'square and multiply' technique. Noting that $49 = 32 + 16 + 1$ we successively compute, by squaring then reducing modulo n, the powers M^2, M^4, M^8, M^{16} and M^{32}, and then use $M^{49} = M^{32}M^{16}M$. It goes like this:

$$\begin{aligned}
704^2 &\equiv 172 \pmod{2173} \\
704^4 \equiv 172^2 &\equiv 1335 \pmod{2173} \\
704^8 \equiv 1335^2 &\equiv 365 \pmod{2173} \\
704^{16} \equiv 365^2 &\equiv 672 \pmod{2173} \\
704^{32} \equiv 672^2 &\equiv 1773 \pmod{2173}
\end{aligned}$$

So

$$\begin{aligned}
704^{49} &= 704^{32} \cdot 704^{16} \cdot 704^1 \\
&\equiv 1773 \cdot 672 \cdot 704 \pmod{2173} \\
&\equiv 505 \pmod{2173}.
\end{aligned}$$

Similarly,

$$\begin{aligned}
1115^2 &\equiv 269 \pmod{2173} \\
1115^4 \equiv 269^2 &\equiv 652 \pmod{2173} \\
1115^8 \equiv 652^2 &\equiv 1369 \pmod{2173} \\
1115^{16} \equiv 1369^2 &\equiv 1035 \pmod{2173} \\
1115^{32} \equiv 1035^2 &\equiv 2109 \pmod{2173}
\end{aligned}$$

So

$$\begin{aligned}
1115^{49} &= 1115^{32} \cdot 1115^{16} \cdot 1115^1 \\
&\equiv 2109 \cdot 1035 \cdot 1115 \pmod{2173} \\
&\equiv 497 \pmod{2173}.
\end{aligned}$$

Thus the encrypted massage is 0505 0497.

Decrypting a message using 'square and multiply':

Finally we illustrate what happens when a message encrypted by our toy RSA system is received. Suppose that the received message is 1516 1362. As creators of this system we know the values of $p = 41$ and $q = 53$, and $d = 849$, as well as the public keys $n = 2173$ and $e = 49$. To decrypt the message, for each block C of the code, we calculate $C^d \bmod 2173$. Since d is larger than e this calculation is a little longer than the calculation for encrypting a message in this system. However, the calculations are no more difficult. We first note that

$$849 = 512 + 256 + 64 + 16 + 1.$$

Our calculations go as follows:

$$\begin{aligned}
1516^2 &\equiv 1395 \pmod{2173} \\
1516^4 \equiv 1395^2 &\equiv 1190 \pmod{2173} \\
1516^8 \equiv 1190^2 &\equiv 1395 \pmod{2173} \\
1516^{16} \equiv 1395^2 &\equiv 2010 \pmod{2173} \\
1516^{32} \equiv 2010^2 &\equiv 493 \pmod{2173} \\
1516^{64} \equiv 493^2 &\equiv 1846 \pmod{2173} \\
1516^{128} \equiv 1846^2 &\equiv 452 \pmod{2173} \\
1516^{512} \equiv 452^2 &\equiv 1764 \pmod{2173}
\end{aligned}$$

So

$$\begin{aligned}
1516^{849} &= 1516^{512} \cdot 1516^{256} \cdot 1516^{64} \cdot 1516^{16} \cdot 1516^{1} \\
&\equiv 1764 \cdot 452 \cdot 1846 \cdot 2010 \cdot 516 \pmod{2173} \\
&\equiv 206 \cdot 1846 \cdot 2010 \cdot 516 \pmod{2173} \\
&\equiv 1 \cdot 2010 \cdot 516 \pmod{2173} \\
&\equiv 614 \pmod{2173}
\end{aligned}$$

Note that throughout these calculations we only perform one multiplication (or squaring) before reducing modulo n. This stops the sizes of numbers getting out of hand.

A similar calculation shows that $1362^{849} \equiv 1403 \bmod 2173$.

So our decoded message is 0614 1403 and this translates to the word GOOD.

References for RSA

.

[1.] Denning, D.E.R., *Cryptography and Data Security*, Addison-Wesley, Reading, Mass., 1982

[2.] Hill, L.S. *Cryptography in an algebraic alphabet,* Am. Math. Monthly, **36**, (1929) 306 – 312.

[3.] Rivest, R, Shamir, S, and Adleman, L, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the Association for Computing Machinery, **31**, (1978), 120 – 128.

END OF CHAPTER TWO

3. LANGUAGES AND GRAMMAR

How is language constructed? We need only think about the languages we speak to make us realise how complicated the structure of a language can be. Indeed linguists spend much time analysing languages, comparing the structure of different languages and trying to decide what makes a language.
We shall investigate some "formal" or artificial languages. Such formal languages were developed as a result of research in the automatic translation of one language to another. Whereas the rules of a natural language are very complex and difficult to characterise completely, the rules by which a formal language is constructed are specified completely. In later chapters we shall also study finite state machines and regular expressions. Finite state machines can be used to describe the effect of computer programs and are useful in the study of formal languages. Regular expressions and regular sets play a most important role in the study of formal languages and finite state machines.

3.1 Formal Languages

Syntax of Statement Logic

The first example we consider, the language of statement logic (also known as Propositional Calculus) should be familiar to you, but we approach it differently. In studying logic previously you would have been concerned mainly with the truth or falsity of compound statements, taking for granted the **readability** of these statements. You would have used parenthesises when necessary, but dropping them according to certain conventions that ensured the readability of a statement. Here "readability" means the ability to determine the structure of a formula without ambiguity. This is easy for some statements such as $(p \rightarrow q)$, but becomes more complicated for longer strings such as

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

Before deciding the truth value of string like this, it is necessary to determine whether it is meaningful or not. A computer can do this if it is given precise rules about the structure of acceptable strings.

In describing a formal language we need to start with an alphabet, which we usually designate by Σ. Usually Σ is a collection of digits or letters or other symbols, but it can be a collection of words. Then Σ^* is the set of all strings (or words), including the empty string ε, over Σ.

63 Definition. *The **alphabet**, Σ, for statement logic comprises the following symbols:*

$p, q, r, \ldots$	*(infinitely many **variable symbols**)*
T, F	*(two **constant symbols**)*
$\wedge, \vee, \to, \leftrightarrow$	*(four **binary connecting symbols**)*
$\neg$	*(a **unary symbol**)*
$(,)$	*(left and right **parentheses**)*

*These are the **symbols of statement logic**.*

Having decided on the alphabet for statement logic we need to decide on the set of rules, or the **grammar,** which identifies the strings we consider part of the **language** of statement logic. It is clear that not all strings in $\Sigma^\star$ are acceptable as formulas in this language, any more than any string of words is acceptable as a sentence in the English language. The following definition is one way of describing which strings are acceptable as formulas.

64 Definition. *A **formula** (or word) in statement logic is a string on the symbols of statement logic that can be obtained by using the following rules:*

1. *Any constant or variable symbol is a formula. Thus $p, q, r, \ldots$ are formulas, as are the constant symbols T and F.*
2. *If α and β are formulae, so are*

$$(i)\ (\alpha \wedge \beta), \quad (ii)\ (\alpha \vee \beta), \quad (iii)\ (\alpha \to \beta), \quad (iv)\ (\alpha \leftrightarrow \beta), \quad (v)\ \neg\alpha.$$

Since α is a string, its length can be defined.

65 Definition. *The length of a formula α, written length(α) is defined as follows:*

1. *If α is a constant or variable symbol, then length$(\alpha) = 1$.*
2. *Suppose β and γ are formulas, with length $(\beta) = m$ and length$(\gamma) = n$.*
   (a) *If α is $(\beta \wedge \gamma)$, $(\beta \vee \gamma)$, $(\beta \to \gamma)$, or $(\beta \leftrightarrow \gamma)$, then length$(\alpha) = m + n + 3$.*
   (b) *If $\alpha = \neg\beta$, then length$(\alpha) = m + 1$.*

Examples

Let $\alpha = (p \wedge q)$, $\beta = ((p \wedge q) \to t)$, and $\gamma = (((p \wedge q) \to t) \vee \neg(p \wedge q))$, then

- length(α) = length $(p) + length(q) + 3 = 5$,
- length(β) = length$(\alpha \to t)$ = length(α) + length(t) + 3 = 9, and
- length(γ) = length$(\beta \vee \neg\alpha)$ = length(β) + length$(\neg\alpha)$ + 3 = 9 + 6 + 3 = 18.

In Definition 64 it is important to note the introduction of the parentheses. These are part of the definition of a formula, and it is the parentheses that allow us to uniquely deconstruct a formula into its components. This deconstruction process is called *parsing*. The user cannot use parentheses at his or her discretion. So $p \wedge q$ and $(\neg q)$ are not formulas, whereas $\neg p$ is a formula.
With these definitions we can determine some important properties of formulas We note the following:

Properties

1. Any formula has an equal number of right and left parentheses.
2. Let α be a formula. Then α is precisely one of the following types. In each case α' and β' are uniquely determined formulas.
   (i) $\alpha = (\alpha' \wedge \beta')$, (ii) $\alpha = (\alpha' \vee \beta')$ (iii) $\alpha = (\alpha' \to \beta')$
   (iv) $\alpha = (\alpha' \leftrightarrow \beta')$, or (v) $\alpha = \neg\alpha'$.

It is these properties, especially the second, that allow us to determine that any formula is uniquely readable. This means that we can give a string to a computer and ask it to decide whether the string is meaningful or not. Given a long string α, we can take it apart in only one way and write it as $\alpha = (\alpha' \ \lambda \ \beta')$, where λ is one of the binary connectives, or as $\alpha = \neg\alpha'$. We continue this process until we have reached the variable

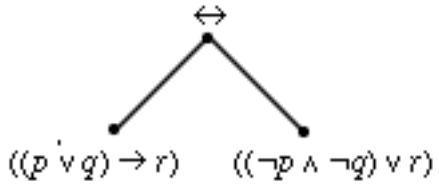
or constant symbols. One method of doing this is to make a tree diagram representing the formula. The way of doing this is to keep track of the excess of left parentheses over right parentheses, after the first left parenthesis. When matching occurs, we get the initial string.

Example

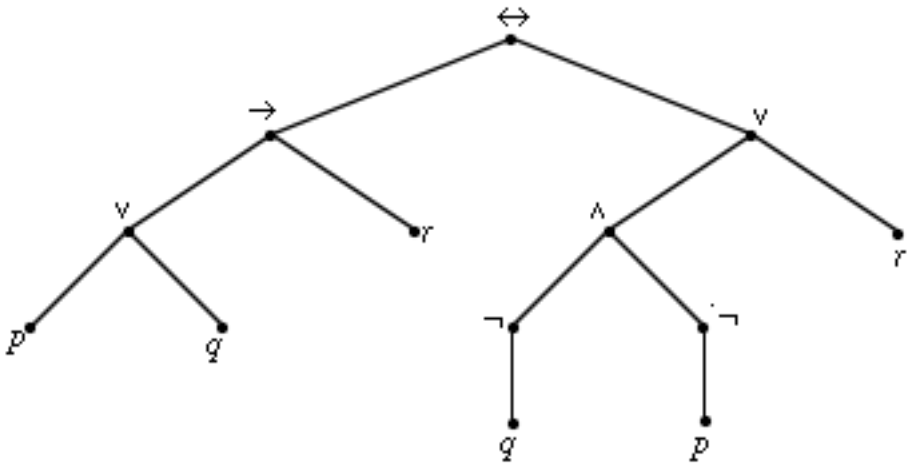
Let $\alpha = (((p \vee q) \rightarrow r) \leftrightarrow ((\neg q \wedge \neg p) \vee r))$. We find the tree representing α, and in doing so we show that α is a formula.
In matching parentheses, we find the first left parenthesis. This will be the first symbol, unless we have one or more negation signs at the beginning of the formula. We number the first left parenthesis 0. Unless the formula is a very simple one, such as $(p \rightarrow q)$, the next parenthesis will also be a left parenthesis. We now add 1 for each left parenthesis, and subtract one for each right parenthesis. When we reach 0 again, we have found the first segment α', of the formula. This right parenthesis, now numbered 0, should be followed by a binary connective. If this is not the case the string is not a formula. What follows the binary connective forms the second segment of the formula, β'. Once we have found α' and β', we can continue the process. For example:

$$\begin{array}{cccccccccccccccccccc} \alpha & = & (& (& (& p & \vee & q &) & \rightarrow & r &) & \leftrightarrow & (& (& \neg & q & \wedge & \neg & p &) & \vee & r &) &) \\ & & 0 & 1 & 2 & & & & 1 & & & 0 & & & & & & & & & & & & \end{array}$$

We have $\alpha' = ((p \vee q) \rightarrow r)$ and $\beta' = ((\neg q \wedge \neg p) \vee r)$, and $\alpha = (\alpha' \leftrightarrow \beta')$. We represent this in a tree indiagram below:



We proceed in a similar way for both α' and β', finally producing the following tree.



You can see that when there is a negation sign, the branch of the tree drops, rather than dividing into two parts. If the string with which we are working is not a formula, at some stage this process will fail. You will also note that given a tree it is possible to work backwards and find the formula it represents.

The above discussion does not provide a *completely* mechanical process for deciding whether a string is a formula of statement logic, as, for example, it does not rule out strings like pq or $(pq \wedge r)$. We will however examine some complete decision processes for other formal languages in the next sections.
We study several formal languages in more detail. These languages are defined by recurrence relations. One of the questions we will be concerned with is that of determining whether or not a given string belongs in the language we have described. Often this is quite easy to determine, but at other times it is more difficult. With one of our examples, we will explore a little further and count the number of strings of a given length. This introduces some important techniques of discrete mathematics.

3.2 Language Alpha

The language α is built from the alphabet $\Sigma = \{a, b, c\}$.
In the following rules x denotes an arbitrary string of a's, b's and c's.

Rules Defining Language α

- Rule $(\alpha\, 0)$: The empty string ε is in Language α.
- Rule $(\alpha\, 1)$: If x is in Language α, so is xa,

- Rule $(\alpha\ 2)$: If x is in Language α, so is xab,
- Rule $(\alpha\ 3)$: If x is in Language α, so is xbc.

Using these rules it is quite easy to produce strings that belong in languageα. For example, all the following strings are in language α:

ε	rule $(\alpha\ 0)$
a	rule $(\alpha\ 1)$ applied to ε
aab	rule $(\alpha\ 2)$ applied to the previous string
$aabbc$	rule $(\alpha\ 3)$
$aabbcbc$	rule $(\alpha\ 3)$ again.

Of course, many more strings can be produced in this way, and Language α is the set of all such strings.

As you can see from the rules for constructing language α, the words that can be produced seem to have very little pattern about them, and so can't be described in a simple, concise way. This makes it important to have some sort of algorithm so that given a word, there is a way of deciding whether or not the word belongs to the language. In fact, if we work backwards we find it is not so difficult to make this decision. We can do this because any new symbols are always added from the right, and the rules allow us to decide unambiguously which rule was the last to be used.

For example, take the word $w = aababbcaa$. Here the last symbol to be added to make w, was the symbol a, using rule $(\alpha\ 1)$. We can remove the symbol a, leaving $aababbca$. Again the last symbol to be added was a, which can again be removed, leaving $aababbc$. At this stage the only rule that can have been used was $(\alpha\ 3)$, adding bc. When we remove that we get $aabab$. We continue in this way until we get to the empty string.

$$aababbca\boldsymbol{a} \overset{(\alpha 1)}{\to} aababbc\boldsymbol{a} \overset{(\alpha 1)}{\to} aabab\boldsymbol{bc} \overset{(\alpha 3)}{\to} aab\boldsymbol{ab} \overset{(\alpha 2)}{\to} a\boldsymbol{ab} \overset{(\alpha 2)}{\to} \boldsymbol{a} \overset{(\alpha 1)}{\to} \varepsilon.$$

It is because we finally reach the empty word that we decide that $w = aababbcaa$ belongs in language α. If we start with a word that doesn't belong in Language α, this procedure doesn't terminate in the empty string.

$$abbaabca \to abbaabc \to abbaa \to abba \to abb \to ?$$

Here there is no rule that allows to remove any further symbols, so we conclude that $abbaabca$ is not a word in Language α.

The technique used here can be formalised into a **decision algorithm** for language α. We define *a*, *ab* and *bc* to be *good suffixes*, and as long as a good suffix can be removed the algorithm does so. If it can't remove a good suffix, because only the empty string is left, the word belongs in α. Otherwise the word does not belong in α.

Decision Algorithm for Language α

decision: = ?

while (decision = ?) {

if (string = ε), decision → YES;

else {

if (string has good suffix) remove suffix:

else decision → NO:}

}

print decision; }

3.3 Language Beta

Language β has alphabet $\Sigma = \{(,)\}$, which is just the set of left and right parentheses. In mathematics, in particular, parentheses are important as they enable us to interpret mathematical expressions in an unambiguous way. Different ways of arranging parentheses will often give a different meaning to an expression, and some arrangements of parentheses mean that an expression is meaningless, or that a mistake or typo has been made and needs correction. For example,

$$((x^2+y)+z^2) \text{ is quite different from } (x^2+(y+z)^2).$$

However, expressions like

$$(x(^2+y)+z^2) \text{ and } ((x^2+y(+z^2($$

do not really have any meaning. The lack of meaning here depends partly on the placement of the parentheses, and partly on the lack of balance in using the parentheses. In our language β, we will consider strings of parentheses that correspond to correct usage of parentheses in mathematical expressions as far as the balance of parentheses is concerned.

Rules Defining Language β

- Rule (β 1): The empty string ε is in language β.
- Rule (β 2): If x is in language β, so is (x).
- Rule (β 3): If x and y are in language β, so is xy.

It is fairly easy to see that expressions like (())() and ((())(()))() are in language β, while ())(and (()(()() are not, and we again look for a decision algorithm for language β.
It is easy to see that any expression in language β must have the same number of right and left parentheses, but this property is not quite enough to determine whether or not a string of parentheses is in β. For example, the string ())(is not in β, even though it has the same number of right and left parentheses. As we move from left to right we realise we have too many right parentheses, until we get to the final left parenthesis. This helps us determine a couple of other properties that will define these strings. One property is that any string in β must finish with a right parenthesis. The second is that as we count from the left, we must always have at least as many left parentheses as right parentheses.
So suppose algorithm β is given a string of left and right parentheses. It will read the string from left to right, one symbol at a time, indicating either YES or NO and then halting. The key to the operation is to introduce a variable COUNT. This is an integer indicating the current balance between left and right parentheses. Initially at step 1, COUNT is set to 0. When a left parenthesis is read COUNT is increased by 1, and when a right parenthesis is read, it is decreased by 1. At all times

COUNT = No. of left parentheses read – No. of right parentheses read.

If a string is in language β, COUNT will always be greater than or equal to 0. If COUNT ever becomes negative, the string cannot be in language β, and the algorithm must print NO. If COUNT doesn't become negative, the algorithm must read all the symbols in the string before it makes its decision. After all the symbols have been read, if COUNT is not 0, then the number of left and right parentheses is not equal, and the algorithm returns NO. If COUNT doesn't become negative and at the end of the string COUNT = 0, the algorithm returns YES and the string is in language β.

Decision Algorithm for Language β

COUNT:= 0;

```
while {COUNT ≥ 0 and not EOS}
   if (next symbol = "("
      COUNT = COUNT + 1;
      else
         COUNT= COUNT −1;
if (COUNT = 0)
      print YES;
else
print NO;
```

Counting Strings in Language Beta

Language β clearly contains no strings of odd length, and it is an interesting exercise to count the number of strings that can be legally made of any given length. We prove the following theorem.

66 Theorem. *Language β contains no strings of odd length, and exactly*

$$\frac{1}{n+1}\binom{2n}{n}$$

strings of length 2n.

This formula predicts that when $n = 1$, there is exactly 1 string of length 2, which, of course is, (). When $n = 2$, the formula predicts that there are 2 strings of length 4. These are ()() and (()). When $n = 3$, it predicts that there are 5 strings of length 6:

$$()()(), (())(), ()(()), (()()), \text{ and } ((())).$$

The formula also predicts that there are 14 strings of length 8, 42 strings of length 10 and 132 strings of length 12 in language β.

This just gives an indication that the theorem is likely to be true. Now we need to prove that it is true for every integer n. The method of proof uses a technique that translates strings of left and right parentheses into geometric figures called **lattice paths**. The idea is to start at a fixed point A, usually $(0,0)$, and for each parenthesis in the string add one diagonal edge to the path. In the case of a left parenthesis the edge should go one unit diagonally in a *northeast* direction; for a right parenthesis, the edge should go

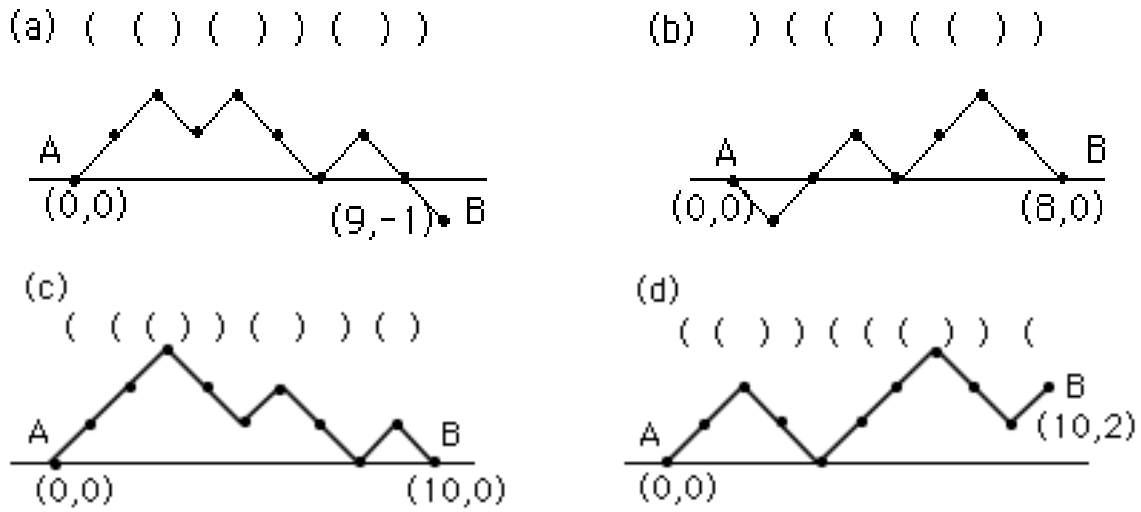


Figure 3.1: Some Lattice Paths.

southeast. After all the symbols in a string have been read, the lattice path will have reached another point, say, *B*. Some lattice paths are illustrated in Figure 3.1

We have introduced Cartesian coordinates as this makes the discussion easier. The point *A* is taken to be the origin with coordinates (0, 0). A left parenthesis corresponds to increasing the x coordinate by 1 and the y coordinate by 1. A right parenthesis corresponds to increasing the x coordinate by 1, but decreasing the right coordinate by 1. Hence the x coordinate counts the number of parentheses in the string, while the y coordinate corresponds to the variable COUNT in the decision algorithm for language β, and so counts the difference between the number of right and left parentheses.

Using the lattice paths it is possible to tell by inspection whether or not the corresponding string is in language β. It is clear that a lattice path represents a string in language β if it terminates on the x-axis (that is, the y coordinate of the end point is 0), and if it never goes below the x-axis (the y coordinate is always positive or 0). This is because of the relationship between the y coordinate and the variable COUNT.

With these conventions the following theorem is equivalent to Theorem 66.

67 Theorem. *The number of lattice paths which begin at (0, 0) and terminate at* $(2n,0)$ *without ever going below the x-axis is*

$$\frac{1}{n+1}\binom{2n}{n}.$$

Proof. We need some terminology. We call a lattice path that satisfies the description in the theorem, "legal". Otherwise it is "illegal". We want to count the number of legal paths, and to do this we count the total number of paths from (0, 0) or A, to $(2n,0)$ or B. We also count the total number of illegal paths from A to B. Then

$$\text{total number of paths} = \text{number of legal paths} + \text{number of illegal paths}. \tag{3.1}$$

Counting the total number of paths from A to B is quite easy. Every path from A to B contains $2n$ edges, n upward edges and n downward edges. The path is uniquely specified by the position of the n upward edges. The number of ways of selecting n positions for the upward edges from the possible $2n$ locations is given by $\binom{2n}{n}$, so

$$\text{total number of paths from } A \text{ to } B = \binom{2n}{n}. \tag{3.2}$$

We now want to count the number of illegal paths from A to B, and this seems a more difficult problem. Fortunately we can simplify the problem.
As Figure 3.2 indicates, every illegal path from A to B must either touch, or cross, the line $y = -1$. So every illegal path will have a *first point of contact* with the line $y = -1$.

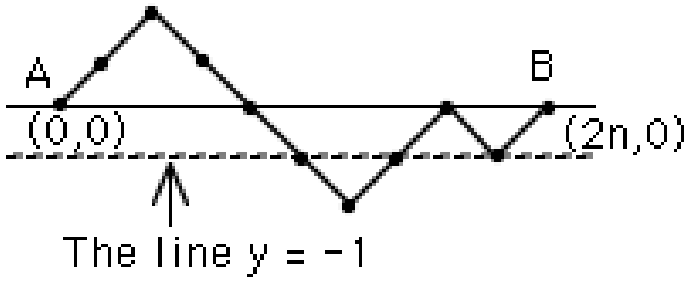


Figure 3.2: An Illegal path fro A to B.

In Figure 3.2 the first point of contact is $(5,-1)$. We reflect the **first part** (from A to the first point of contact with $y = -1$) of the illegal path with respect to the line $y = -1$ to get a path from A' to B, where A' is the point $(0,-2)$. This is illustrated in Figure 3.3 below.
It is also true that every path from A' to B must pass through the line $y = -1$, and if its first part is reflected, we obtain an illegal path fro A to B. This argument shows that

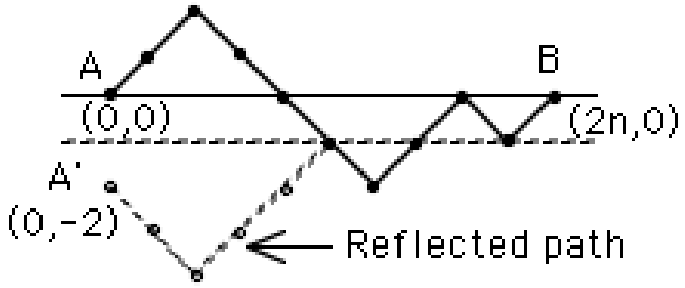


Figure 3.3: The reflected path.

there is a one to one correspondence between the set of illegal paths from A to B, and the set of all paths fromA' to B:

$$\text{number of illegal paths from } A \text{ to } B = \text{total number of paths from } A' \text{ to } B. \tag{3.3}$$

It is now quite easy to count the total number of paths from A' to B. Every path from A' to B must contain $2n$ edges. Since the *rise* of such a path is 2, (it starts at $y=-2$ and ends at $y=0$), there must be $n+1$ upward edges and $n-1$ downward edges. There are exactly $\binom{2n}{n+1}$ such paths:

$$\text{total number of paths from } A' \text{ to } B = \binom{2n}{n+1}. \tag{3.4}$$

Finally, we count the number of legal paths from A to B:

$$\begin{aligned}
\text{number of legal paths} &= \text{total no. of paths} - \text{no. of illegal paths} && \text{from equation 1}\\
&= \tbinom{2n}{n} - \text{no. of illegal paths} && \text{from equation 2}\\
&= \tbinom{2n}{n} - \text{no. of paths from } A' \text{ to } B && \text{from equation 3}\\
&= \tbinom{2n}{n} - \tbinom{2n}{n+1} && \text{from equation 4}
\end{aligned}$$

Hence the total number of legal paths from A to B is $\binom{2n}{n} - \binom{2n}{n+1}$. This corresponds exactly to the number of strings of length $2n$ in language β. A short exercise in algebra shows that this is equal to $\frac{1}{n+1}\binom{2n}{n}$:

$$\begin{aligned}
\tbinom{2n}{n} - \tbinom{2n}{n+1} &= \frac{(2n)!}{n!n!} - \frac{(2n)!}{(n+1)!(n-1)!} = \frac{(2n)!}{n!(n-1)!}\left(\frac{1}{n} - \frac{1}{n+1}\right)\\
&= \frac{(2n)!}{n!(n-1)!}\,\frac{1}{n(n+1)} = \frac{1}{n+1}\,\frac{(2n)!}{n!n!} = \frac{1}{n+1}\tbinom{2n}{n}.
\end{aligned}$$

□

3.4 Language Gamma

The language γ is the invention of Douglas Hofstadter in his Pullitzer prize winning book, *Gödel, Escher, Bach: An Eternal Golden Braid* (New York Books, 1979). It is an example of a language for which it is quite difficult to find a decision algorithm. The alphabet for language γ is given by $\Sigma = \{M, I, U\}$ and following Hofstadter we define the language γ by certain rules.

Rules Defining Language γ

- Rule $(\gamma\,0)$: The string MI is in Language γ.
- Rule $(\gamma\,1)$: If the string xI is in Language γ, so is the string xIU.
- Rule $(\gamma\,2)$: If the string Mx is in Language γ, so is the string Mxx.
- Rule $(\gamma\,3)$: If the string $xIIIy$ is in Language γ, so is xUy.
- Rule $(\gamma\,4)$: If the string $xUUy$ is in Language γ, so is the string xy.

(*Note that, in Rules* $(\gamma\,1)$ — $(\gamma\,4)$, x *and* y *are any strings in* $\Sigma^\star$ *(not necessarily in language* γ*) and may be empty*.)
These rules can be written more simply as follows:

$$(\gamma\,0)\quad MI \qquad (\gamma\,1)\quad xI \to xIU \qquad (\gamma\,2)\quad Mx \to Mxx$$

$$(\gamma\,3)\quad xIIIy \to xUy \qquad (\gamma\,4)\quad xUUy \to xy.$$

At this stage it is good to experiment to find what sort of words belong in the language γ. Each of the following words is in language γ and after the first each is derived from the preceding one by the rule indicated.

MI	$(\gamma\,0)$
MII	$(\gamma\,2)$
$MIIII$	$(\gamma\,2)$
MUI	$(\gamma\,3)$
$MUIUI$	$(\gamma\,2)$
$MUIUIU$	$(\gamma\,1)$
$MUIUIUUIUIU$	$(\gamma\,2)$
$MUIUIIUIU$	$(\gamma\,4)$

In principle, the strings can be displayed in a tree structure, showing the different generations of strings. We can say MI is born on day 0. Its day 1 descendants are MIU and MII. There are no other possibilities. The day 2 descendants of MI are $MIUIU$ from MIU, and $MIIU$ and $MIIII$ from MII. There are six day 3 descendants of MI: $MIUIUIUIU, MIIUIIU, MIIIIU, MIIIIIIII, MUI$ and MIU (born again). All but the first two of these strings are descended from $MIIII$. We also see that MIU is both a day 1 descendent and a day 3 descendent of MI. We can also see that there will be many more day 4 descendants of MI.
This is enough experimenting to indicate that it is quite difficult to determine whether or not any given string in $\Sigma^\star$ is in γ. However, some properties of strings in γ are easy to determine. The following is fairly obvious.

68 Theorem. *Every string in language γ begins with M and has no other M in it.*

The formal proof of this statement is left as an exercise. This immediately allows us to eliminate many strings, such as $I, IM, MIIM$ and many others, as words in γ.
Where do we go from here? Many questions raise themselves at this stage, but without some direction it is possibly difficult to know the best path to take. One of the directions we can take is to look at the number of I's in any string that belongs to γ. In any of the words we have constructed above we have 1, 2, 4 or 8 I's. This gives some clue, but perhaps we need to ask some more questions. Does MU belong in γ? Do $MIII$ or $MIIIII$ belong in γ? It is difficult to see any way of making MU or $MIII$, although this doesn't necessarily mean that they don't belong in γ. On the other hand it is possible to construct $MIIIII$, as we show below.

$$MI \to MIIII \to MIIIIIII \to MIIIIIIIU \to MIIIIIUU \to MIIIII.$$

It is easy to work out which rule is used at each stage. It gives some indication that the next theorem is plausible.

69 Theorem. *In any string in Language γ, the number of I's cannot be a multiple of three.*

Proof. We use mathematical induction to prove this theorem. Let C_n be the statement defined below.

C_n = In any string born on day n, the number of I's cannot be a multiple of three.

We want to prove that this statement is true for all $n \geq 0$. It is clear that C_0 is true, since the only day-0 string is MI. From the strings we have constructed it is also clear that C_1, C_2 and C_3 are also true, although we don't really need to know this to use induction. We make the *induction hypothesis* that $C_0, C_1, \ldots, C_n$ are all true. This means that we are assuming the number of I's in every word in language γ that is born on day n, or earlier, is not divisible by three. Using this assumption we aim to prove C_{n+1}, that is, that the number of I's in any string born on day $n+1$ is not a multiple of three. Let w be any string born on day $n+1$. Then w must come from a string born on day n with the application of one of the rules (γ 1), (γ 2), (γ 3), or (γ 4). Let us consider these possibilities.

- (γ 1): If w is born on day $n+1$, via the rule (γ 1), then $w = xIU$, where xI is a string in language γ born on day n. Therefore, the number of I's in xI is not a multiple of three. Also the number of I's in w is exactly the same as the number of I's in xI. So the number of I's in w is not a multiple of three.

- (γ 2): In this case w is of the form Mxx, where Mx is in language γ and born on day n. Let the number of I's in Mx be k. Then k is not divisible by three. The number of I's in Mxx is twice the number in Mx and so is $2k$. But if k is not divisible by three, neither is $2k$.

- (γ 3): In this case w is of the form xUy, where $xIIIy$ is a string in language γ born on day n. The number of I's in $xIIIy$ is therefore not a multiple of three. But xUy has three less I's than $xIIIy$, so the number of I's in xUy is also not divisible by three.

- (γ 4): In the final case w is of the form xy, where $xUUy$ is a string in language γ born on day n. By the induction hypothesis, the number of I's in $xUUy$ is not a multiple of three. But the number of I's in xy is exactly the same as the number of I's in $xUUy$, and is not a multiple of three.

These are the only ways in which w can be born on day $n+1$, according to the rules of language γ. Hence, using the induction hypothesis we have proved the number of I's in any word, w, born on day $n+1$, is not divisible by three. Finally by mathematical induction we have that the number of I's in any string in language γ is not divisible by three. □

We were able to count the number of strings in language β, but it is not so easy to do the same for language γ. Let us just find the number of strings of length three in γ.

Any string in language γ is formed from the letters M, I and U. The multiplication rule for counting tells us that there are $3 \times 3 \times 3 = 27$ possible strings of length three in the letters M, I and U. We want to sort out exactly how many of these belong in language γ. Theorem 68 says that a string in language γ must begin with M and have no other occurrences of M in it. This means that the number of possible strings is reduced to $1 \times 2 \times 2 = 4$, and these areMII, MIU, MUI and MUU. From our previous exploration we have seen that MII and MIU are born on day 1, and MUI is born on day 3. The string MUU did not appear anywhere, and since the number of I's is a multiple of three (3×0), MUU is not in language γ. So there are only three string of length three in γ. We could continue this process, but it would become quite difficult, especially since we are not able to decide easily whether or not some strings belong in language γ. We know from Theorems 68 and 69 that some strings don't belong in language γ, but we don't know whether all other strings do belong in language γ. We state some further theorems, ending with one that clearly describes words in language γ. We omit the proofs.

70 Theorem. **(i)** *Any string of the form $MII\ldots I(2^n$ I's) is in Language γ.*

(ii) *If a string $xIII$ is in language γ, then so is x.*

(iii) *Any string of the form MI^N, where N is an integer that is not a multiple of three, is in language γ.*

71 Theorem. *Let $\Sigma = \{M, I, U\}$. A string in $\Sigma^\star$ is in language γ if and only if it begins with M and is followed by a string of I's and Us where the number of I's is not a multiple of three.*

Note that Theorem 71 allows us to count the number of words in γ of any given length. For example for the number of words of length 5 we observe that the first letter must be M and the remaining four letters can each be either U or I, giving $2^4 = 16$ strings to start with. From these 16 strings we must remove those with 0 or 3 I's. There is only one with 0 I's, *viz.* $MUUUU$ and four with three I's: $MUIII$, $MIUII$, $MIIUI$ and $MIIIU$. So there are $16 - 1 - 4 = 11$ words of length 5 in language γ.

3.5 Phrase Structure Grammars

There is an alternative way in which the language of propositional calculus can be produced. We us the same alphabet Σ, which contains the variable and constant symbols, the binary logical connectives, the unary logical connective and the left and right

parentheses. In this situation Σ can be thought of as the terminal alphabet. Its members will make up the resulting formulas or words, as in the previous method. We introduce a new set $N = \{S, B\}$, which is the non-terminal alphabet. Its members are used to generate the word patterns. Within N, S is a special element called the start symbol. Finally we introduce P, a finite set of rules or **productions**. These govern the generation of word patterns. Each production is of the form $\alpha \Rightarrow \beta$, where α is a string in $(\Sigma \cup N)^\star$ that contains at least one member of N, $\beta \in (\Sigma \cup N)^\star$ and $\alpha \Rightarrow \beta$ indicates that α can be replaced by β. The productions for the language of propositional logic are

$$S \Rightarrow (SBS) \tag{3.5}$$

$$S \Rightarrow \neg S \tag{3.6}$$

$$S \Rightarrow x (x \text{ any variable or constant symbol}) \tag{3.7}$$

$$B \Rightarrow \wedge;\ B \Rightarrow \vee;\ B \Rightarrow \rightarrow; B \Rightarrow \leftrightarrow \tag{3.8}$$

We may abbreviate these productions as

$$S \Rightarrow SBS \mid \neg S \mid x \qquad B \Rightarrow \wedge \mid \vee \mid \rightarrow \mid \leftrightarrow$$

where the symbol " $\mid$ " is read as "or".
If we start with S and use these productions we arrive at a formula, once we have eliminated all occurrences of both S and B. So a formula must contain only symbols from the terminal alphabet Σ. As an example we produce some formulas using these productions.

The production of $((p \wedge q) \rightarrow \neg r)$:

S	$\Rightarrow$	(SBS)	by 3.5.
	$\Rightarrow$	$(S \rightarrow S)$	by 3.8.
	$\Rightarrow$	$((SBS) \rightarrow S)$	by 3.5 (first occurrence of S).
	$\Rightarrow$	$((S \wedge S) \rightarrow S)$	by 3.8.
	$\Rightarrow$	$((S \wedge S) \rightarrow \neg S)$	by 3.6 (last occurrence of S).
	$\Rightarrow$	$((p \wedge q) \rightarrow \neg r)$	using three applications of 3.7.

The production of $\neg((p \wedge q) \to (\neg p \vee \neg q))$:

$$\begin{aligned}
S &\Rightarrow \neg S && \text{by 3.6.}\\
&\Rightarrow \neg(SBS) && \text{by 3.5.}\\
&\Rightarrow \neg(S \to S) && \text{by 3.8.}\\
&\Rightarrow \neg((SBS) \to (SBS)) && \text{by 3.5 (both occurrences of } S\text{).}\\
&\Rightarrow \neg((S \wedge S) \to (S \vee S)) && \text{using two applications of 3.8.}\\
&\Rightarrow \neg((S \wedge S) \to (\neg S \vee \neg S)) && \text{using two applications of 3.6.}\\
&\Rightarrow \neg((p \wedge q) \to (\neg p \vee \neg q)) && \text{using four applications of 3.7.}
\end{aligned}$$

With different alphabets and a different set of productions, we can produce other formal languages. Here are some examples:

Examples

1. Let $\Sigma = \{0,1\}$. Let the language L be the set of all strings of the form m 0's followed by n 1's, where m and n are arbitrary non-negative integers. Put $N = \{S\}$. The set of productions P required to produce this language is

   $$S \Rightarrow 0S \mid S1 \mid \varepsilon.$$

   The first two productions allow the introduction of 0's at the beginning of a string and 1's at the end of a string, and the third eliminates S to produce a string of the required type.

2. Let $N = \{S,A,B,C\}$ and let $\Sigma = \{0,1\}$. Suppose that P consists of the productions

   $$S \Rightarrow AS \mid ACB, \quad A \Rightarrow 0, \quad B \Rightarrow 0, \quad C \Rightarrow 1.$$

   To find words in this language, we start with S, and use any of the productions that may be appropriate until we come to a word in the alphabet Σ.

   $S \Rightarrow AS \Rightarrow AACB \Rightarrow 0ACB \Rightarrow 00CB \Rightarrow 001B \Rightarrow 0010$,

   $S \Rightarrow AS \Rightarrow AAS \Rightarrow AAAS \Rightarrow AAAAS \Rightarrow AAAAACB \Rightarrow \cdots \Rightarrow 0000010$,

   $S \Rightarrow ACB \Rightarrow \cdots \Rightarrow 010$.

   The aim, in doing these productions is to end with a word in the terminal alphabet Σ. Having constructed several examples here it becomes fairly easy to give a description of the language produced by the productions.

3. Let $\Sigma = \{1\}$ be an alphabet, and let L be the subset of $\Sigma^\star$ consisting of all words with an even number of 1's. It is possible to find at least two systems of productions that will generate L.

   (i) Let $\Sigma = \{1\}$, $N = \{S\}$ and $P = \{S \Rightarrow 11S \mid \varepsilon\}$. Then $G = (N, \Sigma, S, P)$ is called a **phrase structure grammar** that generates L.

   (ii) Let $\Sigma = \{1\}$, $N = \{S, A, B\}$ and $P = \{S \Rightarrow \varepsilon \mid 1A,\ A \Rightarrow 1 \mid 1B,\ B \Rightarrow 1A\}$. Again $G = (N, \Sigma, S, P)$ is a phrase structure grammar that generates L.

Before giving more examples it is time to give the formal definition of a phrase structure grammars.

72 Definition. *A **phrase structure grammar** (or simply, **grammar**) G consists of*

1. *A finite set N of **non-terminal symbols**,*
2. *A finite set Σ of **terminal** symbols where $N \cap \Sigma = \emptyset$,*
3. *A **starting symbol** $S \in N$,*
4. *A finite set P of productions of the form $\alpha \Rightarrow \beta$, where $\alpha \in (N \cup \Sigma)^\star \setminus \Sigma^\star$ and $\beta \in (N \cup \Sigma)^\star$. (Note that $(N \cup \Sigma)^\star \setminus \Sigma^\star$ is the set of all strings from $N \cup \Sigma$ that contain at least one symbol from N.*

We write $G = (N, \Sigma, S, P)$.

The next example of a phrase structure grammar is more complicated in that the left-hand side of some of the productions consist of more than one symbol.

Examples (continued)

4. Let $\Sigma = \{0, 1, 2\}$, $N = \{S, A\}$ and $P = \{S \Rightarrow 1S2,\ S2 \Rightarrow 2A,\ 12A \Rightarrow 0\}$.

   To find the language generated by $G = (N, \Sigma, S, P)$, we must start with the production $S \Rightarrow 1S2$, because this is the only production starting with S. We can use this production as many times as we want. To eliminate S, we must use the

production $S2 \Rightarrow 2A$ and from here the only way to eliminate A is to use the production $12A \Rightarrow 0$. Using $S \Rightarrow 1S2$ repeatedly results in a string of 1's, followed by an S, followed by the same number of 2's as 1's.

$$S \Rightarrow 1S2 \Rightarrow 11S22 \Rightarrow 111S222 \Rightarrow \cdots \Rightarrow 1\ldots1S2\ldots2.$$

When we apply $S2 \Rightarrow 2A$ to this string we have a string of 1's followed by a 2, followed by A, followed by a (possibly empty) string of 2's. But the number of 1's is the same as the number of 2's.

$$1\ldots1S2\ldots2 \Rightarrow 1\ldots12A\ldots2.$$

When we remove the A, using $12A \Rightarrow 0$ we are left with a string of 1's (possibly empty) followed by a 0, followed by a string of 2's of the same length as the string of 1's.

$$1\ldots12A\ldots2 \Rightarrow 1\ldots0\ldots2.$$

Thus, some typical members of the language $L(G)$ are

$$0, 102, 11022, 1110222, 111102222, \ldots.$$

Phrase structure grammars can be classified according to the types of productions that are allowed. Noam Chomsky introduced a classification scheme, and the different types of languages described in this scheme correspond to the classes of languages that can be recognised by different models of computing machines.

73 Definition. *Let $G = (N, \Sigma, S, P)$ be a phrase structure grammar. Then*

- *G is a **type 0 grammar** if there are no restrictions on the members of P.*
- *G is a **type 1 grammar** if for each production, $\alpha \Rightarrow \beta$ in P, $length(\alpha) \leq length(\beta)$ or $\beta = \varepsilon$.*
- *G is a **type 2 grammar** if for each production, $\alpha \Rightarrow \beta$ in P, α is a single symbol from N.*
- *G is a **type 3 grammar** if for each production, $\alpha \Rightarrow \beta$ in P, α is a single symbol from N, and β has at most one non-terminal symbol, which must be at the extreme right. It is also allowed that $\beta = \varepsilon$.*

By extension, a type k language is a language generated by a type k grammar.
Type 0 and type 1 grammars are very difficult to analyse and are important primarily in linguistics. Type 2 grammars are called **context free grammars** since a non-terminal symbol that is the left side of a production can be replaced by a string whenever it occurs. Correspondingly, type 2 languages are called **context free languages**.
When a production is of the form $lwr \Rightarrow lvr$, the grammar is called **context-sensitive**, since w can be replaced by v only when it is surrounded by the strings l and r.
Type 3 grammars and languages are also called **regular grammars** and **regular languages** respectively. We shall study the relationship between regular grammars and finite state machines in the next two chapters.

Final Examples

5. Let $\Sigma = \{0,1\}$, $N = \{S\}$, and $P = \{S \Rightarrow 0S1 \mid \varepsilon\}$. This grammar is a context free or type 2 grammar and generates the set $\{0^n1^n : n = 0,1,2,\dots\}$.

6. Let $\Sigma = \{0,1\}$, $N = \{S,A\}$, and $P = \{S \Rightarrow 0S \mid 1A \mid \varepsilon, \quad A \Rightarrow 1A \mid \varepsilon\}$. This is a regular grammar and generates the (regular) language $\{0^m1^n : m,n = 0,1,2,\dots\}$.

   Note, however, that this regular language can also be generated by the non-regular (type 2 in fact) grammar given by $\Sigma' = \{0,1\}$, $N' = \{S\}$, and $P' = \{S \Rightarrow 0S \mid S1 \mid \varepsilon\}$. Thus $G' = (N', \Sigma', S, P')$ is a type 2 grammar generating a type 3 language, illustrating the general point that type k languages form a subclass of the class of type $k-1$ languages, $k = 1,2,3$.

END OF CHAPTER THREE

4. AUTOMATA AND THEIR LANGUAGES

4.1 Introduction to Automata

A **finite-state automaton** is an idealised machine or mathematical model that embodies the essential idea of a sequential circuit. Each piece of input into a finite state automaton leads to a change in the state of the machine, which affects how subsequent input is processed. For example, if you dial 1-8000 on a telephone, the telephone circuit is in a state of readiness to receive the final six digits of a toll free call. On the other hand, if you dial 6249 it is in a state of expectation for the final four digits of a local call. A vending machine operates in a similar way. The most important finite state automata in modern life are digital computers. The internal memory of a computer is a system that can exist in only a finite (very large) number of different states. There are several different types of finite state machines, but our study is limited to those that do not produce output. They have **final** or **accepting** states, and they recognise strings that take the starting state to a final state. The set of strings that is recognised by a finite state machine is known as the language accepted by that machine. We will be interested in the properties of these languages, and their relationship to the languages we have already studied.

74 Definition. *A **deterministic finite-state automaton** A is specified by the following:*

1. *A finite set of symbols I called the **input alphabet**.*
2. *A finite set S of **states**.*
3. *An **initial state** $s_0 \in S$.*
4. *A subset of F of S called the set of **final** or **accepting states**, and*
5. *A **transition** or **next-state function** $N : S \times I \to S$ that assigns a next state to every pair of state and input.*

We write $A = (I, S, s_0, F, N)$.

In this course all automata will be finite-state, so we can safely drop that qualification. In this *chapter* all automata will be deterministic, so we can also drop that qualification here. So no confusion should arise from abbreviating "deterministic finite-state automatoma" to just 'automaton'.

The operation of a finite state automaton is often described by a directed graph or **transition diagram**. The states of the machine are represented by vertices that are usually drawn as circles, and the inputs label edges. There is a directed edge m from state s_i to s_j if $N(s_i, m) = s_j$. We will label the initial state $\ominus$, and the accepting states by $\oplus$. If the initial state is also accepting it will be labelled $\pm$ or $\mp$.

Examples

1. Let $I = \{a, b\}$, $S = \{s_0, s_1, s_2, s_3\}$ and let the set F of accepting states be $\{s_2, s_3\}$. (The designated initial state is s_0.) Define $N : S \times I \to S$ by

$$\begin{array}{llll} N(s_0,a) = s_2, & N(s_1,a) = s_2, & N(s_2,a) = s_3, & N(s_3,a) = s_0, \\ N(s_0,b) = s_1, & N(s_1,b) = s_2, & N(s_2,b) = s_1, & N(s_3,b) = s_3. \end{array}$$

   The transition diagram has four vertices corresponding to each of the four states. There are two edges, one labelled a and one labelled b, from each vertex. Altogether there are eight edges. The transition diagram is shown in Figure 4.1.

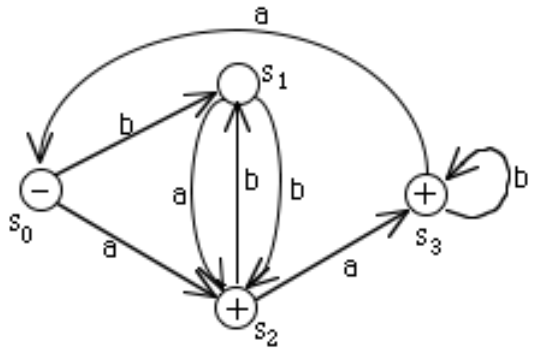
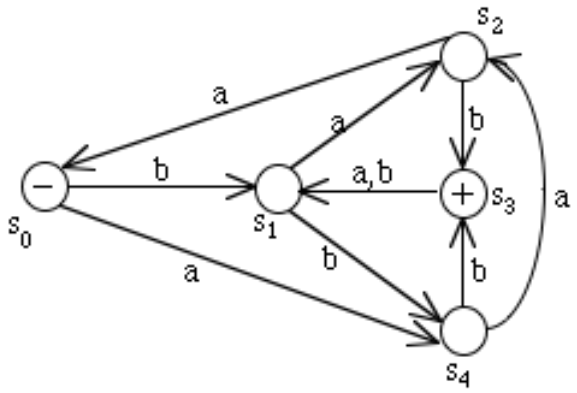


Figure 4.1: Transition diagram for Example 1

2. A table, called the **state table**, gives another way of describing an automaton. The state table is a matrix whose rows are labelled by the states, where the first row has the initial state, and accepting states are marked with a plus. The columns are labelled by the inputs. The entry in row s_i and column m is $N(s_i, m)$. Since the state table specifies the states, the inputs, the transition function and the initial and final states of a finite state machine, it gives a complete description of an automaton. Consider for example the finite automaton with transition diagram Figure 4.2.

   First note that $I = \{a, b\}$, $S = \{s_0, s_1.s_2, s_3, s_4\}$ and $F = \{s_3\}$. We construct a table whose rows are labelled s_0, s_1, s_2, s_3 and s_4, with s_0 marked as initial and s_3 as final. The columns are labelled a and b. In row s_i the entries are $N(s_i, a)$ in the column labelled a and $N(s_i, b)$ in the column labelled b, for all $i = 0, 1, 2, 3, 4$.



		a	b
$\ominus$	s_0	s_4	s_1
	s_1	s_2	s_4
	s_2	s_0	s_3
$\oplus$	s_3	s_1	s_1
	s_4	s_2	s_3

Figure 4.2: Example 2: Transition diagram and corresponding state table.

One of the questions that now arises is the question of which strings of the input alphabet are **accepted** by an automaton. If we consider the automaton with transition diagram given in Figure 4.2 we see that if the string ab is input into the machine, the machine first goes to state s_4, and then to state s_3. Note that the automaton works by scanning the string from left to right, starting at the first letter of the string. When it is in any state, and viewing a given letter, the machine changes its state, depending only on the letter it is viewing and its current state. When there are no more letters to view, the machine stops. In the situation we have just looked at the machine has stopped at state s_3. Since this is an accepting state for this automaton we say that the string ab is accepted by this automaton. In a similar way we can say that the string ba is not accepted by the automaton, because when the string ba is input the machine goes to state s_2, which is not an accepting state.

We can then ask if there is any way of describing all the strings that are accepted by a given automaton. If we look again at the automaton above we see that the strings $ab, bab, bbb, abaab, abbab, abbbb, baabbb, aab, \ldots$ are all accepted by this machine. This **language** is not so easy to describe. We would notice that all the strings we have written end with the input b, and we have to have a "correct" number of b's and a "correct" pattern. We can also see that there are strings that are not accepted by this machine, namely, the strings $a, b, bb, aa, aaab, aba$ and many more.

75 Definition. *If A is an automaton, the **language** $L(A)$ is the set of all strings that are accepted by A.*

Examples (continued)

3. We determine the language accepted by the finite state automata A_1, A_2 and A_3 shown in Figure 4.3.

   In the automaton A_1, recall that the symbol ⊕ represents a state that is both the initial state and an accepting or final state. The only strings that take this state to itself are those consisting of zero or more consecutive b's. Since there are no other accepting states it follows that $L(A_1) = \{b^n : n = 0, 1, 2, \ldots\}$.

   For A_2 there is also only one accepting state. If the initial input is b, it goes directly to the accepting state. If the input is ab the machine also goes to the accepting state. These are the only strings that are accepted. Hence $L(A_1) = \{b, ab\}$.

   For A_3 there are two accepting states, the initial state and one other. The only strings that take the initial state to itself are $\varepsilon, a, aa, aaa, \ldots$. The only other

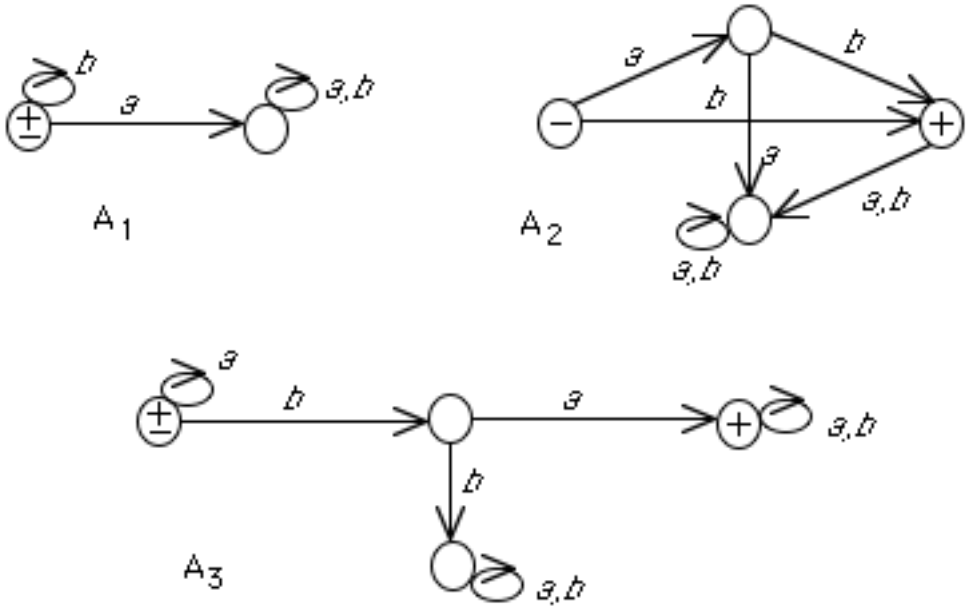


Figure 4.3: Automata A_1, A_2, A_3.

strings that will take A_3 to an accepting state are strings with 0 or more consecutive a's followed by ba, followed by any (or no) string. Hence, $L(A_3) = \{a^n,\ a^n bax : n = 0,1,2,\ldots$ and x is any string on $\{a,b\}\}$.

4.2 Designing automata

We consider the problem of designing an automaton that accepts a given language and only that language. So we start with a description of a language and design a machine that accepts exactly that language, when that is possible.
It is hard or almost impossible to give rules for designing an automaton. Later when we have a concise way of writing a language we will be able to give a process through which we can make an automaton that has a given language. But our approach at this stage is much less formal. Although there are no set rules, there are some questions we can ask that help in the process of designing an automaton. Given the description of a certain language, they are:

- What is the shortest string this automaton can accept?
- How many states seem to be necessary so that the automaton can accept this shortest string?

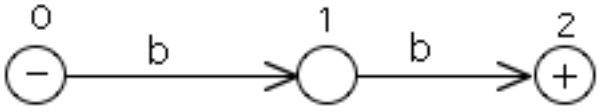
- What are some other strings that can be accepted by the automaton?
- What are some strings that we don't want to be accepted by the automaton?
- Do we need a sink? A *sink* is a state s from which there is no escape; $N(s,x) = s$ for every $x \in I$. We need a **non-accepting sink** (respectively **accepting sink**) when there are conditions in the language that imply the existence of one or more strings w for which no (respectively every) string starting with w is accepted. In Example 3 of the last section Automata A_1 and A_2 have non-accepting sinks, while A_3 has both kinds.

Examples

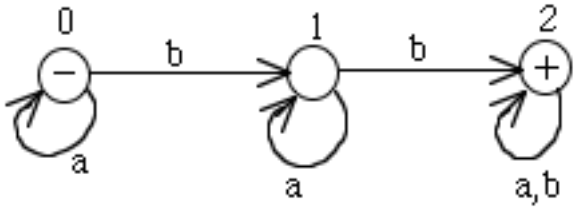
In each of the the three examples below, $I = \{a,b\}$.

1. We construct an automaton whose language consists of all strings containing two or more b's.

   The shortest string accepted by an automaton with this language is bb, so we need at least three states: one for the beginning, one that will indicate that the string has one b so far, and another to indicate that a string has two b's.



   To complete the diagram we need edges that indicate what happens when symbol a is input at states 0 and 1, and what happens when a or b is input at state 2.



   At state 0, if an a is input before any b's, the machine will just stay at state 0, and if two b's follow later the string ca be accepted. The situation is similar at state 1. If an a is input after the machine has got to state 1, it will just stay there, essentially waiting for another b to get to the accepting state.

   If what has been input so far gets the machine to state 2, it doesn't matter what comes afterwards, the string will be accepted. So the machine stays in the accepting state even if more a's or b's are input. State 2 is an accepting sink.

Note that state 0 means 'I'm waiting for the first b', state 1 means 'I'm waiting for the second b", and state 2 means 'I've seen at least two bs'.

We could explicitly give the next-state function from this diagram, but it seems unnecessary to do so since it is more easily seen from the labelled transition diagram of the automaton.

2. Let L be the language consisting of strings with an even number of a's. We find an automaton A with $L(\boldsymbol{A}) = L$.

   In this case the shortest string that is accepted is the empty string, ε. This means that the initial state will also be an accepting state. Apart from that we need only one more state. We can think of the two states as "Even" and "Odd". This quickly leads to Figure 4.4. We need to be careful to include the loops around the vertices, because each vertex must have an edge leaving it for each of the letters of the input alphabet.

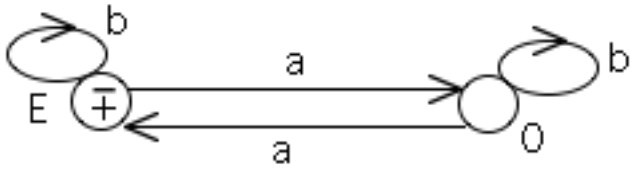


Figure 4.4: Automaton to accept all and only stings with an even number of a's.

3. We find an automaton whose language consists of just two strings: ab and ba.

   The automaton of Figure 4.5 clearly works. Let's examine its construction.

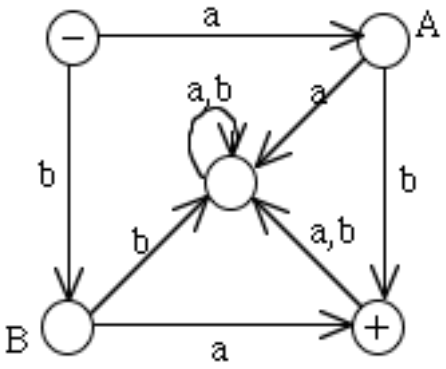


Figure 4.5: An automaton for $L = \{ab, ba\}$

Note that the centre state is a non-accepting sink. Once there, the string must necessarily be rejected. A good technique for constructing diagrams of automata is to regard each vertex as representing a set of strings with appropriate properties. Thus, ⊖ represents the empty string ε as well as the initial state. In Figure 4.5 the vertices labelled "A" and "B" represent the strings a and b respectively. The vertex ⊕ represents the required set, and the rejected set of strings with length ≥ 2 is the sink noted earlier.

4.3 Simplifying automata

It can sometimes be the case that two automata actually accept the same language, even though one may be considerably more complicated than the other. Or we may be working with an automaton which is large and complicated and it seems clear that there should be a simplified version that does the same thing, that is, accepts the same language. In this section we look at a way of simplifying an automaton in the sense of finding an automaton with fewer states that accepts the same language. There are many advantages to having the simplest possible machine. The smaller the automaton the less memory space it needs and the easier it is to write a computer algorithm based on it.

Consider the finite-state automata $\boldsymbol{A}$ and $\boldsymbol{A}'$ in Figure 4.6.

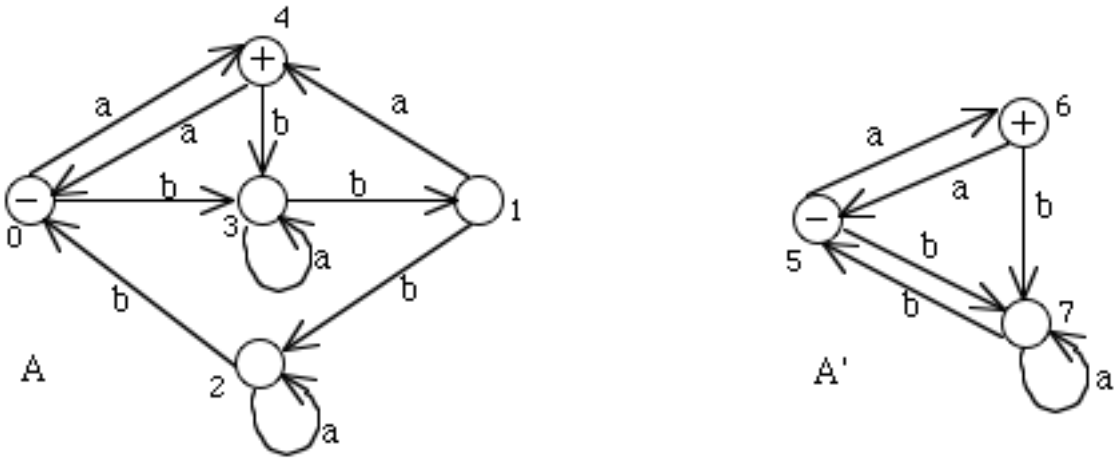


Figure 4.6: Are these automata equivalent?

If you look closely at these machines you will discover that they both accept the same strings, that is they accept the same language, although $\boldsymbol{A}$ is more complicated and has

more states than $\boldsymbol{A}'$. These automata are "equivalent". The reason for the equivalence of these automata is that some of the states of $\boldsymbol{A}$ can be combined without affecting the acceptance or non-acceptance of any input strings. It turns out that state 0 can be combined with state 1 and state 2 can be combined with state 3. The automaton with the combined states $\{0, 1\}$ and $\{2, 3\}$ and the state $\{4\}$ is called the **quotient automaton** of $\boldsymbol{A}$ and is denoted $\bar{\boldsymbol{A}}$. Its transition diagram is obtained by combining the vertices for 0 and 1 and for 2 and 3. In $\boldsymbol{A}'$ we have state 5 = $\{0,1\}$, 6 = $\{4\}$, 7 = $\{2,3\}$. Then, since there is an arrow labelled b from state 1 to state 2 in $\boldsymbol{A}$, there is an arrow labelled b from state 5 to state 7 in $\boldsymbol{A}'$, and so on.

In general, simplification of an automaton involves identifying "equivalent states" that can be combined without affecting the action of the automaton on input strings. Mathematically speaking, this means defining an equivalence relation on the set of states of the automaton and forming a new automaton whose states are the equivalence class of the relation.

In defining this equivalence we need a definition. Suppose an automaton is in one of its states and a string w is input into the machine. To what state will the machine eventually go? The function that defines this outcome is called the **eventual state function.**

76 Definition. *Let $\boldsymbol{A} = \{I, S, s_0, F, N\}$ be a (deterministic finite-state) automaton. Let $I^\star$ be the set of all strings over the alphabet I, and define the **eventual state function** $N^\star : S \times I^\star \to S$ as follows:*
For any state s and any input string w,
$N^\star(s,w)$ = the state that $\boldsymbol{A}$ goes if the symbols of w are input to $\boldsymbol{A}$ in sequence, starting when $\boldsymbol{A}$ is in state s.

From here we define what we mean when we say that two states are $\star$-**equivalent**.

77 Definition. *Let $\boldsymbol{A}$ be a (deterministic finite-state) automaton. Given any states s and t of $\boldsymbol{A}$, we say that s and t are $\star$-**equivalent** if and only if, for all input strings w either $N^\star(s,w)$ and $N^\star(t,w)$ are both accepting states or both are non-accepting states.*

In other words, states s and t are $\star$-equivalent, if and only if, for all input strings w

$$N^\star(s,w) \text{ is an accepting state} \Leftrightarrow N^\star(t,w) \text{ is an accepting state.}$$

From this definition it is clear that a state s is $\star$-equivalent to itself. Also if state s is $\star$-equivalent to state t, then state t is $\star$-equivalent to state s. Further if state s is $\star$-equivalent to state t, and state t is $\star$-equivalent to state u, then state s is $\star$-equivalent

to state u. This means that in a mathematical sense $\star$-equivalence is an equivalence relation on s, the set of states of $\boldsymbol{A}$. However, the definition of $\star$-equivalence is a very difficult one to use directly. The condition needs to be checked for **all** input strings w, that is for all words that can be made from the input alphabet I. This is usually an infinite set. Therefore we need some alternative procedure for determining when two states s and t are $\star$-equivalent. To do this we use an iterative procedure based on a simpler kind of equivalence called ***k*-equivalence**.

78 Definition. *Let **A** be a (deterministic finite-state) automaton. Given any states s and t, and an integer $k \geq 0$, we say that s is **k-equivalent** to t if, and only if, for all strings w of length less than or equal to k, either $N^\star(s,w))$ and $N^\star(t,w)$ are both accepting states or are both non accepting states.*

Let k be any integer. Some useful facts about k-equivalence follow from the definition:

- For $k \geq 0$, k-equivalence is an equivalence relation.
- For $k \geq 0$, the k-equivalence classes partition the set of all states of the automaton into a union of mutually disjoint subsets.
- For $k \geq 1$, if two states are k-equivalent, then they are also $(k-1)$-equivalent.
- For $k \geq 1$, each k-equivalence class is a subset of a $(k-1)$-equivalence class.
- Any two states that are k-equivalent for all integers $k \geq 0$ are $\star$-equivalent.

The next theorem gives a recursive description of k-equivalence states.

79 Theorem. *Let **A** be a (deterministic finite-state) automaton with next-state (transition) function N. Let s and t be any states in **A**. Then*

1. *s is 0-equivalent to t* $\Leftrightarrow$ *either s and t are both accepting states or are both non-accepting states*

2. *for every integer $k \geq 1$, s is k-equivalent to t* $\Leftrightarrow$ *s and t are $(k-1)$-equivalent, and for any input symbol m, $N(s,m)$ and $N(t,m)$ are also $(k-1)$-equivalent.*

The validity of this theorem follows from the fact that inputting a string w of length k has the same effect as inputting the first symbol of w and then the remaining $k-1$ symbols of w. The theorem implies that if you know which states are $(k-1)$-equivalent and if you know the action of the transition function, then you can decide which states

are k-equivalent. This means that the k-equivalence classes are obtained by subdividing the $(k-1)$-equivalence classes according to the action of the transition function on the members of the classes.

Finding k-equivalence classes

We find the 0-equivalence classes, the 1-equivalence classes, and the 2-equivalence classes of the automaton shown in Figure 4.7 below.

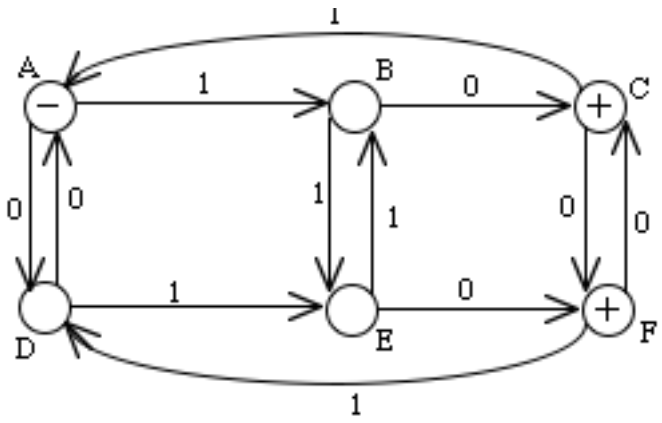


Figure 4.7: An automaton to be simplified

0-equivalence classes: By theorem 79, two states are 0-equivalent if and only if they are both accepting states or both non-accepting states. Thus the two sets of 0-equivalence classes are

$$\{A,B,D,E\} \qquad \text{and} \qquad \{C,F\}$$

1-equivalence classes: It is helpful at this stage to make Table 4.1 that shows the action of the next-state function:

Table 4.1: Action of N on 0-classes of automaton of Figure 4.7

	A	B	D	E	C	F
0	D	C	A	F	F	C
1	B	E	E	B	A	D

We use Theorem 79 again to decide which states are 1-equivalent. We consider first the 0-equivalence class $\{A,B,D,E\}$, since the 1-equivalence classes will be subsets of the 0-equivalence classes.

- We can immediately see that states A and B are not 1-equivalent, since $N(A,0) = D$ and $N(B,0) = C$ and D and C are not 0-equivalent.
- In a similar way, A and E are not 1-equivalent, since $N(A,0) = D$ and $N(E,0) = F$ and D and F are not 0-equivalent.
- However, A and D are 1-equivalent since:
  (i) $N(A,0) = D$ and $N(D,0) = A$ and D and A are 0-equivalent and
  (ii) $N(A,1) = B$ and $N(D,1) = E$ and B and E are 0-equivalent.
- In a similar way B and E are 1-equivalent since:
  (i) $N(B,0) = C$ and $N(E,0) = F$, and C and F are 0-equivalent and
  (ii) $N(B,1) = E$ and $N(E,1) = B$ and E and B are 0-equivalent.

A similar analysis of the 0-equivalence class $\{C,F\}$ shows that it is also a 1-equivalence class. Hence the 1-equivalence classes are

$$\{A,D\}, \quad \{B,E\} \quad \text{and} \quad \{C,F\}.$$

2-equivalence classes: By Theorem 79 two states are 2-equivalent if they are 1-equivalent, and after input of any of the input symbols, their next states are 1-equivalent. So it is convenient to rearrange Table 4.1 into 1-classes (Table 4.2):

Table 4.2: Action of N on 1-classes of automaton of Figure 4.7

	A	D	B	E	C	F
0	D	A	C	F	F	C
1	B	E	E	B	A	D

With the help of the new table we conclude that A and D are 2-equivalent since they are 1-equivalent and:
(i) $N(A,0) = D$ and $N(D,0) = A$, and D and A are 1-equivalent, and
(ii) $N(A,1) = B$ and $N(D,1) = E$ and B and E are 1-equivalent.

Similarly, B is 2-equivalent to E, and C is 2-equivalent to F. Hence the 2-equivalence classes are

$$\{A,D\}, \quad \{B,E\} \quad \text{and} \quad \{C,F\}.$$

In other words, the 2-equivalence classes are the same as the 1-equivalence classes.

Finding the $\star$-equivalence classes

The aim of the work we are doing is to find the $\star$-equivalence classes and we only find the k-equivalence classes in order that we might find the $\star$-equivalence classes. This is because to simplify an automaton we need the $\star$-equivalence classes. The next theorem shows the relationship between the $\star$-equivalence classes and the k -equivalence classes.

80 Theorem. *Let $\boldsymbol{A}$ be a (deterministic finite-state) automaton. Then there is an integer $K \geq 0$ such that the set of K-equivalence classes of states of $\boldsymbol{A}$ is equal to the set of $(K+1)$-equivalence classes of states of $\boldsymbol{A}$. For all such K these sets of equivalence classes are equal to the $\star$-equivalence classes.*

Let us consider again the automaton of Figure 4.7. We showed that the 1-equivalence classes are the same as the 2-equivalence classes. Since the process of finding the 3-equivalence classes from the 2-equivalence classes is exactly the same as finding the 2-equivalence classes from the 1-equivalence classes, we can see that the 3-equivalence classes will be the same as the 2-equivalence classes. By an inductive process we can use this to see that for any integer $k \geq 2$ the k-equivalence classes are the same as the 1-equivalence classes. This means the if s and t are 1-equivalent, and w is an input string of any length, then $N^\star(s,w)$ and $N^\star(t,w)$ will either both be accepting states or both will be non-accepting states. So for the automaton of Figure 4.7, the $\star$-equivalence classes are

$$\{A,D\}, \quad \{B,E\} \quad \text{and} \quad \{C,F\}.$$

We usually use the notation:

$$[A] = \{A,D\} = [D], \quad [B] = \{B,E\} = [E], \quad \text{and} \quad [C] = \{C,F\} = [F].$$

Here, we read "$[A]$" as "the equivalence class containing A".

The quotient automaton

The preceding work shows us that to find the $\star$-equivalence classes we find the 0-equivalence classes, the 1-equivalence classes, the 2-equivalence classes ... until there is no change going from the k-equivalence classes to the $(k+1)$-equivalence classes. At that stage the equivalence classes we have found will be the $\star$-equivalence classes. We can now use the $\star$-equivalence classes to define the *quotient automaton* $\overline{\boldsymbol{A}}$ of an automaton $\boldsymbol{A}$. Two facts are important in the construction of the quotient automaton.

No $\star$-equivalence class of states of $\boldsymbol{A}$ can contain both accepting states and non-accepting states.

This is because the 0-equivalence classes divides the set of states of $\boldsymbol{A}$ into accepting and non-accepting states and the $\star$-equivalence classes are subsets of the 0-equivalence classes.

If two states are $\star$-equivalent, then their next states are also $\star$-equivalent for any input symbol m.

The proof of this statement is more difficult, and it is a good exercise. The statement itself is important when we want to define the quotient automaton. Recalling the definition of an automaton, to define a new automaton we need to describe the states of the new machine, the input alphabet, the initial state, the accepting states and the next-state (transition) function. We do this as follows.

81 Definition. *Let $\boldsymbol{A}$ be a (deterministic finite-state) automaton with set of states S, set of input symbols I, and transition function N. The **quotient automaton** $\overline{\boldsymbol{A}}$ is defined as follows:*

1. *The set of states, $\overline{S}$, of $\overline{\boldsymbol{A}}$ is the set of $\star$-equivalence classes of $\boldsymbol{A}$;*
2. *The set of input symbols, $\overline{I}$, of $\overline{\boldsymbol{A}}$ is the same as I ;*
3. *The initial state of $\overline{\boldsymbol{A}}$ is $[s_0]$, the equivalence class containing s_0, where s_0 is the initial state of $\boldsymbol{A}$;*
4. *The accepting states of $\overline{\boldsymbol{A}}$ are the states of the form $[s]$, where s is an accepting state of $\boldsymbol{A}$;*
5. *The transition function $\overline{N} : \overline{S} \times I \to \overline{S}$ is defined as follows:*

   for all states $[s]$ in $\overline{S}$ and input symbols m in I , $\overline{N}([s], m) = [N(s, m)]$.

   (That is, if m is input into $\overline{\boldsymbol{A}}$ when $\overline{\boldsymbol{A}}$ is in state $[s]$ then $\overline{\boldsymbol{A}}$ goes to the state that is the $\star$-equivalence class of $N(s, m)$.)

Since the states of $\overline{\boldsymbol{A}}$ are sets of the states of $\boldsymbol{A}$, there are usually fewer states in $\overline{\boldsymbol{A}}$ than in $\boldsymbol{A}$ and in this sense we can say that $\overline{\boldsymbol{A}}$ is simpler than $\boldsymbol{A}$. However, by the construction the quotient automaton $\overline{\boldsymbol{A}}$ accepts exactly the same strings as $\boldsymbol{A}$.

82 Theorem. *Let $\boldsymbol{A}$ be a (deterministic finite-state) automaton. Then the quotient automaton $\overline{\boldsymbol{A}}$ accepts exactly the same language as $\boldsymbol{A}$. If $L(\boldsymbol{A})$ denotes the language accepted by $\boldsymbol{A}$, and $L(\overline{\boldsymbol{A}})$ the language accepted by $\overline{\boldsymbol{A}}$, then*

$$L(\boldsymbol{A}) = L(\overline{\boldsymbol{A}}).$$

In some cases we may find that the quotient automaton is exactly the same as the original automaton. This would occur when all the $\star$-equivalence classes consist of a single state. In this case the automaton is as simple as it can be.

Constructing the Quotient Automaton

Let $\boldsymbol{A}$ be an automaton with set of states S and next-state function N. From the preceding discussion and theorems we can now give the recipe for constructing the quotient automaton $\overline{\boldsymbol{A}}$ of $\boldsymbol{A}$.

1. Find the set of 0-equivalence classes of S.

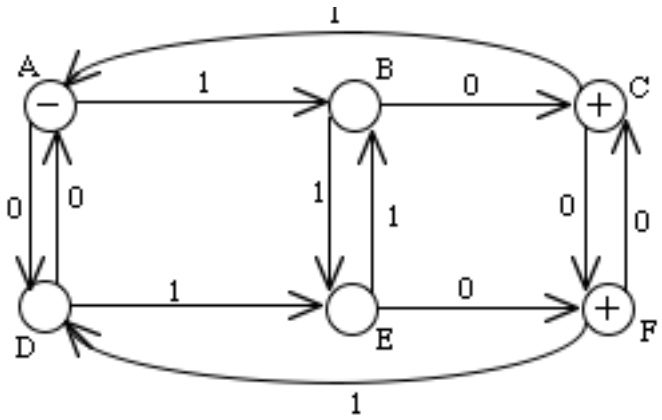
2. For each integer $k \geq 1$, subdivide the $(k-1)$-equivalence classes to find the k-equivalence classes of S. Stop dividing when, for some integer K, the set of $(K+1)$-equivalence classes is exactly the same as the set of K-equivalence classes. At this point we the set of K-equivalence classes is the same as the set of $\star$-equivalence classes.

3. Construct the quotient automaton $\overline{\boldsymbol{A}}$. The states of $\overline{\boldsymbol{A}}$ are the $\star$-equivalence classes of states of $\boldsymbol{A}$, and the transition function $\overline{N}$ is given by

$$\overline{N}([s],m) = [N(s,m)]$$

   for any state of $\overline{\boldsymbol{A}}$ and any input symbol m.

Let us apply the recipe to the automaton $\boldsymbol{A}$ of Figure 4.7, shown again below for convenience.



Method. We already know that the $\star$-equivalence classes of the states of $\boldsymbol{A}$ are

$$[A] = \{A, D\} = [D], \quad [B] = \{B, E\} = [E], \quad \text{and} \quad [C] = \{C, F\} = [F].$$

These then become the states of the quotient automaton $\overline{\boldsymbol{A}}$. The accepting states of $\boldsymbol{A}$ are C and F, so the accepting state of $\overline{\boldsymbol{A}}$ is $[C] = [F]$. The initial state of $\boldsymbol{A}$ is A, so the initial state of $\overline{\boldsymbol{A}}$ is $[A] = [D]$. The transitions function $\overline{N}$ for $\overline{\boldsymbol{A}}$ is given as follows:

$$\begin{aligned}
\overline{N}([A], 0) &= [N(A, 0)] = [D] = [A], \\
\overline{N}([A], 1) &= [N(A, 1)] = [B] = [E], \\
\\
\overline{N}([B], 0) &= [N(B, 0)] = [C] = [F], \\
\overline{N}([B], 1) &= [N(A, 1)] = [E] = [B], \\
\\
\overline{N}([C], 0) &= [N(C, 0)] = [F] = [C], \\
\overline{N}([C], 1) &= [N(C, 1)] = [A] = [D],
\end{aligned}$$

The transition diagram for $\overline{\boldsymbol{A}}$ is shown below in Figure 4.8.

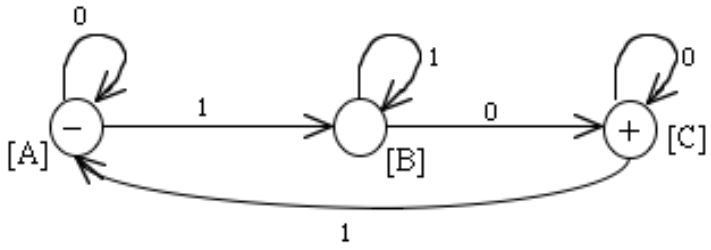


Figure 4.8: The quotient automaton.

This new automaton accepts the same language as the original.

Question Can you describe this language?

Equivalent Automata

Two automata are called *equivalent* if every input string leads to the same accept/non-accept result in each automaton. This means that two automata are equivalent if they accept the same language.

83 Definition. *Let A and A' be (deterministic finite-state) automata with the same set of input symbols I. Let $L(A)$ denote the language accepted by A , and $L(A')$ be the language accepted by A'. Then A is said to be equivalent to A' if, and only if, $L(A) = L(A')$.*

We can use the process of simplifying automata to decide whether or not two automata are equivalent.

Example

Show that the two automata in Figure 4.9 are equivalent.

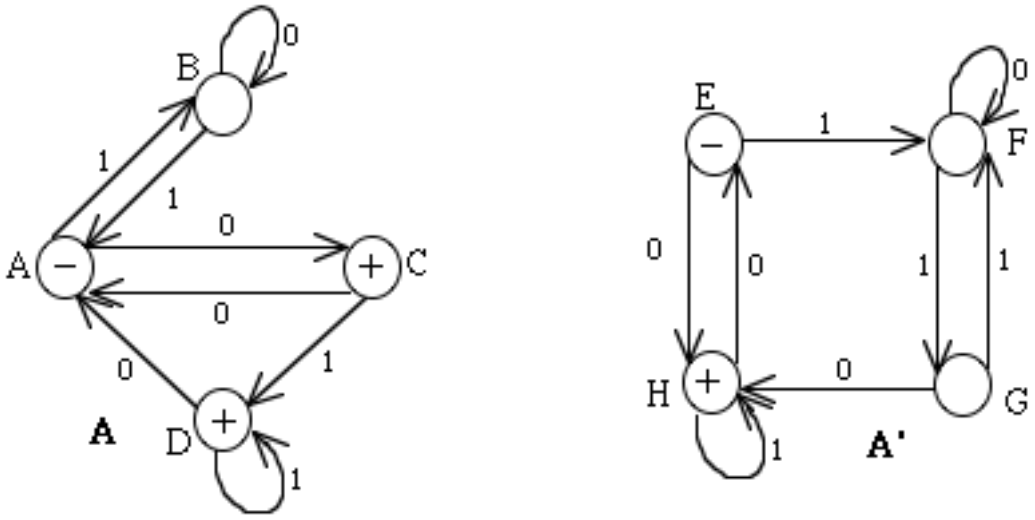


Figure 4.9: Are these equivalent?

Method **For the automaton A** : *The 0-equivalence classes* are $\{A,B\}$, (the non-accepting states) and $\{D,C\}$ (the accepting states).

	A	B	C	D
0	C	B	A	A
1	B	A	D	D

The 1-equivalence classes : The 1-equivalence classes will be subsets of the 0-equivalence classes. We can immediately see that A and B are not 1-equivalent, since when 0 is input into A, the machine goes to state C , and when 0 is input into state B , the machine stays in state B , and B and C are not 0-equivalent. This means that A and B will each form an equivalence class that contains only one element. Looking at the other

0-equivalence class, we see that C and D are 1-equivalent since they go to exactly the same states when both 0 and 1 are input. Hence the 1-equivalence classes are

$$\{A\}, \quad \{B\}, \quad \text{and} \quad \{C,D\}.$$

The 2-equivalence classes are the same as the 1-equivalence classes and hence the $\star$-equivalence classes are

$$[A] = \{A\}, \quad [B] = \{B\}, \quad \text{and} \quad [C] = \{C,D\}.$$

For the automaton A':
The 0-equivalence classes are $\{E,F,G\}$, and $\{F\}$.

	E	F	G	H
0	H	F	H	E
1	F	G	F	H

The 1-equivalence classes are $\{E,G\}$, $\{F\}$, and $\{H\}$.

	E	G	F	H
0	H	H	F	E
1	F	F	G	H

The *2-equivalence classes* are the same as the 1-equivalence classes and hence the $\star$-equivalence classes are

$$[E] = \{E,G\} = [F], \quad [F] = \{F\}, \quad \text{and} \quad [H] = \{H\}.$$

We now calculate the next-state function for both $\boldsymbol{A}$ and $\boldsymbol{A'}$.

	$[A]$	$[B]$	$[C]$		$[E]$	$[F]$	$[H]$
0	$[C]$	$[B]$	$[A]$		$[H]$	$[F]$	$[E]$
1	$[B]$	$[A]$	$[C]$		$[F]$	$[E]$	$[H]$

The initial state for $\boldsymbol{A}$ is $[A]$ and the initial state for $\boldsymbol{A'}$ is $[E]$. The accepting state for $\boldsymbol{A}$ is $[C]$, and the accepting state for $\boldsymbol{A'}$ is $[H]$. We now have all the information we need and can draw the transition diagrams for the quotient automata $\overline{\boldsymbol{A}}$ and $\overline{\boldsymbol{A'}}$ (Figure 4.10).

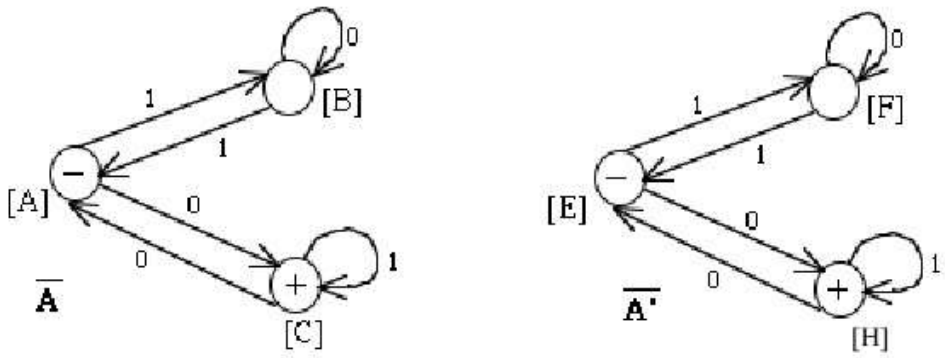


Figure 4.10: Quotient Automata for A and A'.

It is clear from these diagrams that, except for the labelling, $\overline{A}$ and $\overline{A'}$ are the same and accept the same language. Since each original automaton accepts the same language as its quotient automaton, this means that A and A' accept the same language, and hence are equivalent.

This method can also be used to help us decide that two automata are not equivalent. However, it is sometimes easier to find a string that is accepted by one machine and not the other, if we suspect they are not equivalent.

END OF CHAPTER FOUR

5. REGULAR SETS AND AUTOMATA

We have seen that finite state automata can be used as language recognisers. In this section we aim to determine the language recognised by a finite state automaton. This may seem to be quite a difficult problem, but we find there is a simple characterisation of the sets that can be recognised by finite state automata. An American mathematician, Stephen Kleene, first solved this problem in 1956. He showed that there is a finite state automaton that recognises a set if and only if the set is a **regular set**.
We defined regular grammars (or type-3 grammars) in Section 3.4. It will not be surprising then, that there is a connection between regular sets, which are recognised by finite state automata, and regular grammars. A set is regular if and only if it is generated by a regular grammar. There are also sets, which cannot be recognised by any finite state automata. We give some examples.

5.1 Regular Sets

Regular sets can be built up from the null set, the empty string, and singleton strings using operations called concatenation, union, and Kleene closure in arbitrary order.

84 Definition. *Let Σ be an alphabet. We define three basic operations on sets of strings (subsets of $\Sigma^\star$)*

1. ***Union**. If A and B are subsets of $\Sigma^\star$, then $A+B$ denotes the union $A \cup B$.*
2. ***Concatenation**. If A and B are subsets of $\Sigma^\star$, then $A.B$, or simply AB denotes the set of all concatenations of a string in A with a string in B. Thus*

$$AB = \bigcup_{\alpha \in A, \beta \in B} \{\alpha\beta\}$$

3. ***Kleene Closure.** Let A be a subset of $\Sigma^\star$. The Kleene closure, $A^\star$, consists of all possible concatenations $\alpha_1\alpha_2\ldots\alpha_k$, where k is an arbitrary natural number and all $\alpha_i \in A$. The empty string, ε, is included in $A^\star$, and corresponds to $k = 0$.*

Examples

Following standard practice, in the examples below and subsequently, we abbreviate a singleton set $\{s\}$ to just s.
Let $\Sigma = \{a,b\}$. Then

(a) $a+b = \{a\} \cup \{b\} = \{a,b\}$. So $a+b$ is the set consisting of a and b.

(b) $(bb)^\star + a^\star$ is the set of strings that have an even number of b's, and no a's, or any number of a's and no b's, or the empty string.

(c) $a^\star ba^\star ba^\star$ is the set of strings containing exactly two b's.

(d) $ab^\star$ is the set of strings beginning with a single a and followed by any number of b's. So $ab^\star = \{a, ab, abb, abbb, \ldots\}$.

(e) $a^\star b + b^\star a + (aa+bb)^\star$ is the set of all strings beginning with any number of a's and ending with a single b, and all strings beginning with any number of b's and ending with a single a, and all strings consisting of consecutive pairs of the form aa or bb.

(f) $(a+b)^\star$ is the set of all strings on Σ using only the letters a and b.

(g) $a(a+b)^\star$ is the set of all strings beginning with a.

(h) $(ab+ba+aa+bb)^\star$ is the set of all strings of a's and b's of even length.

85 Definition. *A **regular set** on an alphabet Σ is the set of strings that can be finitely expressed by using only the individual letters of Σ, the empty string ε, and the operators $+$, $.$ and *. The null set $\emptyset$ is also defined as a regular set.*

The above examples are all regular sets.

Alternative Definition of Regular Set

It is sometimes more convenient to give a recursive definition of a **regular set** and we do this as follows:

1. $\emptyset$ is a regular set; $\{\varepsilon\}$ is a regular set.
2. If $x \in \Sigma$, then $\{x\}$ is regular.
3. A is regular if and only if it is of type 1 or 2, or of the form:
   (a) $A = B + C$, where B and C are regular, or
   (b) $A = BC$, where B and C are regular, or
   (c) $A = B^\star$, where B is regular.

Some Identities for Regular Expressions

Let R, S, T and V be any regular sets of strings. The following identities hold.

$$(ST)^\star S = S(TS)^\star$$

$$R(S+T) = RS + RT$$

$$(S+T)R = SR + TR$$

$$(S+T)^\star = (S^\star T^\star)^\star$$

$$(S+T)^\star = S^\star + (S^\star T S^\star)^\star$$

$$(S+T)^\star = S^\star (TS^\star)^\star$$

$$(S+T)^\star = T^\star (ST^\star)^\star$$

$$(S+T)^\star = (T^\star S)^\star T^\star$$

$$(S+T)^\star = (S^\star T)^\star S^\star$$

$$(R+S)(T+V) = RT + RV + ST + SV$$

$$R + RSS^\star = RS^\star$$

5.2 Nondeterministic Finite State Machines

We want to show that regular sets are precisely **those sets that are the language of some automaton**. However, before we do this we introduce two generalisations of an automaton. Recall that we have been using the single word 'automaton', as an abbreviation for **deterministic finite-state automaton** (DFA). The generalisations we consider are **nondeterministic finite-state automata** (NFA's). We make the definitions in terms of directed graphs and use the symbols $\ominus$ for an initial state and $\oplus$ for an accepting state.

86 Definition. *A **transition graph** is a directed graph G with finitely many vertices, one or more of which are labelled $\ominus$, and some of which are labelled $\oplus$. There are finitely many directed edges of the graph G each of which is labelled with a string from $\Sigma^\star$, for some alphabet Σ.*

Figure 5.1 gives an example of a transition diagram.

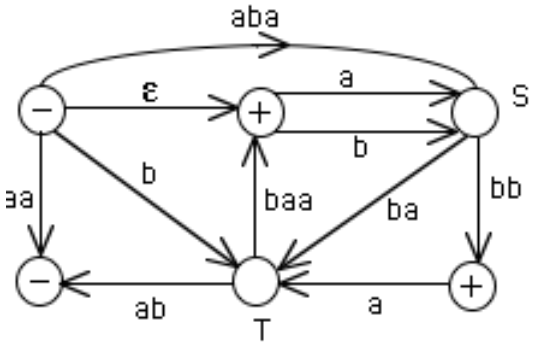


Figure 5.1: A transition graph.

You will notice that there are some significant differences between a graph like this, and a diagram representing a (deterministic finite-state) automaton.

1. For an automaton the number of edges leaving each vertex is always exactly the number of elements in the input alphabet. This is not the case for a transition diagram, as there can be any finite number of edges (or none) leaving a vertex.

2. For an automaton, the edges are labelled only with the letters from the input alphabet Σ. For a transition graph, the edges can be labelled with any string from $\Sigma^\star$, including ε, the empty string.

3. An automaton is completely **deterministic** in the sense that each string determines a unique path from the initial state. A transition graph is **nondeterministic** in the sense that there are possibly several paths, or perhaps none, that

correspond to a given string. For the transition graph of Figure 5.1 the string *aba* corresponds to two paths from the initial state; one leading to S and another leading to T.

Transition graphs represent nondeterministic finite state automata or NFA's. Despite the nondeterminism it is still possible to define the language of a transition graph, since any path on the graph determines a unique string from $\Sigma^\star$.

87 Definition. *The **language** $L(G)$ of a transition graph G is the set of strings that correspond to paths joining a vertex labelled $\ominus$ to a vertex labelled $\oplus$.*

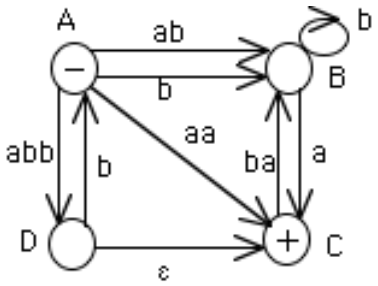


Figure 5.2: A transition graph.

In the transition graph G of Figure 5.2, $abb \in L(G)$ since *abb* provides a path from A to the accepting vertex C via the vertex D. Note however that *abb* also provides a path from A to the non-accepting vertex B using the edge from A to B labelled *ab*, followed by going around the loop at B once. A string $\omega \in \Sigma^\star$ is in $L(G)$ if and only if of all paths (if any) starting at an initial vertex and for which the concatenation of the edge labels gives ω, at least one ends at an accepting vertex.

88 Definition. *A **generalised transition graph** is the same as a transition graph, except that every edge is labelled with a regular expression.*

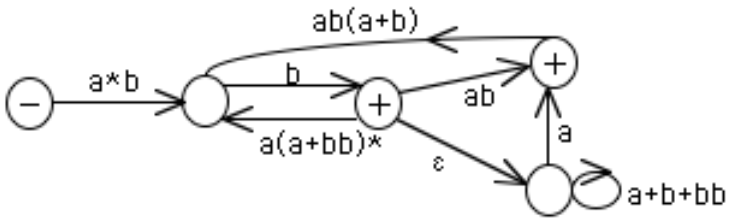


Figure 5.3: A generalised transition graph.

Figure 5.3 above provides an example of a generalised transition graph. Generalised transition graphs also represent nondeterministic finite state automata (NFA's). Any path in a generalised transition determines a regular expression that is obtained by concatenation of the expressions along the edges of the path. The regular expression then determines a regular set.

89 Definition. *The language $L(G)$ of a generalised transition graph G is the union of all sets corresponding to all directed paths joining a vertex labelled $\ominus$ to a vertex labelled $\oplus$.*

One of the main aims of this chapter is to show that the language of an automaton (DFA) is a regular set, and conversely, that any regular set is the language of some automaton. We don't do this directly, but rather use the generalised transition graph (NFA) we have just described. The proof is constructive, and as part of the proof we show how a nondeterministic automaton can be converted into a deterministic one.

5.3 Kleene's Theorem Part I

The theorem we prove is Kleene's Theorem. Stephen Kleene was an American mathematician/logician and he made important contributions to many branches of mathematical logic. This theorem results from his work for the RAND Corporation and dates from 1956. This section deals with just the first part of the theorem, but here is the statement in full:

90 Theorem. (Kleene)

Part I. *Let G be a generalised transition graph. Then $L(G)$ is a regular set. In particular, $L(A)$ is a regular set for any (deterministic finite state) automaton A, since an automaton is a special case of a generalised transition graph.*

Part II *Let R be a regular set. Then there is a (deterministic finite state) automaton A, such that $L(A) = R$.*

Proof of Part I. The proof is in several steps. At each stage we transform the transition graph in such a way that the languages of the original and changed graphs are the same. **Step 1.** This step is only necessary if the transition graph has more than one vertex labelled $\ominus$. This will not often happen in the context in which we are working. The aim of this step is to replace all the vertices labelled $\ominus$ by a single vertex labelled $\ominus$. To do this we introduce a new vertex labelled $\ominus$, and we make directed edges to the

vertices that were originally labelled $\ominus$. We label the new edges with ε, the empty string, and remove the designation $\ominus$ from the original vertices. Figure 5.4 below illustrates this procedure.

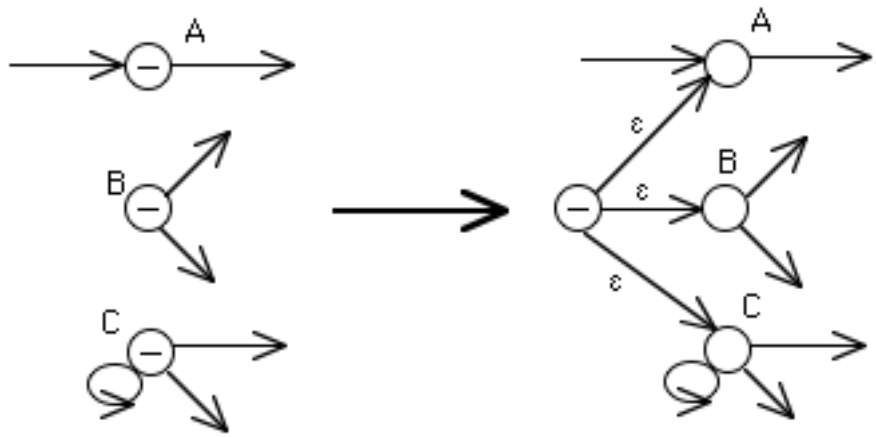


Figure 5.4: Step 1.

Note that the language of the resulting graph is the same as the language of the original graph.

Step 2. We examine the (unique) initial vertex (labelled $\ominus$). If it has only edges going *from* it, and no loops, we make no change at this stage. But if it has one or more edges going *into* it, or one or more loops, we introduce a new vertex labelled $\ominus$, and we make a directed edge to the vertex that was originally labelled $\ominus$. The new edge is labelled with the empty string, ε, and the designation $\ominus$ is taken away from the original initial vertex. Figure 5.5 below illustrates this procedure.

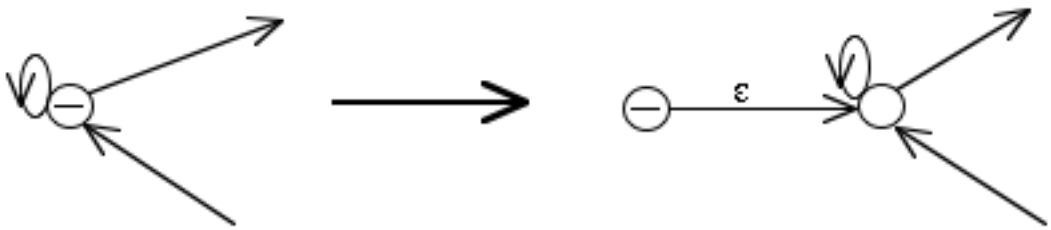


Figure 5.5: Step 2.

As a result of this procedure, the language of the resulting graph will be the same as that of the original graph.

Step 3. It is quite often the case that a transition graph has more than one accepting vertex. The aim of this step is to make a graph that has only one accepting vertex, which we may call the terminal vertex. The procedure is very similar to the procedure outlined in Step 1 and is illustrated in Figure 5.6 below.

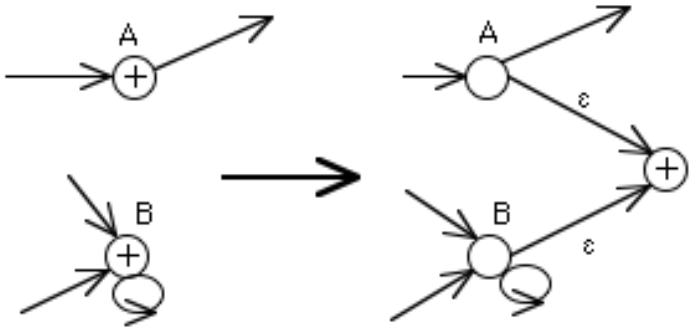


Figure 5.6: Step 3.

With this procedure the language of the new graph will be the same as the language of the original graph.

Step 4. This step is similar to Step 2, except we examine the terminal vertex. If the terminal vertex, $\oplus$, has only edges going into it, and no loops, then we make no change. If the terminal vertex has edges going from it, or has any loops, we introduce a new vertex labelled $\oplus$, and a new edge from the original terminal vertex to the new terminal vertex. We label the new edge with ε, the empty string, and remove the $\oplus$ designation from the original terminal vertex.

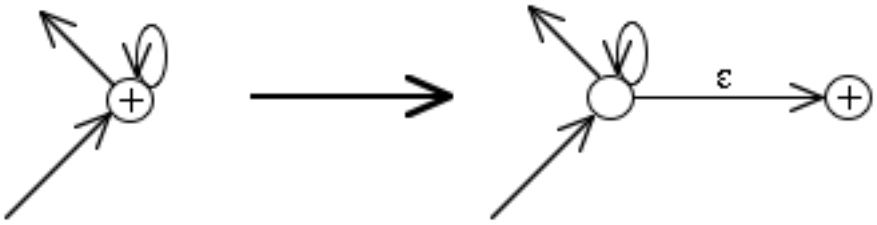


Figure 5.7: Step 4.

The language of the new graph remains the same as the language of the original graph.

Step 5. If there is more than one edge connecting a vertex u to a vertex v we combine these edges into one edge and use the "+" operation to combine the expressions that

label each edge. If there is more than one loop at a single vertex u we replace the loops with one loop using the "+" operation to combine the expressions on each loop. This is illustrated in Figure 5.8 below:

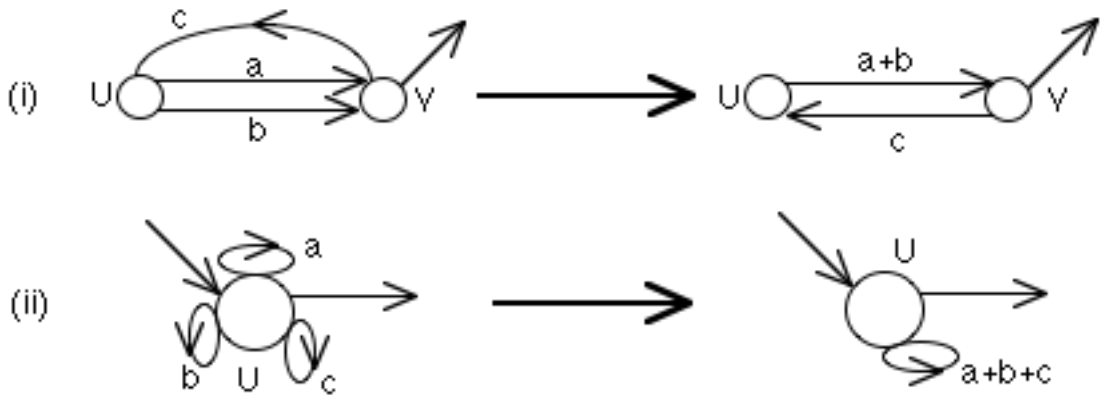


Figure 5.8: Step 5.

Step 6. The aim of the remainder of the process is to eliminate *intermediate* vertices, *i.e* vertices that are not labelled $\ominus$ or $\oplus$. If there are no such vertices we can go directly to step 8. Otherwise, for a given intermediate vertex V any string in the language of G that uses vertex V must enter V on one directed edge. Let's say it comes from vertex W. It may loop around V for a finite number of steps, if there is a loop there. Finally it will exit along a directed edge to a vertex X. Assuming for a moment that $X \neq W$ we have a fragment of the transition graph that looks something like what is shown below in Figure 5.9:

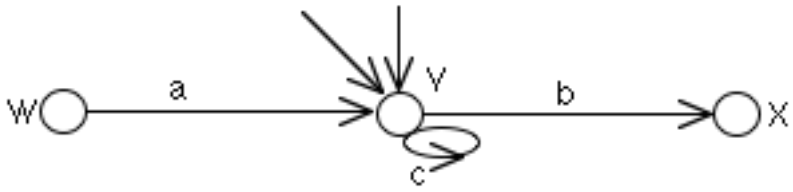


Figure 5.9: Edges through vertex V.

In this situation we introduce a new edge from W to X and label it with the expression $ac^{\star}b$. If there is no loop the new edge is just labelled ab. This resulting fragment of the graph looks as shown in Figure 5.10.

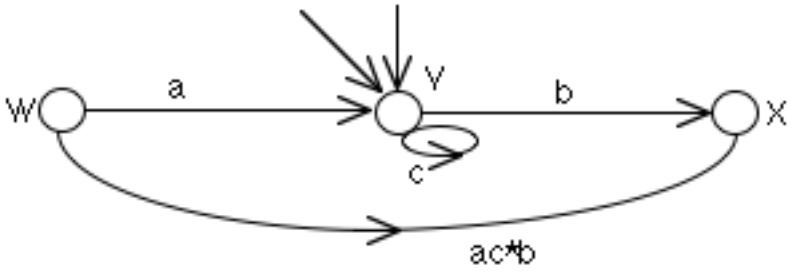


Figure 5.10: A new edge.

Now we must consider the case where $X = W$. In this case a string in the language uses vertex V by entering V on a directed edge from a vertex W, possibly then looping around V for a finite number of steps (if there is a loop there) and finally exiting V by a directed edge that goes back to W. An example is illustrated in Figure 5.11 below:

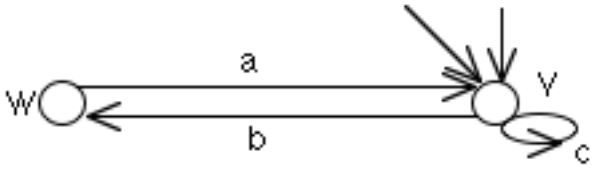


Figure 5.11: Edges from W through V and back to W.

The procedure described above then creates a loop at the vertex W, again labelled $ac^{\star}b$. See Figure 5.12 below:

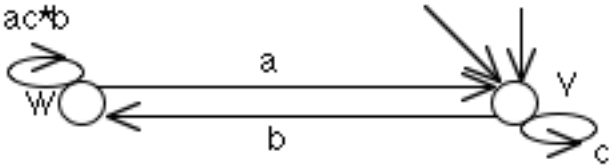
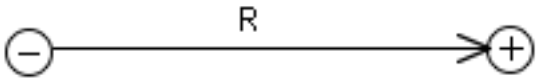


Figure 5.12: A new loop at vertex W.

We repeat this step at vertex V using every possible choice of W and X. When this is all done we then eliminate vertex V from the graph altogether, along with any of the edges entering or leaving it.

Step 7. Repeat Steps 5 and 6 until all intermediate vertices and parallel edges have been eliminated.

Step 8. At this stage we are left with a generalised transition graph of the form shown in Figure 5.3 below:



.

The language of this transition graph is the regular set R. This is also the language of the original graph. □

This completes the proof of Part I of Kleene's Theorem. It shows that the language of a transition graph (and therefore of a finite state automaton) is a regular set. An outline of the algorithm is as follows:

1. Normalise the initial vertices by introducing a single initial vertex $\ominus$ that only has edges going from it. (Steps 1 and 2).
2. Normalise the accepting vertices by introducing a single 'terminal' accepting vertex $\oplus$ that only has edges going into it. (Steps 3 and 4).
3. While there are intermediate vertices (vertices other than $\ominus$ and $\oplus$) :
   (i) Combine multiple edges and combine multiple loops using +. (Step 5)
   (ii) Eliminate an intermediate vertex and all its incident edges by introducing new compensating edges. (Step 6)
4. Combine any remaining multiple edges using +.
5. Find the regular set R.

We illustrate the algorithm with three examples:

Example 1

We find the language of the transition graph shown in Figure 5.13 below.

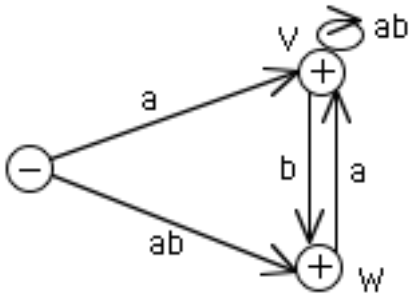


Figure 5.13: Example 1.

In this graph there is only one vertex labelled $\ominus$, and it is normalised, so we move to step 3.
We have two vertices, V and W, labelled $\oplus$, so we add a new vertex, labelled $\oplus$. We remove the labelling from vertices V and W, and add directed edges from both V and W to the new vertex. These edges are labelled with ε, the empty string. The new graph is shown in Figure 5.14 below.

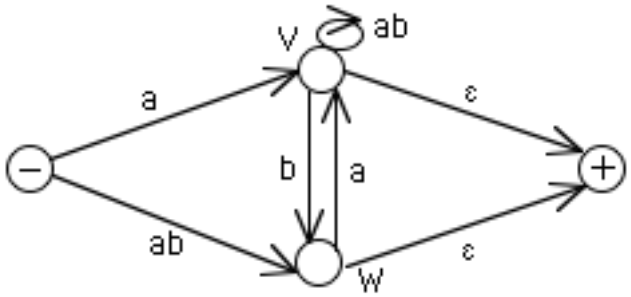


Figure 5.14: There is now only one accepting state.

There is now a single terminal vertex and it is normalised, so we can move to step 5.
Since There are no multiple edges, we bypass step 5 and move to step 6. We aim to eliminate vertices V and W, and we can choose which one we want to eliminate first. It will not make any difference to the language of the graph, but the expression we finally get may be different depending on the choices we make.
Say we eliminate vertex V first. We will introduce a new edge from $\ominus$ to $\oplus$, labelled $a(ab)^\star\varepsilon = a(ab)^\star$. A second new edge will go from $\ominus$ to W carrying the label $a(ab)^\star b$. Also we will add a loop at W labelled $a(ab)^\star b$. Finally, we will add an edge from W to $\oplus$, labelled $a(ab)^\star$. This takes into account every path that goes through vertex V, so we eliminate vertex V, and the edges going into and out of it. We get the new graph shown in Figure 5.15 below. This completes the first round through step 6.

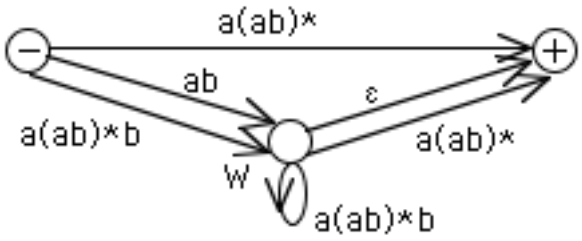


Figure 5.15: Vertex V has been eliminated.

Step 7 directs us back to step 5. We now have two pairs of parallel edges so we combine these to get the graph shown in Figure 5.16 below. Note that $\varepsilon + a(ab)^\star$ is different from $a(ab)^\star$, and the combined edge form W entering $\oplus$ needs to be labelled $\varepsilon + a(ab)^\star$.

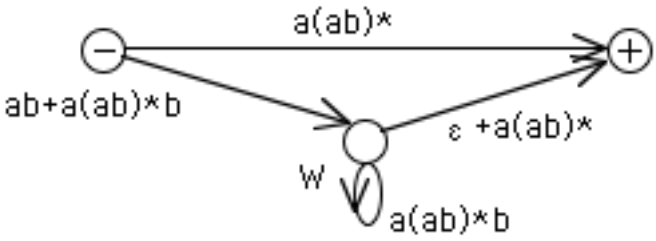


Figure 5.16: Two sets of parallel edges have been combined.

We now follow the instructions of step 6 again to eliminate the vertex W. We need add only one new edge, from $\ominus$ to $\oplus$. It will be labelled $[ab + a(ab)^\star b][a(ab)^\star b]^\star[\varepsilon + a(ab)^\star]$, yielding the graph in Figure 5.17 below.

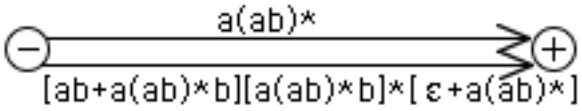


Figure 5.17: Vertex W has been eliminated.

Step 7 again directs us back to step 5. We have multiple edges, so we combine these to get the final diagram shown in Figure 5.18 below.

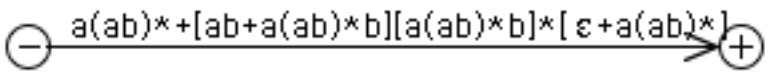


Figure 5.18: Final transition graph for Example 1 shown in Figure 5.13

We conclude that the language represented by the original graph is

$$R = a(ab)^\star + [ab + a(ab)^\star b][a(ab)^\star b]^\star[\varepsilon + a(ab)^\star].$$

It is a good exercise to repeat this procedure, eliminating vertex W before eliminating vertex V. The resulting expression should be

$$R' = ab + (a + aba)(ab + ba)^\star(\varepsilon + b).$$

This expression is actually simpler to write. You might try to convince yourself that the two expressions represent the same regular set.

Example 2

This example shown in Figure 5.19 below is similar to the last one, but with some differences.

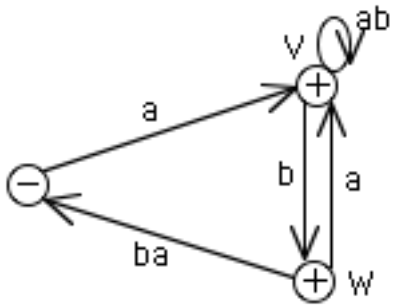


Figure 5.19: Example 2.

There is only a single initial vertex so we can move directly to step 2. Since the initial vertex has an edge entering it, it is not in normal form. So following step 2, we add a new initial vertex and an edge from the new initial vertex to the original initial vertex. This edge will be labelled with ε. The result is shown in Figure 5.20 below.

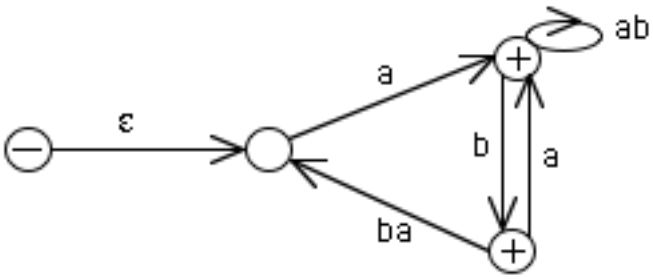


Figure 5.20: Initial vertex normalised.

We move to step 3. Since there are two terminal vertices, we make a new terminal vertex and we add edges from the old terminal vertices to the new one. We also remove the designation $\oplus$ from the original terminal vertices. This is shown in Figure 5.21.

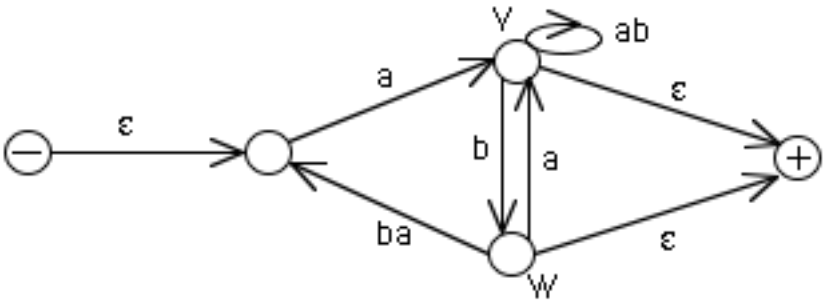


Figure 5.21: Accepting vertices normalised.

There are no parallel edges, so we move to step 6 and eliminate a vertex. Let us eliminate vertex W first. Considering all the paths through W the new graph becomes (Figure 5.22):

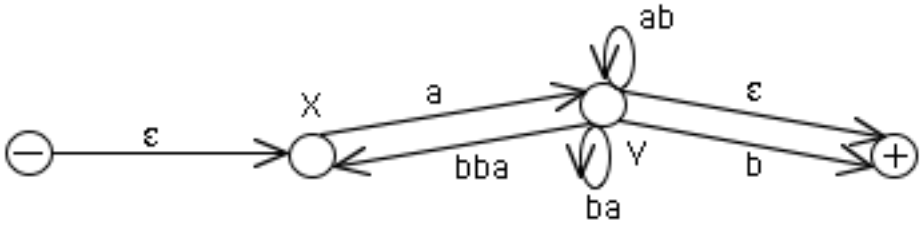


Figure 5.22: Vertex W eliminated.

We combine the parallel edges from V to the terminal vertex, and the two cycles at V to get Figure 5.23 :

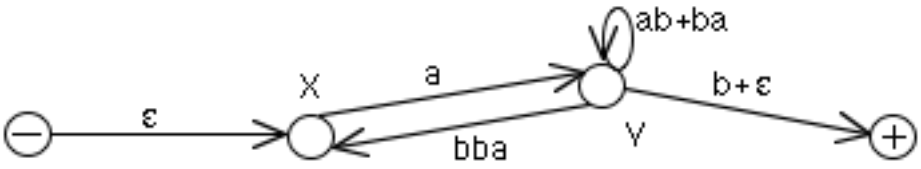


Figure 5.23: Parallel edges combined.

We can eliminate either vertex X or vertex$V$. Choosing X leads to Figure 5.24.

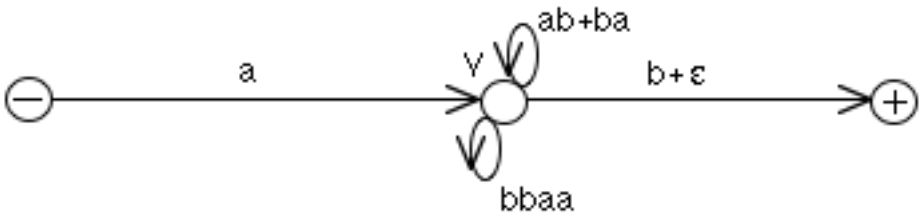


Figure 5.24: Vertex X eliminated.

We now have parallel loops at V so we combine these using "+" to get Figure 5.25:

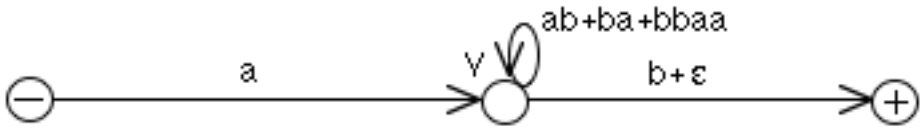


Figure 5.25: Parallel loops combined.

Finally we must eliminate vertex V: (Figure 5.26)

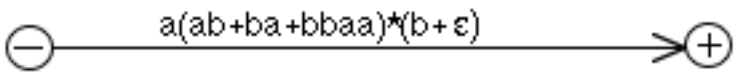


Figure 5.26: Vertex V eliminated.

We have now finished and can say that the language of the initial graph is

$$R = a(ab + ba + bbaa)^\star(b + \varepsilon).$$

It is always good, when you have found an expression like R to check back with the original graph to see that acceptable strings in the graph actually belong in the set given by R, and that strings that belong in R are acceptable strings for the graph. Again, it is also a good idea to try eliminating vertices in a different order. If we eliminate vertex V, then vertex X and finally vertex W, we will get an expression, (please check)

$$R = a(ab)^\star + a(ab)^\star b[a(ab)^\star b + baa(ab)^\star b]^\star[a(ab)^\star + \varepsilon].$$

These expressions look very different and it takes some convincing to believe that they describe the same set. It makes it clear that it would be good to have some methods for simplification of these expressions.

Example 3

We take the transition graph shown in Figure 5.27 as a final example.

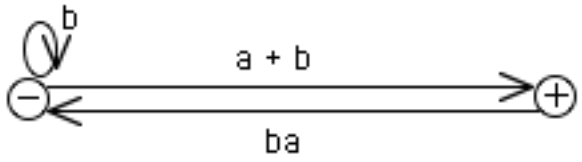


Figure 5.27: Example 3.

Since there is a loop at the initial vertex and an edge going into it, we need to normalise the initial vertex. Since there is an edge going from the terminal vertex we also need to normalise it. This gets us to Figure 5.28

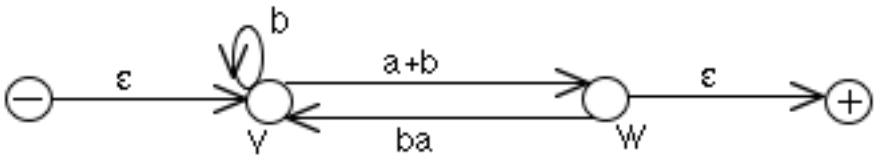


Figure 5.28: Initial and terminal vertices normalised.

We have a choice between eliminating vertex V or vertex W. I suggest we eliminate vertex W first. Since it has fewer edges going in and out, it seems to make the process simpler if we eliminate it first. We get Figure 5.29.

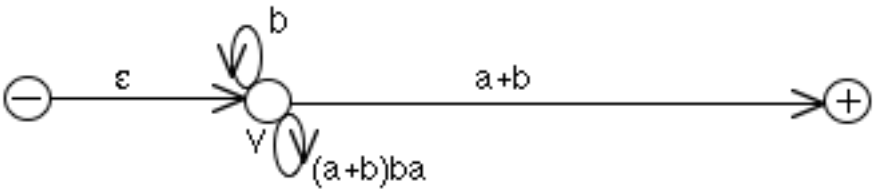


Figure 5.29: Vertex W eliminated.

We now have parallel loops at vertex V. Combining these using "+" gives Figure 5.30.

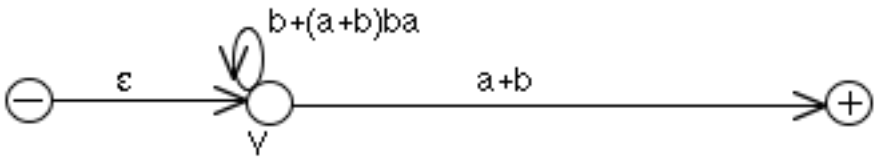


Figure 5.30: Parallel loops combined.

For the final step we eliminate vertex V giving Figure 5.31

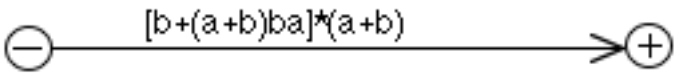


Figure 5.31: Verex V eliminated.

From this we read that

$$R = [b + (a+b)ba]^{\star}(a+b).$$

Eliminating vertex V first would have given: (Check this please)

$$R = b^{\star}(a+b)[bab^{\star}(a+b)]^{\star}.$$

Again these expressions look different. Try to convince yourself that they both represent the same regular set.

5.4 Kleene's Theorem Part II

We have proved the first part of Kleene's Theorem, including that the language of a DFA is a regular set. We now want to show that, given a regular set R we can construct a DFA A, such that $L(A) = R$.

NFA's With Language R.

Let R be any regular set. It is quite easy to find an NFA, or generalised transition graph, whose language is R. It is the generalised transition graph that has an initial vertex and a terminal vertex, with just one directed edge from the initial vertex to the terminal vertex. The edge is labelled with the expression R. See Figure 5.32

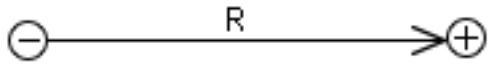
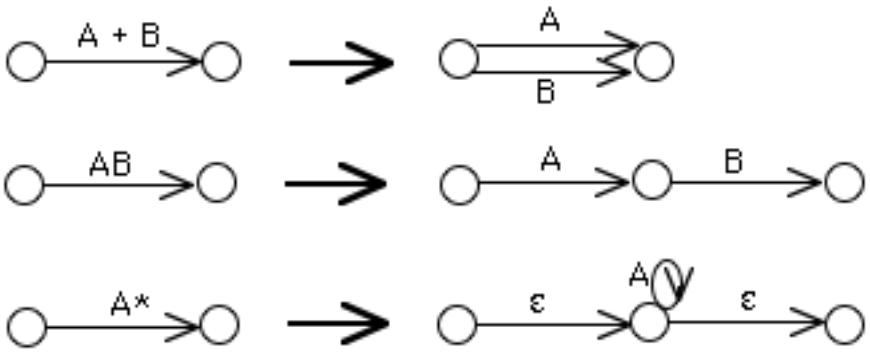


Figure 5.32: Basic generalised transition graph for language R.

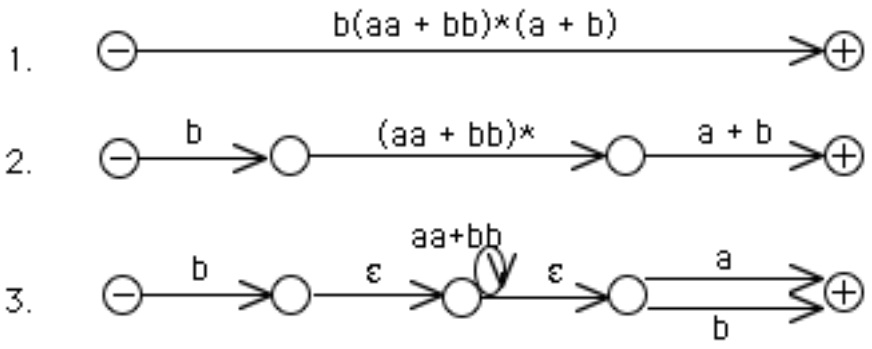
This is not very helpful in ultimately finding a DFA that accepts the language R, so we find a series of NFA's until we find one whose edges are labelled with the letters of the input alphabet or with ε. The process used is very much the opposite of that used to find the language of an automaton. For example, if we have a directed edge between two vertices carrying the label $A+B$, we replace it with it with two edges between the same pair of vertices. One edge is labelled A, the other labelled B. This step, and other replacements are shown below.



Each of these replacements leaves the language of the transition graph unchanged. This process is continued until we arrive at a transition graph whose edges are labelled with the letters of the input alphabet (usually a and b) or with ε. We illustrate the procedure with some examples.

Example 1

Consider $R = b(aa+bb)*(a+b)$. We show the series of conversions that results in a transition graph whose language is R.



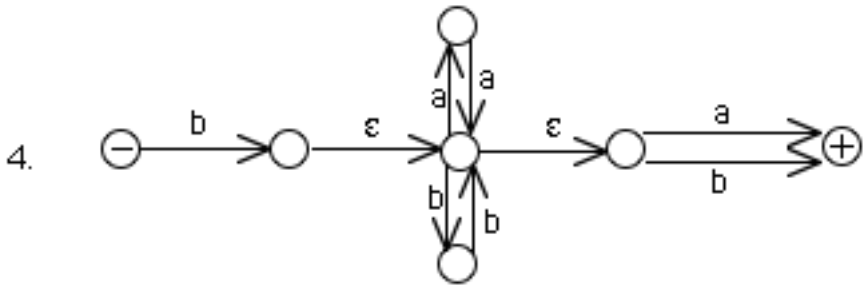
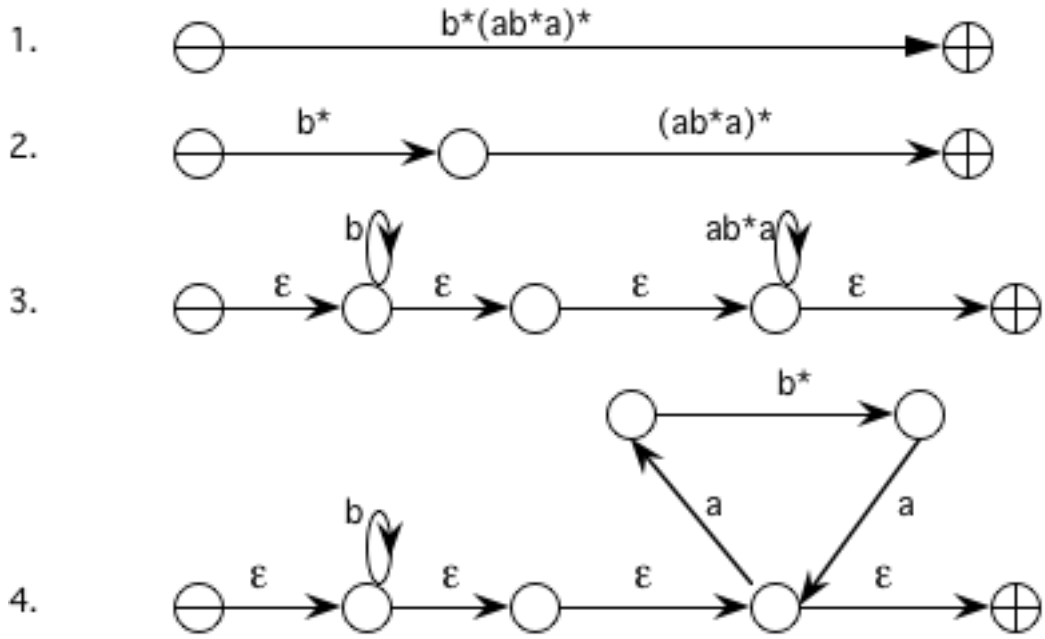


Figure 5.33: The resulting transition graph for Example 1: $R = b(aa+bb)*(a+b)$.

Note that, as expected, this transition graph (Figure 5.33 above) is not deterministic. Input of string ba leads to both the terminal vertex and the vertex at the top of the diagram. Also there are not two arrows, labelled a and b respectively, from each vertex.

Example 2

Consider $R = b^\star(ab^\star a)^\star$. We proceed as in the previous example.



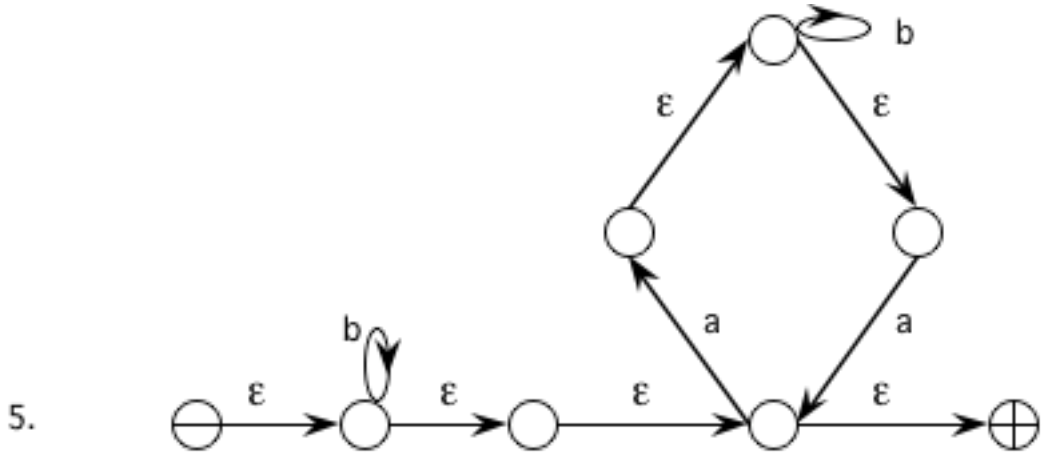


Figure 5.34: The resulting transition graph for Example 2: $R = b^\star(ab^\star a)^\star$.

Figure 5.34 above completes the process we started, in that we have constructed a transition graph whose edges are labelled only with the letters of the input alphabet and the empty string, ε.

From Nondeterministic to Deterministic

The main aim of these sections is to start with a regular set, given by an expression R, and to construct a deterministic finite state automaton (DFA) with this regular set as its language. So far, we have given a procedure for starting with a regular set R, and constructing a transition graph, that we can view as a nondeterministic finite state automaton (NFA), that has language R. The next step in the process is to show how we can construct a DFA with the same language as our NFA. We will investigate this stage of the process by showing how to convert several NFA's to DFA's.

Example 3

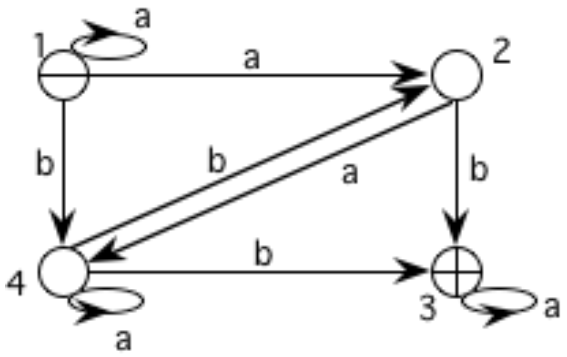


Figure 5.35: A nondeterministic automaton.

Our first step in converting this NFA to a DFA is to construct a transition table for the NFA:

s	sa	sb
1	1,2	4
2	4	3
3	3	
4	4	2,3

The format of this table is only slightly different from that of the now famililiar transition tables for deterministic machines. The difference is that when the machine is in a state s, and a letter of the input alphabet is input, the outcome is not necessarily a single state. As above, the outcome may simply contain one element, or it may be more than one element, or it may not be any state.

We aim to construct an automaton whose vertices are subsets of the states of the nondeterministic machine. In general, suppose that the vertices are labelled $1,2,\ldots,k$, with $\ominus$ labelled 1. We start a table with three columns, labelled S, Sa and Sb. We start with $S = \{1\}$ in the first column. In the column labelled Sa we put the set of all the possible outcomes for elements in the set S when a is input. In column Sb we put the set of all the possible outcomes when b is input into any of the states in S. As a new set of states arises in the columns Sa or Sb we put it in the column labelled S. We continue this process until no new sets of states arise.

The table for the transition graph shown above begins as follows.

S	Sa	Sb
$\{1\}$	$\{1,2\}$	$\{4\}$
$\{1,2\}$		
$\{4\}$		

For $S = \{1,2\}$, we find $Sa = \{1,2,4\}$ (the union of the states that occur when a is input into state 1 and state 2) and $Sb = \{3,4\}$. For $S = \{4\}$, $Sa = \{4\}$ and $Sb = \{2,3\}$. So we extend the previous table as shown below.

S	Sa	Sb
$\{1\}$	$\{1,2\}$	$\{4\}$
$\{1,2\}$	$\{1,2,4\}$	$\{3,4\}$
$\{4\}$	$\{4\}$	$\{2,3\}$
$\{1,2,4\}$		
$\{3,4\}$		
$\{2,3\}$		

We continue the process. The completed table is shown below.

	S	Sa	Sb
$\ominus$	$\{1\}$	$\{1,2\}$	$\{4\}$
	$\{1,2\}$	$\{1,2,4\}$	$\{3,4\}$
	$\{4\}$	$\{4\}$	$\{2,3\}$
	$\{1,2,4\}$	$\{1,2,4\}$	$\{2,3,4\}$
$\oplus$	$\{3,4\}$	$\{3,4\}$	$\{2,3\}$
$\oplus$	$\{2,3\}$	$\{3,4\}$	$\{3\}$
$\oplus$	$\{2,3,4\}$	$\{3,4\}$	$\{2,3\}$
$\oplus$	$\{3\}$	$\{3\}$	$\emptyset$
	$\emptyset$	$\emptyset$	$\emptyset$

This is now the state table for a (deterministic) finite state automaton. It has nine states, each one labelled by a subset of the set of states of the original deterministic machine. The initial state is the state $S = \{1\}$, where state 1 was the initial state of the original machine. The accepting states are those states containing any of the accepting states of the original graph. In the example the accepting state was 3, so any subset that contains

3 becomes an accepting state. Notice that when b is input into the state $\{3\}$, there is no state designated for the machine to go. We represent this situation by using the empty set, which then becomes one of the subsets on the state table.

The deterministic automaton we have constructed from the non-deterministic one of Figure 5.35 is shown below in Figure 5.36

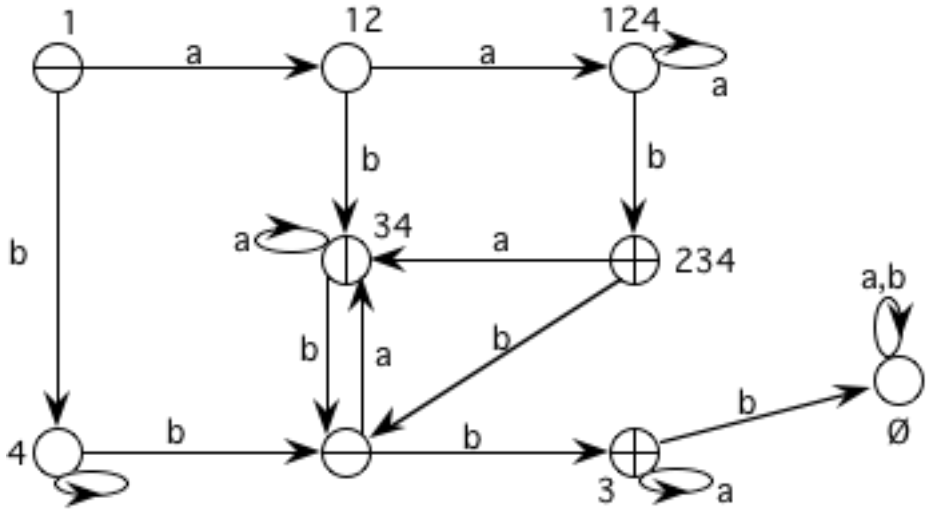


Figure 5.36: The deterministic automaton.

Example 4

We convert the nondeterministic automaton in Figure 5.37 to a deterministic one.

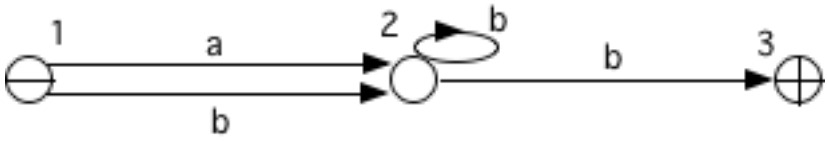


Figure 5.37: A second NFA to be converted to a DFA.

The first step is to make a transition table for this machine.

s	sa	sb
1	2	2
2		2,3
3		

We now aim to construct the transition table for a deterministic automaton with the same language.

	S	Sa	Sb
$\ominus$	$\{1\}$	$\{2\}$	$\{2\}$
	$\{2\}$	$\emptyset$	$\{2,3\}$
$\oplus$	$\{2,3\}$	$\emptyset$	$\{2,3\}$
	$\emptyset$	$\emptyset$	$\emptyset$

This table looks very similar to the previous one. The difference is that for this table the column S contains only sets that have arisen in the previous lines of the table. This is except for the first entry in the column S, which is always the set containing the initial vertex. The deterministic automaton with this transition table is shown in Figure 5.38 below.

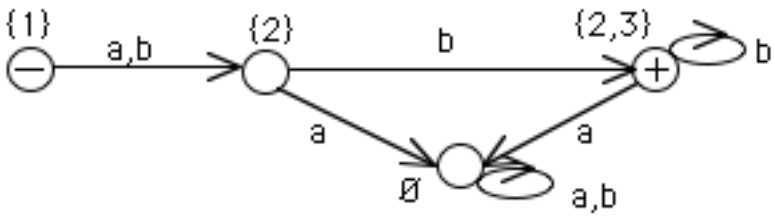


Figure 5.38: The DFA from the NFA of Figure 5.37.

Return to Example 2

In Example 2 we constructed a transition graph for the language $b*(ab*a)*$. The graph was shown as Figure 5.34 and as is repeated below for as 5.39 for convenience. We want to convert this NFA to a DFA for the same language.

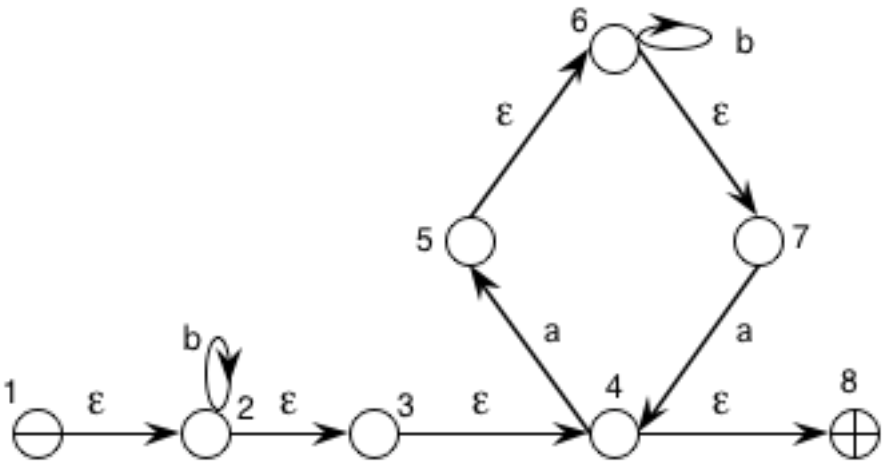


Figure 5.39: The NFA from Example 2.

Again, the first step is to make a transition table for this NFA.

s	sa	sb
1		
2		2
3		
4	5	
5		
6		6
7	4	
8		

The table is vary sparsely populated. As we have seen in the previous examples, the empty spaces represent 'missing arrows'. That is, when there is no entry in row s, column sa, it means that the graph has no edge labelled a leading from vertex s, and no entry in row s, column sb, means there is no edge labelled b leading from vertex s.

The row representing the initial vertex is completely empty, and so we can't immediately make a start on the transition table for the required DFA by following the procedure used for Examples 3 and 4. Instead we use a modified procedure that takes into account the edges labelled by ε .

We define the **ε-closure** of a state s as the set of states that can be reached from s using only edges labelled with ε. In our current example we have

$$\varepsilon\text{-closure}(1) = \{1,2,3,4,8\} \qquad \varepsilon\text{-closure}(2) = \{2,3,4,8\} \qquad \varepsilon\text{-closure}(3) = \{3,4,8\}$$

$$\varepsilon\text{-closure}(4) = \{4,8\} \qquad \varepsilon\text{-closure}(5) = \{5,6,7\} \qquad \varepsilon\text{-closure}(6) = \{6,7\}$$

$$\varepsilon\text{-closure}(7) = \{7\} \qquad \varepsilon\text{-closure}(8) = \{8\}.$$

More generally, if S is a set of states of an NFA we define the ε-closure of S by

$$\varepsilon\text{-closure}(S) = \bigcup_{s\in S} \varepsilon\text{-closure}(s).$$

So, for example with our current NFA,

$$\varepsilon\text{-closure}(\{3,6\}) = \{3,4,8\} \cup \{6,7\} = \{3,4,6,7,8\},$$

$$\varepsilon\text{-closure}(\{3,4\}) = \{3,4,8\} \cup \{4,8\} = \{3,4,8\}.$$

We shall say that a set of vertices (states) S of a transition graph is **ε-closed** if its ε-closure is itself. In particular, the ε-closure of any set of states is automatically ε-closed. The states of our DFA will all be ε-closed sets of states of our NFA. The modified procedure is as follows:

First, the initial state of the DFA will be the ε-closure of the initial state of the NFA. So in our example the first entry in column S of the transition table of our DFA will be ε-closure($\{1\}$), rather than $\{1\}$. Second, our transition table will have five columns instead of three. The two extra columns will be labelled ε-closure(Sa) and ε-closure(Sb). New ε-closed sets will be added in the column S when they appear in either of the columns ε-closure(Sa) or ε-closure(Sb).

So the table begins like this:.

S	Sa	Sb	ε-closure(Sa)	ε-closure(Sb)
$\{1,2,3,4,8\}$	$\{5\}$	$\{2\}$	$\{5,6,7\}$	$\{2,3,4,8\}$
$\{5,6,7\}$				
$\{2,3,4,8\}$				

The table is completed when no new sets arise in either of the columns ε-closure(Sa) or ε-closure(Sb).

S	Sa	Sb	ε-closure(Sa)	ε-closure(Sb)
$\{1,2,3,4,8\}$	$\{5\}$	$\{2\}$	$\{5,6,7\}$	$\{2,3,4,8\}$
$\{5,6,7\}$	$\{4\}$	$\{6\}$	$\{4,8\}$	$\{6,7\}$
$\{2,3,4,8\}$	$\{5\}$	$\{2\}$	$\{5,6,7\}$	$\{2,3,4,8\}$
$\{4,8\}$	$\{5\}$	$\emptyset$	$\{5,6,7\}$	$\emptyset$
$\{6,7\}$	$\{4\}$	$\{6\}$	$\{4,8\}$	$\{6,7\}$
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

To construct the DFA we use the columns S, ε-closure(Sa) and ε-closure(Sb). The initial state is $\{1,2,3,4,8\} = A$ say, and the accepting states are those containing the accepting states of the NFA, in this case: A and C $\{2,3,4,8\} = C$ say and $\{4,8\} = D$ say. Let us also put $B = \{5,6,7\}, E = \{6,7\}$ and$F = \emptyset$. The state table then looks like

	S	Sa	Sb
$\ominus,\oplus$	A	B	C
	B	D	E
$\oplus$	C	B	C
$\oplus$	D	B	F
	E	D	E
	F	F	F

The transition diagram for the DFA is shown below in Figure 5.40

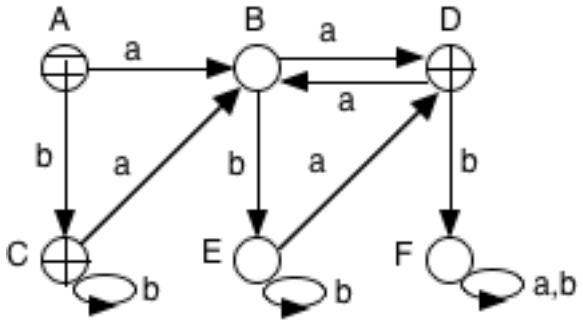


Figure 5.40: The DFA for Example 2.

This DFA has the same language as the NFA from Example 2, which was the language given by the regular expression $R = b^\star(ab^\star a)^\star$. We could end this example now as we have constructed a DFA with the required language. However, when we look closely it is fairly easy to see that we can simplify this even further. We can find the quotient automaton for the DFA shown above. It is a good exercise to check the equivalence classes, but they are $[A] = \{A,C\} = [C]$, $[B] = \{B,E\} = [E]$, $[D] = \{D\}$, and $[F] = \{F\}$. The quotient automaton is shown as diagram 5.41 below.

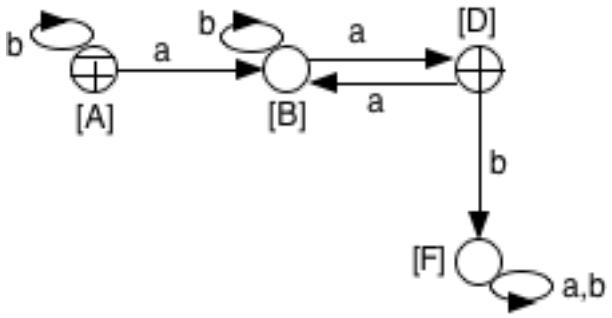


Figure 5.41: The simplest DFA for the language $b^\star(ab^\star a)^\star$ of Example 2 .

Example 5

The main point of this whole section is to show that any regular expression is the language of a DFA. So, as a final example we will take a regular expression R, from R we will construct an NFA and finally we will construct a DFA with the same language. This process can be followed for any regular expression. Let us take $R = (ab+ba)^\star a^\star$. The following diagrams show the steps involved in constructing an NFA with this language.

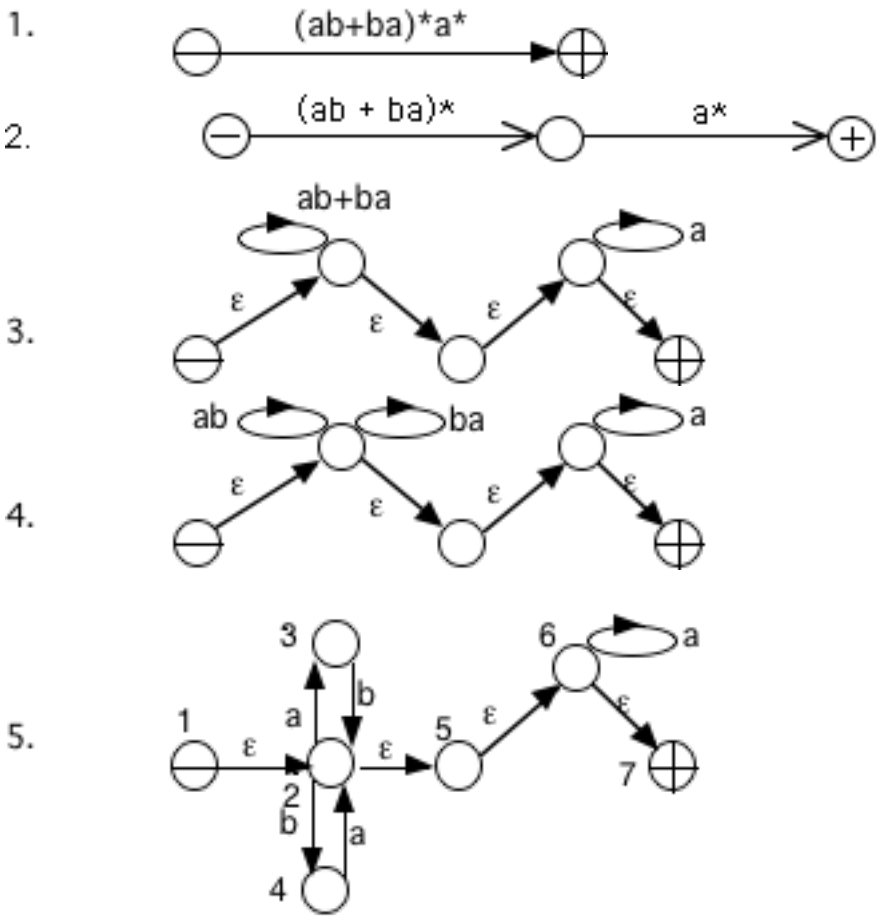


Figure 5.42: NFA for the language $R = (ab + ba)^\star a^\star$ of Example 5.

At this stage we have constructed an NFA with R as its language. We now need to use the same method as in the previous example (continuation of Example 2) to find an appropriate DFA. We first make a transition table for the NFA.

s	sa	sb
1		
2	3	4
3		2
4	2	
5		
6	6	
7		

Since the NFA has edges labelled with ε we also find the ε-closures of each state.

ε-closure$(1) = \{1,2,5,6,7\}$ ε-closure$(2) = \{2,5,6,7\}$ ε-closure$(3) = \{3\}$

ε-closure$(4) = \{4\}$ ε-closure$(5) = \{5,6,7\}$ ε-closure$(6) = \{6,7\}$

ε-closure$(7) = \{7\}$.

We can now begin the transition table for a DFA with the same language.

S	Sa	Sb	ε-closure(Sa)	ε-closure(Sb)
$\{1,2,5,6,7\}$	$\{3,6\}$	$\{4\}$	$\{3,6,7\}$	$\{4\}$
$\{3,6,7\}$	$\{6\}$	$\{2\}$	$\{6,7\}$	$\{2,5,6,7\}$
$\{4\}$	$\{2\}$	$\emptyset$	$\{6,7\}$	$\emptyset$
$\{6,7\}$	$\{6\}$	$\emptyset$	$\{6,7\}$	$\emptyset$
$\{2,5,6,7\}$	$\{3,6\}$	$\{4\}$	$\{3,6,7\}$	$\{4\}$
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

At this stage the table is complete as no new sets arise in the ε-closure columns. Take

$$A = \{1,2,5,6,7\}, B = \{3,6,7\}, C = \{4\}, D = \{6,7\}, E = \{2,5,6,7\} \text{ and } F = \emptyset.$$

The transition table is shown below.

	S	Sa	Sb
$\ominus, \oplus$	A	B	C
$\oplus$	B	D	E
	C	E	F
$\oplus$	D	D	F
$\oplus$	E	B	C
	F	F	F

The accepting states are indicated and the transition graph is shown in Figure 5.43 below.

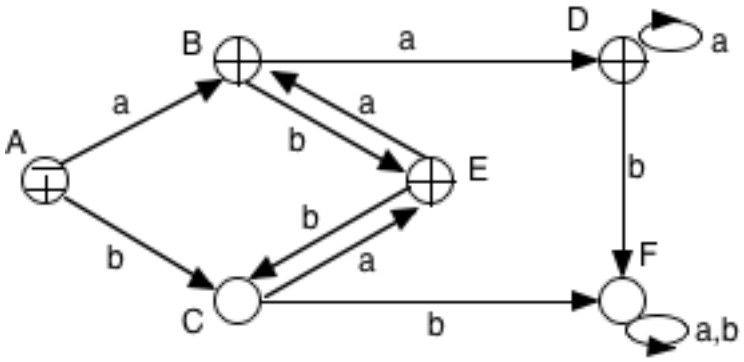


Figure 5.43: A DFA for $R = (ab + ba)^\star a^\star$ of Example 5.

Again this can be simplified by finding the quotient automaton. This is drawn as Figure 5.44 below. It is a good exercise to check that it is correct.

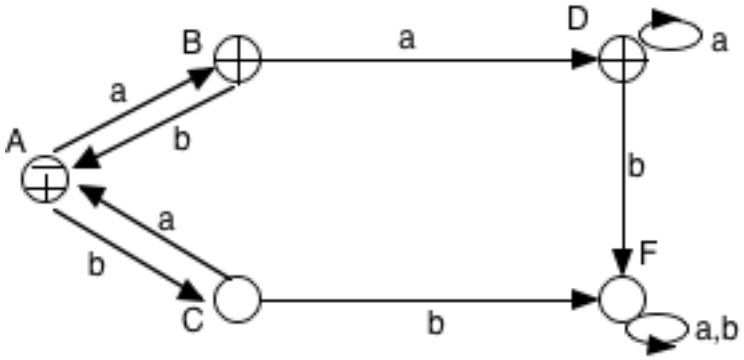
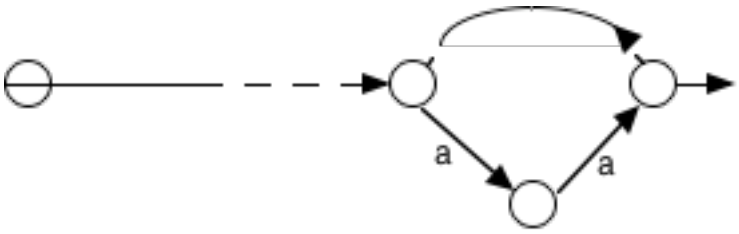


Figure 5.44: Simplest DFA for $R = (ab + ba)^\star a^\star$ of Example 5.

Example 6

Let $L = \{a^n b^n : n = 1, 2, \ldots K\}$. L is the set of strings with an equal number of a's and b's. We show that L is not the language of a DFA, and then by Kleene's theorem deduce that L is not a regular set.

Suppose that A is a DFA for which$L = L(A)$, and suppose that A has N vertices. Consider the string $\alpha = a^N b^N$ which is in L. T he path for α starts at $\ominus$ and must end at a terminal vertex labelled $\oplus$. The path for the part of α labelled a^N has N edges and uses $N = 1$ vertices (counting repetitions). Since A has only N vertices, one vertex at least is repeated in the path for a^N. So there must be a circuit in the path for a^N as shown below.



Suppose that this cycle has length $K \geq 1$. Since the path for α starts with a^N we can introduce a loop into the path before finishing the path for α. This means that the path $a^{N+K}b^N$ is in $L(A)$, because it begins at $\ominus$ and ends at a vertex labelled $\oplus$. But this string does not have an equal number of a's and b's and so is not in the language L. Hence $L(A) \neq L$, and L cannot be the language of any finite state automaton. Since L can't be the language of any automaton, it is not a regular set.

5.5 Grammars and Automata

In Section 3.4 we defined a phrase structure grammar as follows.

91 Definition. *A **phrase structure grammar** (or simply, **grammar**) G consists of*

1. *A finite set N of **non-terminal symbols**,*
2. *A finite set Σ of **terminal** symbols where $N \cap \Sigma = \emptyset$,*
3. *A **starting symbol** $S \in N$,*
4. *A finite set P of productions of the form $\alpha \Rightarrow \beta$, where $\alpha \in (N \cup \Sigma)^\star \setminus \Sigma^\star$ and $\beta \in (N \cup \Sigma)^\star$. (Note that $(N \cup \Sigma)^\star \setminus \Sigma^\star$ is the set of all strings from $N \cup \Sigma$ that contain at least one symbol from N.*

We write $G = (N, \Sigma, S, P)$.)

We also said that G is a **type 3 grammar** or **regular grammar** if for each production, $\alpha \Rightarrow \beta$ in P, α is a single symbol from N, and β has at most one non-terminal symbol, which must be at the extreme right. It is also allowed that $\beta = \varepsilon$. Further, we have called a language regular if it can be generated by a regular grammar.

In this chapter we have used the term 'regular expression', and 'regular set' for the set represented by a regular expression, without making any comment on the connection

between these and the earlier two usages of 'regular'. In this section we make that connection by showing that the terms 'regular set' and 'regular language' are coextensive; *i.e* a language is regular if and only if it is a regular set. Given that we now know that a set is regular if and only if it is the language recognised by an automaton, we can establish the aforementioned connection by showing that we can always construct a DFA from a regular grammar and *vice-versa*, with the corresponding language preserved in each direction.

From DFA to Regular Grammar

By way of introduction, consider the simple automaton shown below as Figure 5.45. This is a DFA that accepts those strings over $\Sigma = \{a, b\}$ that contain an odd number of a's. A regular expression for this language is $b^\star a(b + ab^\star a)$.

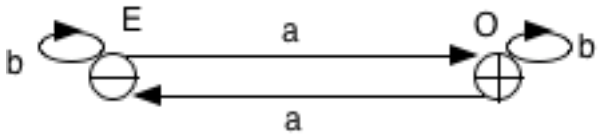


Figure 5.45: A simple automaton

To write a regular grammar for this language the terminal symbols must be $\Sigma = \{a, b\}$. For the non-terminal symbols we can use the states E and O, so $N = \{E, O\}$. The initial state E becomes the starting symbol and the productions correspond to the directed edges as follows: $E \xrightarrow{a} O$ and $O \xrightarrow{a} E$ become $E \Rightarrow aO$ and $O \Rightarrow aE$ respectively and similarly $E \xrightarrow{b} E$ and $O \xrightarrow{b} O$ become $E \Rightarrow bE$ and $O \Rightarrow bO$. Finally the fact that O is accepting becomes the additional production $O \to \varepsilon$, or alternatively the pair of productions $E \to a$ and $O \to b$. If we set P equal to the set of these five (or six) productions then $G = (N, \Sigma, S, P)$ is a regular grammar generating our language (the language accepted by the DFA whose diagram is shown in Figure 5.45 above.)

Moving to the general case, recall that for any DFA $A = (I, S, s_0, F, N)$, I is the input language, S is the set of states, s_0 is the initial sate, F is the set of accepting states and N is the transition function. We can always construct a regular grammar that generates the same language as A. In constructing such a grammar G the terminal alphabet Σ will be the same as the input alphabet I. The set of non-terminal symbols will be the set of states S and s_0 becomes the starting symbol. Productions $s \Rightarrow \sigma s'$ are introduced for each edge labelled σ from state s to state s' (*i.e.* for each transition $N(s, \sigma) = s'$).

Further productions $S \Rightarrow \varepsilon$ are introduced whenever S is accepting. Alternatively, so long as the initial state is not accepting, all productions involving ε can be avoided by introducing productions $s \Rightarrow \sigma$ for whenever there is an edge labelled σ from the state s to any accepting state. Then $G = \{S, I, s_0, P\}$ is a regular grammar which generates the same language as the automaton A.

Example 1

We find a regular grammar that generates the language of the DFA shown in Figure 5.46 below. The language has regular expression $b^\star(ab^\star a)^\star$. (Indeed, the DFA was constructed in Example 2 of Section 5.4 precisely to accept this language.)

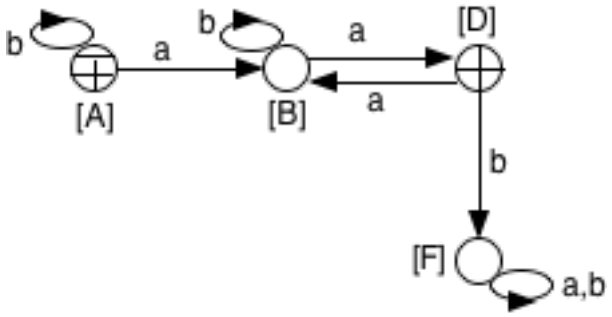


Figure 5.46: Construct a regular grammar for the language of this automaton.

We follow the instructions given above. The terminal alphabet is $\Sigma = \{a, b\}$. The non-terminal alphabet is $N = \{A, B, D, F\}$ and A is the starting symbol. The set P of productions is

$$\{ \quad A \Rightarrow aB \mid bA \mid \varepsilon, \quad B \Rightarrow aD \mid bB, \quad D \Rightarrow aB \mid bF \mid \varepsilon, \quad F \Rightarrow aF \mid bF \quad \}.$$

It will be seen that the subset of productions $\{D \Rightarrow bF,\ F \Rightarrow aF \mid bF\}$ can be removed from the set P without changing the language produced. This is because the use of these productions will never lead to a string in the language.

From Regular Grammar to DFA

We now describe the opposite process. We want to start with a regular grammar and produce a DFA with the same language. The process we will describe will actually produce an NFA. As we have seen in Section 5.4, from there it is always possible to convert to a DFA. As you would guess, we aim to reverse the procedures described above.

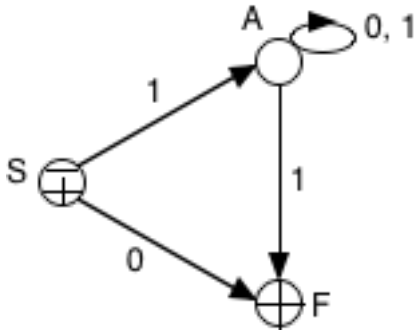
As an introductory example, consider the regular grammar $G = (N, \Sigma, S, P)$, where $N = \{A, S\}$, $\Sigma = \{0, 1\}$, and the productions in P are

$$S \Rightarrow 1A \mid 0 \mid \varepsilon, \quad A \Rightarrow 0A \mid 1A \mid 1.$$

We construct an NFA M, that has a state for each non-terminal symbol in G and an extra final (accepting) state F. So the set of states for M is $\{A, S, F\}$. The initial sate for M will be identified with the start symbol S. The transitions of M are formed from the productions of G. If there is a production of the form $s \Rightarrow \sigma$, we make an edge from the state s to F labelled with σ. If there is a production $s \Rightarrow \sigma s'$, we make an edge from state s to state s' labelled with σ. The set of accepting states comprises F and any state s for which there is a production $s \Rightarrow \varepsilon$ in G. Using this procedure we can make a transition table for M.

	s	$s0$	$s1$
$\ominus\oplus$	S	F	A
	A	A	A, F
$\oplus$	F		

The NFA M is shown below.



Example 2

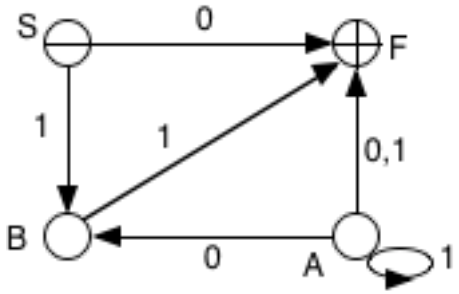
We construct an NFA with the same language as that generated by the grammar $G = (N, \Sigma, S, P)$, where $N = \{S, A, B\}$, $\Sigma = \{0, 1\}$, and the productions are

$$S \Rightarrow 1B \mid 0, \quad A \Rightarrow 1A \mid 0B \mid 1 \mid 0, \quad B \Rightarrow 1.$$

The set of states for the NFA is $\{S,A,B,F\}$ and the starting state is S. The only accepting state is F. The transition table is:

	s	$s0$	$s1$
$\ominus\oplus$	S	F	B
	A	B,F	A,F
	B		F
$\oplus$	F		

The graph of the NFA is shown below.

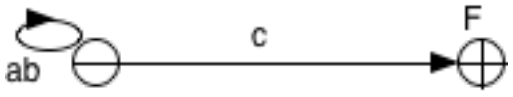


It can be seen that A is quite irrelevant, since it cannot be reached fro the initial state. We might have already spotted this if we examined the set of productions. Even if we did not notice the irrelevance of A at either of those stages, the state would be eliminated at the next stage when we converted the NFA to a DFA.

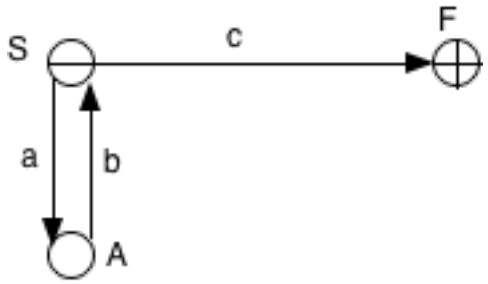
Example 3

The process of constructing an NFA from grammar, and the reverse process, can sometimes be used to find an alternative grammar with shorter (though usually more) productions. Here is a simple example:

Let G be the grammar defined by $N = \{S\}$, $\Sigma = \{a,b,c\}$ and the productions $S \Rightarrow abS \mid c$. It is easy to see that the language generated by G consists of strings of the form $(ab)^n c$, where $n \geq 0$. It consists of $c, abc, ababc, abababc$, and so on. When we design an NFA that recognises this language we get a diagram as shown below.



This is a generalised transition graph. To start converting this to a DFA we need to add another vertex so as to construct a transition graph with edges labelled only with *a, b, c* or ε. This is shown below.



We will not continue the conversion process but instead note that diagram shows us that the grammar G could have equivalently been defined by the alternative set of productions $\{ \quad S \Rightarrow aA \mid c, \quad A \Rightarrow bS \quad \}$.

END OF CHAPTER FIVE

6. HAMILTON PATHS AND CIRCUITS

Most of this chapter will rely on a little knowledge of Graph Theory. However, for those who haven't met graph theory before and as a small reminder for those who have, we give some of the basic definitions.

6.1 Definitions and Examples

92 Definition. *1. A (**simple**) **graph**, G consists of a finite non-empty set V, and a set E of two element subsets of V. The set V is called the **vertex set** of G, E is called the **edge set** of G, and we write $G = (V,E)$ to denote the graph with the vertex set V and the edge set E.*

*2. A **directed graph** or **digraph** G consists of a finite non-empty set V together with a subset A of the Cartesian product $V \times V$. We call V the **vertex set** of G, and A the **arc set** of G. We write $G = (V,A)$ to denote the digraph G with arc set A.*

Example

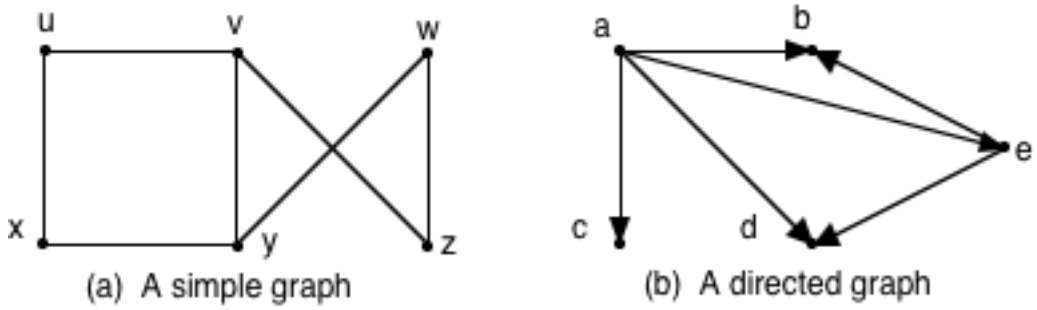


Figure 6.1: Examples.

In graph (a) $G = (V,E)$ with $V = \{u,v,w,x,y,z\}$,

$$E = \{\{u,v\},\{u,x\},\{v,y\},\{x,y\},\{v,z\},\{w,y\},\{w,z\}\}.$$

In graph (b) $G = (V,A)$ with $V = \{a,b,c,d,e\}$, $A = \{(a,b),(a,c),(a,d),(b,e),(e,d)\}$.

93 Definition. *Let u and w be vertices of a graph G. Then a **walk of length** k joining u to w is a sequence of vertices*

$$u = u_0, u_1, u_2, \ldots, u_{k-1}, u_k = w$$

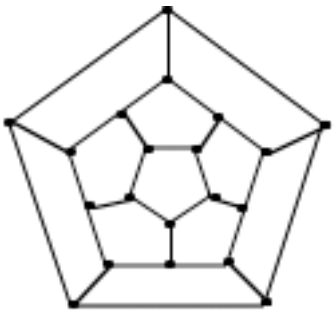
*where $\{u_{i-1}, u_i\}$ is an edge of the graph for $i = 1,2,\ldots,k$. If $u = w$ the path is called a **cycle**. If there are no repeated vertices the walk is called a **simple path**. If the only repeated vertices are the first and the last, the walk is called a **simple circuit.***

This is enough for now to enable us to proceed with the topic, Hamiltonian Paths and Circuits.

94 Definition. *Let G be a graph. A simple path of G that includes every vertex is called a **Hamilton path**, and a simple circuit that includes every vertex is called a **Hamilton circuit**. A graph that has a Hamilton path is called **traceable**, while a graph that has a Hamilton circuit is called **Hamiltonian**.*

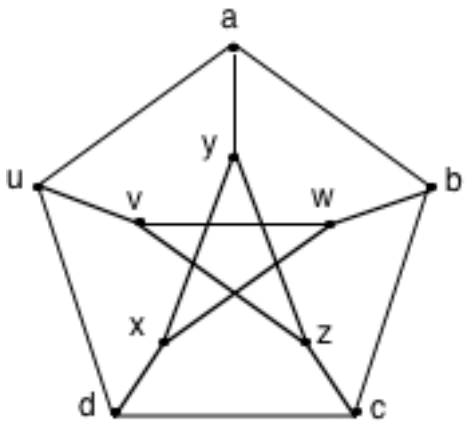
This concept is named after the famous Irish mathematician, Sir William Rowan Hamilton (1805 – 1865). He invented and tried to market a game in which the player was to traverse a route passing through twenty cities, each city only once, and arriving back at the starting point. He made use of a solid wooden dodecahedron, with its twenty vertices labelled with the important cities of the day. To keep track of which cities had been visited, the player was given twenty pegs and a long piece of string. The pegs were inserted at the vertices of the dodecahedron and the string was used to connect the pegs. The puzzle was not commercially successful, but it was a forerunner of such puzzles as Rubik's Cube. Perhaps the puzzle was too easy to solve.

We can draw the faces of a dodecahedron in the plane, and then attempt to find a Hamilton circuit.



Graph of the Dodecahedron

The graph of the dodecahedron is Hamiltonian. It is a good exercise to find a Hamilton circuit. It is not too difficult to do this, although it is not obvious. But not all graphs are Hamiltonian. The graph shown below, known as Petersen's graph, is traceable but not Hamiltonian. Proving that it is traceable is easy; we just need to exhibit one Hamilton path, such as a,b,c,d,u,v,w,x,y,z. However, proving that Petersen's graph is not Hamiltonian is quite difficult.



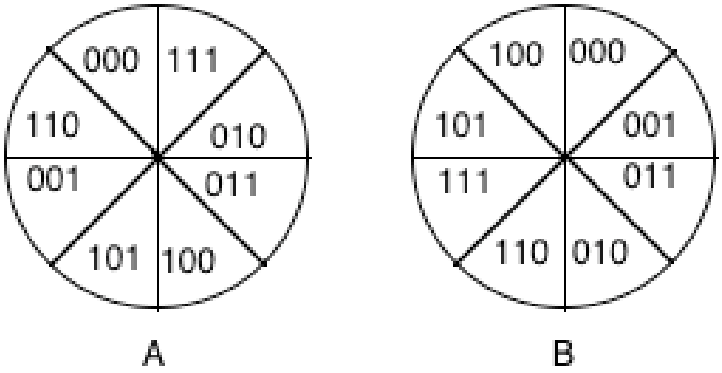
Petersen' s Graph

Unlike Euler paths and circuits, there is no known efficient algorithm for finding Hamilton circuits, or even for showing that they exist. One could list the $n!$ arrangements of the n vertices of a graph and check each one to see if it provides a Hamilton circuis. But of course, this is not an efficient way of solving the problem.
Hamilton circuits arise in routing problems. A salesperson, who must visit various cities by train, might try to find a Hamilton circuit so that all cities are visited one, with the home base as the star and final position.

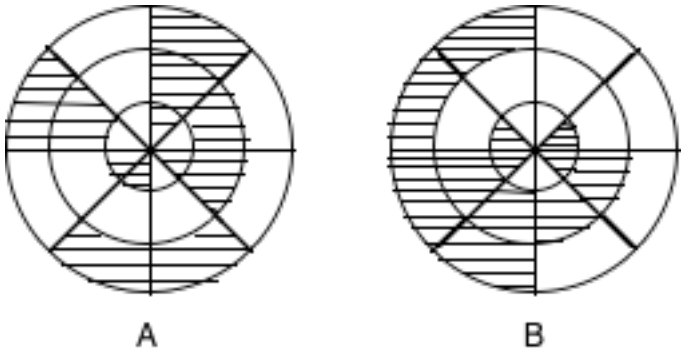
Gray Codes

There are several instances in which it is necessary to list all n-bit strings (a sequence of n symbols, each being a zero or 1) in such a way that each n-bit string differs from the preceding one in exactly one position, and the last n-bit string differs from the first in exactly one position. This kind of listing is called a **Gray code**. For $n = 2$, the listing 00, 01, 11, 10 is a Gray code, but 00, 01, 10, 11 is not a Gray code. Strings 2 and 3 differ in both digits, and strings 1 and 4 differ in both digits.
One way a Gray code is used is in determining the position of a circular disc after it stops rotating. In this situation a circular disc is divided into 2^n equal sectors and

an n-bit string is assigned to each sector. The diagram below shows two different assignments of the 3-bit strings to a disc divided into $2^3 = 8$ sectors.



To determine which n-bit string is to be assigned, the circular disc is divided into n circular rings. This means that each sector is subdivided into n parts, each of which is treated in one of two ways: as for example, opaque or translucent. Under the rotating disc are placed n electrical devices, such as photoelectric cells, that can determine what type of treated material is above it. The diagram below illustrates how this can be done for the assignments shown above. The electrical device below a shaded area will send a 1, and the device below an unshaded area will send a 0. The digits are read from the outside circle to the inside.



In the diagrams below, a pointer has been used to indicate where the rotating disc has stopped with respect to the indicator lights. In the first case the pointer is entirely within one sector, so the device can send the signal 001. In the second case, because of the position of the pointer there is the possibility of an incorrect signal being sent. The signal device could send either 110 or 001. This means there could be an error in every position.

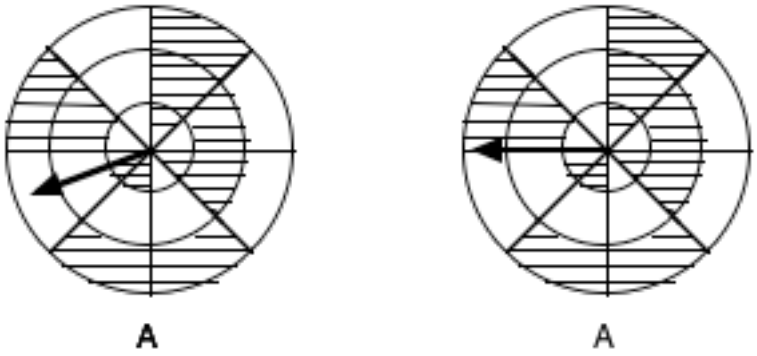


Figure 6.2: Diagram A with a pointer in two different configurations.

We consider a similar situation for the diagram *B*. In the first case again the pointer is entirely within one sector, so the measuring device can send the signal 001. In the second case the pointer is on the border of sectors, so again there is an ambiguity about which signal should be sent. The signal may be 000 or 001. The difference here is that these signals only differ in one digit. What is more important is that wherever the pointer stops, if it is on the border of two sectors, there will be only two possible signals that may sent. Further these signals will differ in only one digit and these two strings will identify the two adjacent sectors where the pointer has stopped.

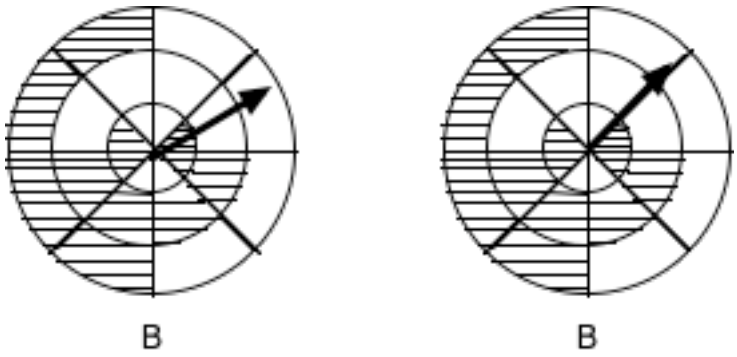
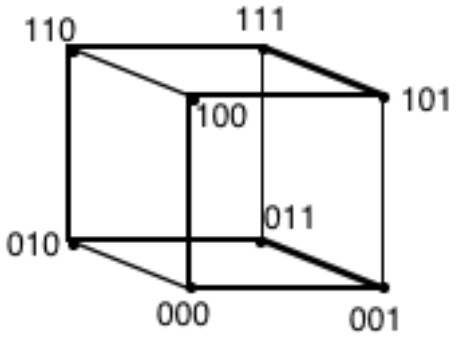


Figure 6.3: Diagram B with a pointer in two different configurations.

Case *B* minimises any error that may occur because the pointer stops at the border of two segments. If you look at the labelling for case *B* you will notice that the labelling on any two adjacent sectors differs in only one digit. The labelling is derived from a Gray code.

To find a Gray code, we construct a graph using 2^n vertices representing the 2^n possible n-bit strings. Two vertices are connected with an edge if the corresponding n-bits differ in exactly one position. It can be proved that a graph constructed in this way always has a Hamilton circuit, and so this can be used to find a Gray code for any n. The diagram below illustrates the case for $n = 3$. The darker lines indicate the Hamiltonian cycle.



The Gray code associated with this graph is 000, 001, 011, 010 110, 111, 101, 100. It is not the only possibility.
For further information and reading consult the web at

http://www.cs.bham.ac.uk/Mirrors/ftp.de.uu.net/EC/clife/www/Q21.htm
http://www.ils.unc.edu/~losee/gray1.html
http://web.usna.navy.mil/~wdj/gray.htm

6.2 Necessary and Sufficient Conditions

It was stated above that the Petersen graph is not Hamiltonian. Yet this is difficult to prove. In this section we look for ways of proving that a graph is or is not Hamiltonian. Of course, if we can find a Hamilton circuit, the graph must be Hamiltonian. But it is often good to be able to decide, without actually finding a circuit, that a graph is Hamiltonian.

Necessary Conditions

The next theorem is little more than an observation that comes from the definition.

95 Theorem. *If a graph G is Hamiltonian, then G has a subgraph H with the following properties:*

1. *H contains every vertex of G,*
2. *H is connected* (i.e. *there is a walk between every pair of vertices),*
3. *H has the same number of edges as vertices, and*
4. *every vertex of H has degree 2* (i.e. *belongs to exactly two edges).*

The diagrams below give some indication of the basis for this theorem. However, they do not prove theorem.

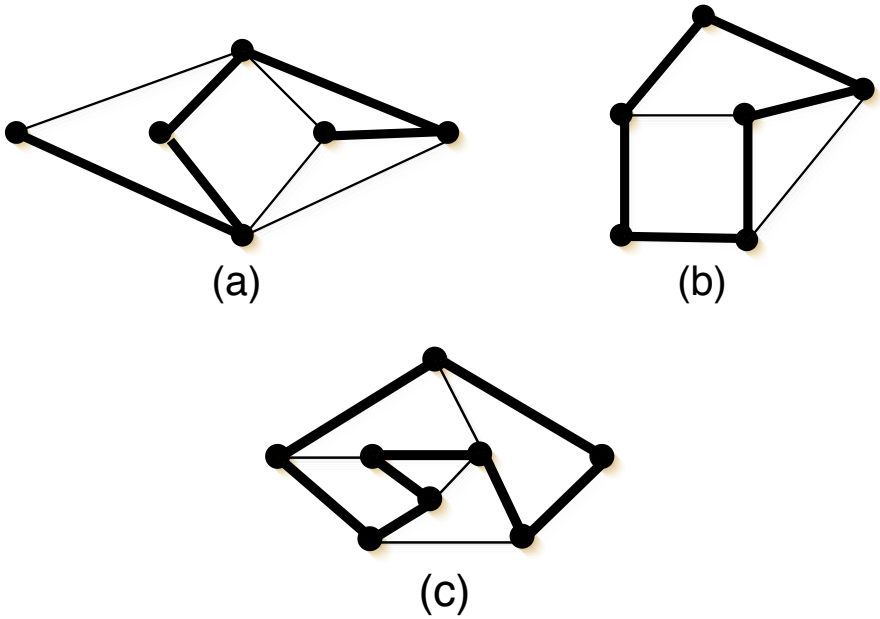


Figure 6.4: Some examples.

The diagram (a) does not have a Hamilton circuit, although it does contain a Hamilton path. The diagrams (b) and (c) both have Hamilton circuits. These are shown by the darker lines, which, together with the vertices, form a subgraph of the original graph. This subgraph satisfies the properties described in the theorem.

Theorem 95 allows us to show that the graph G shown below does not have a Hamiltonian circuit.

We can argue as follows: If G has a Hamiltonian circuit, then by Theorem 95, G has a subgraph H that (i) contains every vertex of G, (ii) is connected, (iii) has the same

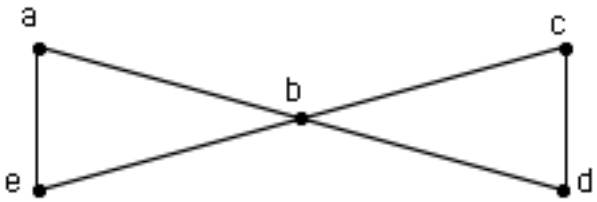


Figure 6.5: A graph that is not Hamiltonian.

number of edges as vertices, and (iv) is such that every vertex has degree 2. Suppose that such a subgraph H exists. This means that there is a connected subgraph H of G that has 5 vertices $\{a,b,c,d,e\}$ and 5 edges, and every vertex of H has degree 2. Since the degree of vertex b in G is 4, two edges incident on b must be removed from G to make H. Edge $\{a,b\}$ cannot be removed because if it were vertex a would have degree less than 2 in H. A similar argument shows that the edges $\{b,c\},\{b,d\}$, and $\{b,e\}$ cannot be removed. It follows that the degree of vertex b in H must be 4. This contradicts the condition that every vertex in H must have degree 2. We conclude that no subgraph H exists, so that G does not have a Hamiltonian circuit.

We continue to find some further conditions that are necessary if a graph is Hamiltonian. Following from Theorem 95 a Hamiltonian graph must be connected, and it must have order at least 3 (that is, it must have at least 3 vertices).

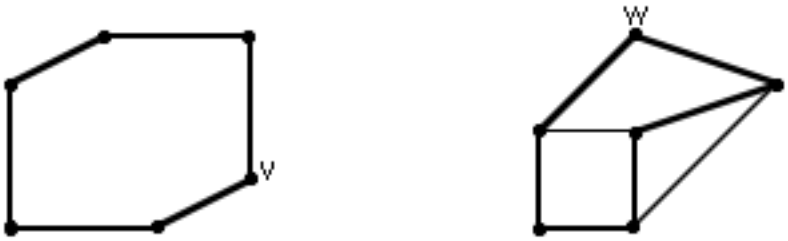
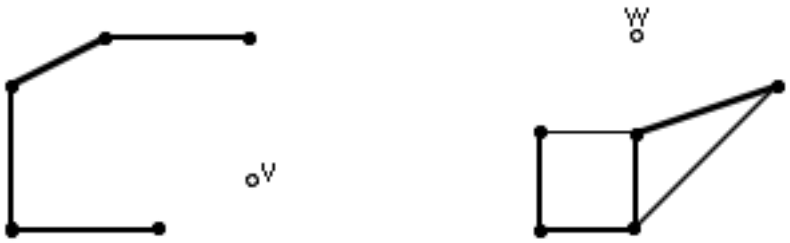


Figure 6.6: Two Hamiltonian graphs.

Each of the graphs shown above has a Hamilton circuit, which is indicated by the darker lines. In the first graph let us remove vertex v and the edges connected to it. In the second graph we remove vertex w and the edges connected to it. The results are shown below.



We see that the resulting graphs are still connected, and they still contain at least one Hamilton path. In fact, if we remove a vertex, and the edges connected to it, from any Hamiltonian graph, the remaining graph will always contain a Hamilton path, and will continue to be connected.

96 Definition. *A graph $G = (V,E)$ of order at least three is called **2-connected** (or doubly connected) provided $G \setminus v$ is connected for every vertex v in V.*

The graph that we considered earlier, and is shown again below is not 2-connected.

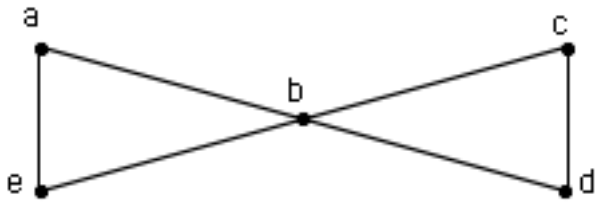


Figure 6.7: A non-Hamiltonian graph.

It is easy to see that if we remove vertex b, the graph $G \setminus b$ is not connected:

Figure 6.8: $G \setminus b$ is not connected.

97 Lemma. *If a graph G is Hamiltonian, then it is 2-connected.*

Suppose that $G = (V,E)$ is a Hamiltonian graph where $|V| = n$. Suppose the vertices of G are $v_1, v_2, \ldots, v_n$. Let $H = (V,E_1)$ be the subgraph of G induced by some Hamilton circuit. Then from Lemma 97 $H \setminus v_1$ is connected. Further, $(H \setminus v_1) \setminus v_2$ can have

at most two components. (A 'component' is a maximal connected subgraph; *i.e.* a connected subgraph that contains all edges and vertices of the parent that are connected to any of the members of the subgraph.) Similarly $((H \setminus v_1) \setminus v_2) \setminus v_3$ can have at most three components. In general, if m vertices $(m<n)$ are removed from H the resulting graph can have at most m components. This leads to the following necessary condition for a Hamiltonian graph.

98 Theorem. *Let $G = (V,E)$ be a graph of order at least three. If G is Hamiltonian, then for every proper subset U of the vertices, V, the subgraph of G induced by $V \setminus U$ has at most $|U|$ components.*

Example

Consider G, the graph shown below.

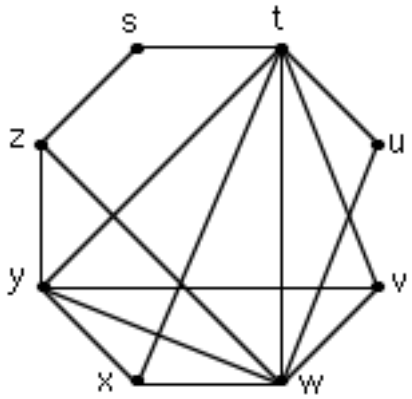


Figure 6.9: Does this graph have a Hamilton circuit?

We want to be able to decide whether or not this graph is Hamiltonian. Just looking at the graph gives no clear answer, as it is not possible to quickly find a Hamilton circuit. But this isn't enough evidence on which to make a decision. Let $U = \{t,w,y\}$. We will use Theorem 98. Here $|U| = 3$, so by the theorem, if G is Hamiltonian then the subgraph of G induced by $V \setminus U$ can have at most three components. The subgraph of G induced by $V \setminus U$ is shown below.

The remaining graph has four components: vertices s and t and the edge joining them make one component, and each of the vertices u, v and x make a component. (The other three vertices are shown as 'ghosts'.) Since there are more than three components, the graph is not Hamiltonian.

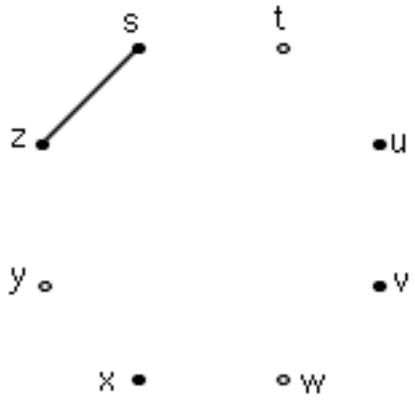


Figure 6.10: $G \setminus U$.

We need to be careful using a theorem like Theorem 98. It tells us a condition that a graph must have if it is Hamiltonian. This is called a "necessary condition". If a graph doesn't have a necessary condition, it can't be Hamiltonian. But a necessary condition can not be used to prove that a graph is Hamiltonian. We now look for some "sufficient conditions". These are conditions, which if satisfied by a graph will enable us to say, "Yes, that graph is Hamiltonian".

Sufficient Conditions

The first conditions we consider ensure that a graph has a large enough number of edges to form the sort of circuit we require.

99 Theorem. *Let G be a graph of order $n \geq 3$. If for every pair of distinct non-adjacent vertices u and v,*

$$deg(u) + deg(v) \geq n,$$

then G is Hamiltonian.

We can use this theorem to show that the graph below has a Hamilton circuit.

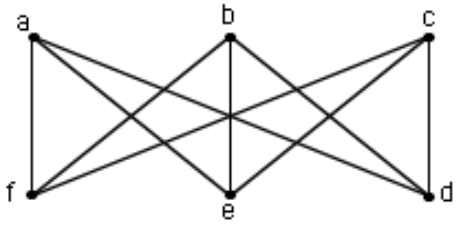
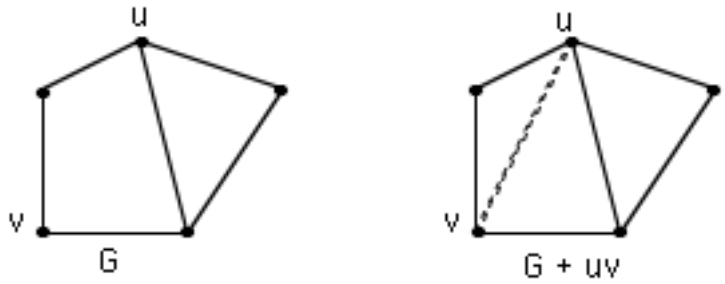


Figure 6.11: A Hamiltonian graph.

This graph has 6 vertices. The pairs of non-adjacent vertices are $\{a,b\},\{a,c\},\{b,c\},\{d,e\},\{d,f\}$, and $\{e,f\}$. Each vertex has degree 3 so the sum of the vertex degrees for each pair is 6. Since 6 is equal to the number of vertices of the graph, Theorem 99 allows us to conclude that the graph is Hamiltonian. We do note that the theorem doesn't actually help us to find a Hamilton circuit. It just gives us the confidence to know that we will be able to find one.

When considering the proof of this theorem, Bondy and Chvátal observed that they could use it to prove a much stronger statement.

Before stating their theorem we need to introduce some new notation. If u and v are non-adjacent vertices of a graph G, then $G+uv$ denotes the graph obtained from G by adding the edge uv.



100 Theorem. *Let G be a graph of order $n \geq 3$. Suppose that u and v are distinct non-adjacent vertices of G such that*

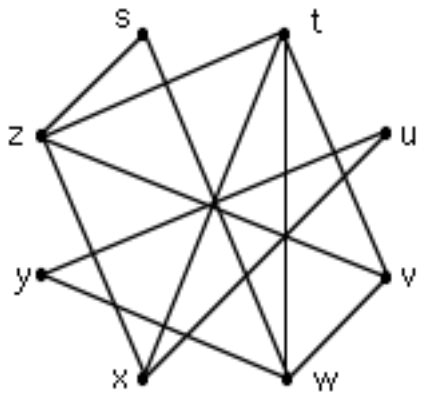
$$deg(u)+deg(v) \geq n.$$

Then G is Hamiltonian if and only if $G+uv$ is Hamiltonian.

In proving this theorem, it is clear that if the graph G is Hamiltonian, then the graph $G+uv$ is Hamiltonian. This means that the real burden of the proof is to show that if, under the conditions of the theorem, $G+uv$ is Hamiltonian, then G must be Hamiltonian. The proof proceeds by contradiction. It assumes that $G+uv$ is Hamiltonian, and that G is not Hamiltonian. It is then able to arrive at a contradiction concerning the degrees of u and v.

Example

Consider the graph G shown below.



We are interested in deciding whether or not this graph is Hamiltonian. Let us call this graph $G_0 = (V, E_0)$. Note that G_0 has order 8 and that w and z are non-adjacent vertices with the property that

$$\deg(w) + \deg(z) = 8.$$

So we can use Theorem 100 to say that G_0 is Hamiltonian if and only if $G_1 = G_0 + wz$ is Hamiltonian.

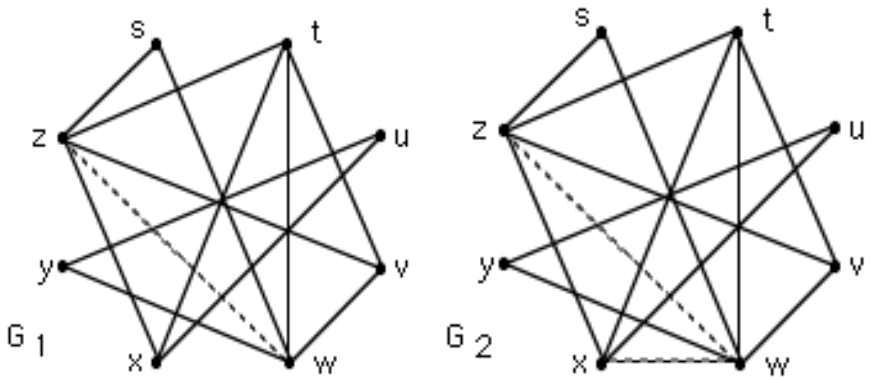


Figure 6.12: The graphs G_1 and G_2.

Now consider G_1. In this graph, vertices w and x are non-adjacent vertices with $\deg(w) + \deg(x) = 8$. So again, using Theorem 100 we conclude that G_1 is Hamiltonian if and only if $G_2 = G_1 + wx$ is Hamiltonian. We can continue to use this procedure, obtaining a sequence of graphs, G_0, G_1, $G_2, \ldots, G_{15}$, and the final graph G_{15} is isomorphic to K_8, the complete graph on 8 vertices. Each of G_1, G_2, $\ldots$, G_{15} is obtained from its predecessor by the addition of a single edge, and each of G_0, G_1,

$G_2, \ldots, G_{15}$ is Hamiltonian provided its successor is Hamiltonian. Thus G_0 is Hamiltonian if and only if G_{15} is Hamiltonian. G_{15} is Hamiltonian because it is a complete graph, and any complete graph is Hamiltonian. We conclude that G_0 is Hamiltonian.

Closure

The process we have just used may have a more general application. Because of this we focus on the process which is important in itself. Let us consider any graph $G = G_0$ with n vertices. If we can find non-adjacent vertices u_0 and v_0 such that

$$\deg(u_0, G_0) + \deg(v_0, G_0) \geq n,$$

then let us make $G_1 = G_0 + u_0v_0$.
Now consider G_1 and look for non-adjacent vertices u_1 and v_1 with the property that

$$\deg(u_1, G_1) + \deg(v_1, G_1) \geq n.$$

If we can find such vertices we make a new graph $G_2 = G_1 + u_1v_1$. We continue this process for as long as we can, making a sequence of graphs $G = G_0$, G_1, $G_2, \ldots, G_k$, $k \geq 0$, where G_k does not contain two non-adjacent vertices whose degree sum in G_k is at least n. Then G is Hamiltonian if and only if G_k is Hamiltonian. If G_k is a complete graph then both G and G_k are Hamiltonian. Bondy and Chvátal called G_k the "closure" of G. This leads to the following definition.

101 Definition. *Given a graph G of order n, the **closure of** G is the graph $Cl(G)$ defined recursively as follows:*

> *If G contains two distinct non-adjacent vertices u and v such that $deg(u) + deg(v) \geq n$, then $Cl(G) = Cl(G + uv)$; otherwise $Cl(G) = G$.*

It is possible that the definition of the closure of a graph G is not well defined, since it gives no indication of which vertices to choose if there are more than one pair of non-adjacent vertices u and v with the property that $\deg(u) + \deg(v) \geq n$. However, it can be shown that the closure operation is independent of any possible choices we may make, and it is therefore a well defined process.

Examples of the closure operation.

Below we find the closure of some graphs. In the final case it becomes quite difficult to show all the edges, one from each vertex to every other vertex.

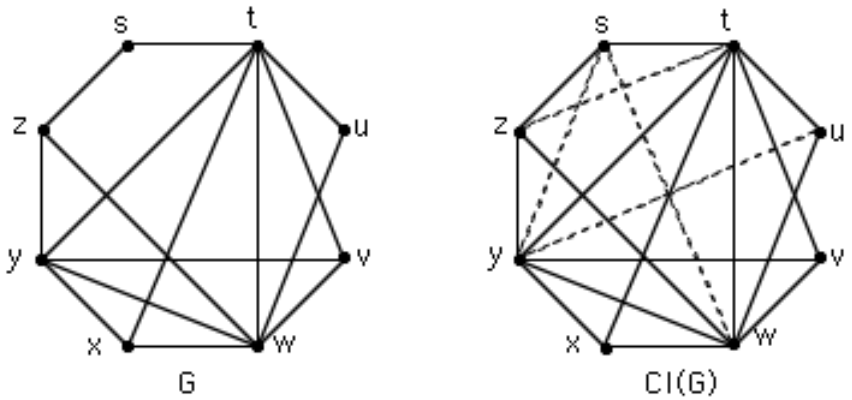


Figure 6.13: The closure is bigger than the graph, but not the complete graph.

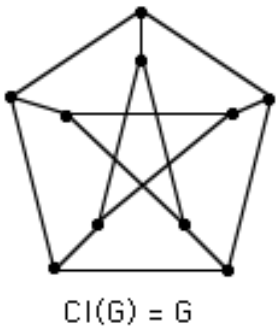


Figure 6.14: The closure is the same as the graph.

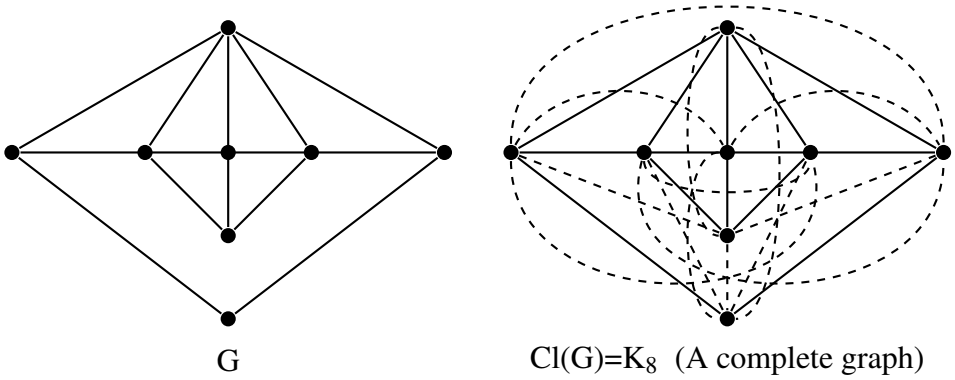


Figure 6.15: The closure is a complete graph.

The next result follows immediately from Theorem 100.

102 Corollary. *Let G be a graph of order $n \geq 3$. Then G is Hamiltonian if and only if $Cl(G)$ is Hamiltonian.*

The final theorem of this section presents a series of sufficient conditions for a graph to be Hamiltonian. They are listed so that the corresponding results are in order of decreasing strength, in the sense that each condition from the second on implies the condition preceding it.

103 Theorem. *Let G be a graph of order $n \geq 3$, and let $d_1, d_2, \ldots, d_n$ be the degrees of the vertices of G, listed so that $d_1 \leq d_2 \leq \cdots \leq d_n$. If G satisfies any of the following conditions, then G is Hamiltonian.*

(a) (Bondy and Chvátal, 1976) $Cl(G) \cong K_n$.

(b) (M. Las Vergnas, 1971) There exists a bijection $f : V \to \{1,2,\ldots,n\}$ such that, if u and v are distinct non-adjacent vertices with $f(u) < f(v)$, $f(u) + f(v) \geq n$, $deg(u) < f(u)$, and $deg(v) < f(v)$, then $deg(u) + deg(v) \geq n$.

(c) (Chvátal, 1972) For each i with $2 \leq 2i < n$, if $d_i \leq i$, then $d_{n-i} \geq n - i$.

(d) (Bondy, 1969) For all i, j with $1 \leq i < j \leq n$, if $d_i \leq i$, and $d_j \leq j$, then $d_i + d_j \geq n$.

(e) (Pósa, 1962) For each i with $2i < n$, $|\{v : deg(v) \leq i\}| < i$.

(f) (Ore, 1962) For each pair of distinct non-adjacent vertices u and v, $deg(u) + deg(v) \geq n$.

(g) (Dirac, 1952) For every vertex v, $2deg(v) \geq n$.

Some of these conditions seem fairly complicated and show the difficulty that has been experienced in finding suitable conditions to show that a graph is Hamiltonian. The problem is still not solved, and there is no condition, or set of conditions, that says a graph is Hamiltonian if and only if the conditions are satisfied.

6.3 The Travelling Salesperson Problem

This problem is a natural extension of the problem of finding a Hamilton circuit in a graph, and it is one that has long been discussed with great interest. Let $G = (V, E, w)$ be a complete graph of n vertices, where w is a function from E, the set of edges, to the set of positive real numbers in such a way that for any three vertices i, j and k in V,

$$w(i,j)+w(j,k) \geq w(i,k).$$

This inequality is known as the triangle inequality, and would certainly be true for places on a map. This is important as the problem is modelling a situation where we may consider places on a map. We refer to $w(i,j)$ as the length of the edge $\{i,j\}$. The travelling salesperson problem asks for a Hamilton circuit of minimum length. The length of a circuit is defined as the sum of the lengths of the edges in the circuit.

A physical interpretation of this formulation is fairly obvious. We consider the graph G as a map of n cities where $w(i,j)$ is the distance between cities i and j. A salesperson wants to make a tour of the cities which starts and ends at the same city and includes visiting each of the remaining $n-1$ cities once and only once. Further, an itinerary that has a minimum total distance is desired.

The travelling salesperson problem is a difficult one in that there is no known "efficient" algorithm to solve the problem. However, we look for simple procedures that will give good results for the problem. We first illustrate a procedure known as the nearest neighbour method. It gives reasonably good results.

A particular problem might look something like the following graph:

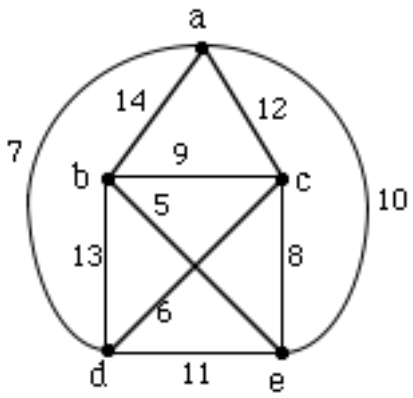


Figure 6.16: A Travelling Salesperson Problem.

The Nearest Neighbour Method

The **nearest neighbour method** for solving the problem goes as follows:

1. Choose any vertex v and find the vertex that is closest to v to form an initial path of one edge. We enlarge this path in a vertex-by-vertex fashion as described in the next step.

2. Let x denote the latest vertex that was added to the path. Among all the vertices that are not in the path, pick the one that is closest to x, and add this vertex to the path by adding the edge connecting x to this vertex. Repeat this step until all vertices in G are included in the path.

3. Form a circuit by adding the edge connecting the starting vertex and the last vertex added.

We illustrate this process with the graph given in Figure 6.16. Suppose that we start at vertex a. Its nearest neighbour is d, so construct a path from a to d. The nearest neighbour to d is c, so our path now goes from a to d to c . The vertex-by vertex construction goes as follows:

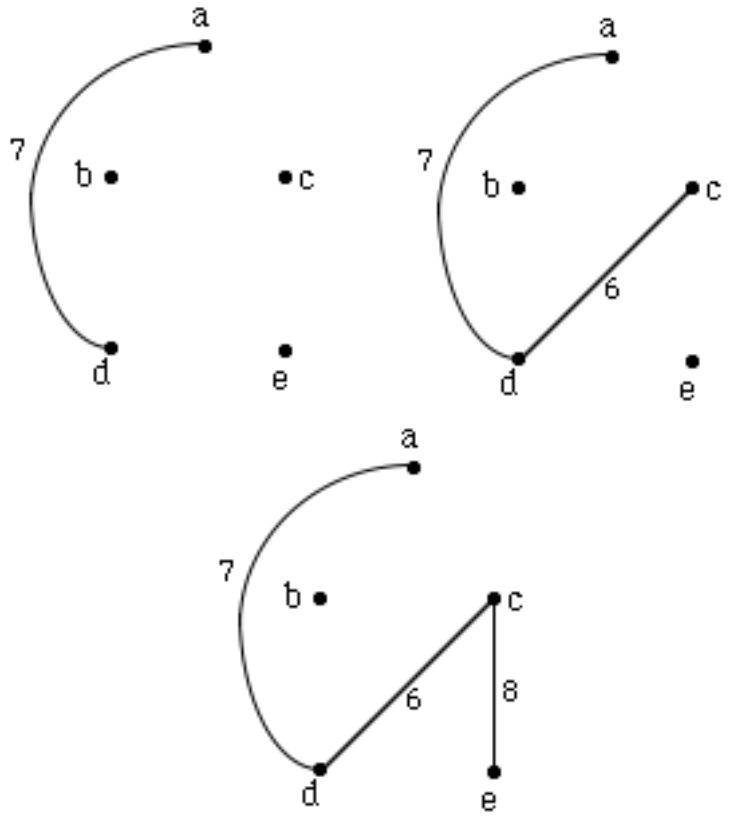


Figure 6.17: The first three steps.

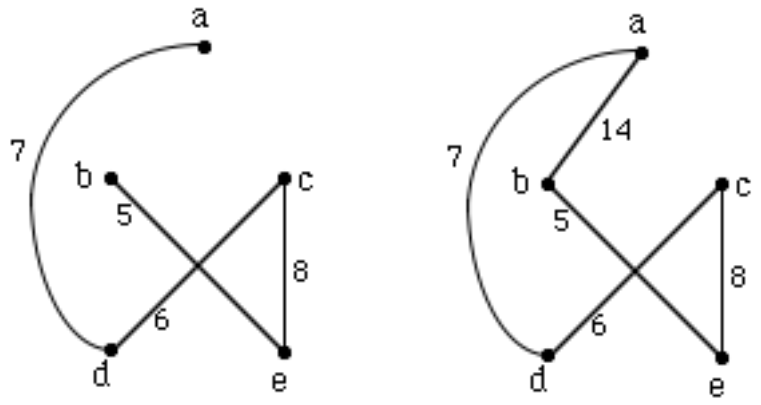


Figure 6.18: The final two steps.

The total distance in this circuit is 40, and this is in fact not the minimal Hamilton circuit as there is one of length 37 as shown in Figure 6.19.

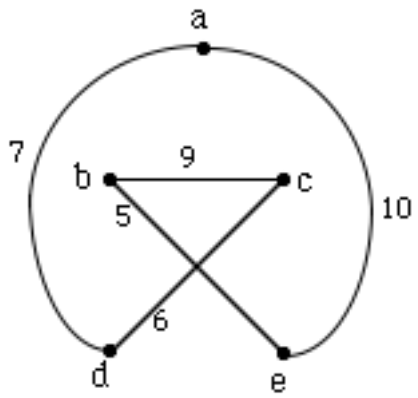


Figure 6.19: A shorter circuit.

Starting at a different vertex may influence the circuit produced by the nearest neighbour algorithm. For example, for the graph of Figure 6.16, starting at vertex *e* produces the shorter circuit above (Figure6.19).

The Closest Insertion Method

This method differs from the last in that it builds up longer and longer "mini-tours" of towns (vertices) until a Hamilton circuit is achieved. It begins at town *a*. This means that the first tour is trivial - it comprises simply a tour in town *a* , going nowhere else.

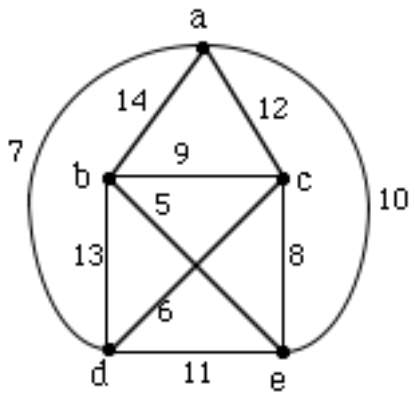


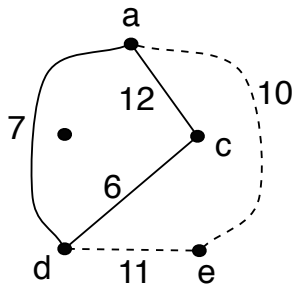
Figure 6.20: Revisiting the same example.

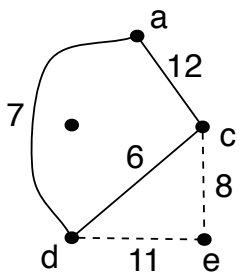
At the next stage the town closest to a is identified as town d. The first real tour, $\langle a,d,a\rangle$, is created, and the total distance is **14**. This is not a feasible solution. However, successive tours are built up in this way, each with one more vertex than the previous tour.

In the general step we ask what is the town not yet on the tour, which is closest to a town on the tour? We then insert this town on the tour, taking as little extra distance as possible. Let us illustrate this with the example.

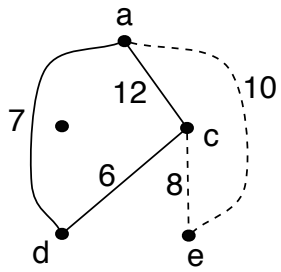
We ask which vertex is closest to a or d. The answer is c at distance 6. To insert c into the tour we can go from a to d to c to a. This adds the distances 6 and 12 and removes the distance from d to a which is 7. So the change in distance is $6+12-7=11$ and the tour $\langle a,d,c,a\rangle$ has distance **25**.

Next, notice that e is the vertex closest to a,d or c. We can insert e between a and d to get $a,\boldsymbol{e},d,c,a$, between d and c to get: $a,d,\boldsymbol{e},c,a$ or between c and a to get: $a,d,c,\boldsymbol{e},a$. In the first case we would add ae and ed and remove ad, which would make a total difference of $10+11-7=14$. In the second case we would add de and ec and remove dc. This would make a total difference of $11+8-6=13$.

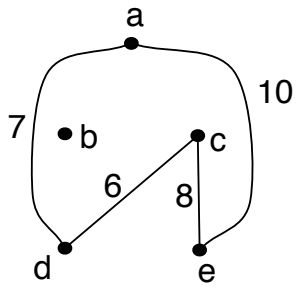




In the third case we would add ce and ea and remove ca. This would make a total difference of $8+10-12=6$.

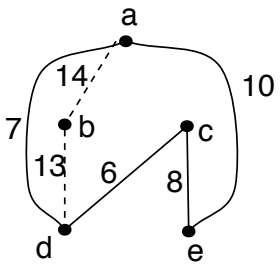


We choose the third option as it increases the tour by the least amount. The tour now becomes $\langle a,d,c,e,a\rangle$ which has distance **31**.

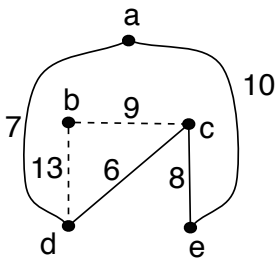


The only vertex left to add to the tour is b and we need to decide where to add it. The possibilities are (i) *a, **b**, d, c, e, a*, (ii) *a, d, **b**, c, e, a* (iii) a, *d, c, **b**, e, a* or (iv) *a, d, c, e, **b**, a*.

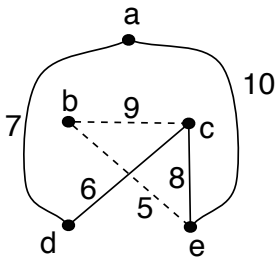
(i) *a, **b**, d, c, e, a* : Here we add ab and bd and remove ad with a total change of $13+14-7=20$.



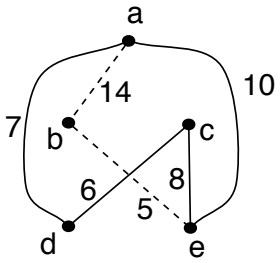
(ii) *a, d,* ***b****, c, e, a* : Here we add *db* and *bc* and remove *dc* with a total change of $13+9-6=16$.



(iii) *a, d, c,* ***b****, e, a* : Here we add *cb* and *be* and remove *ce* with a total change of $9+5-8=6$.



(iv) *a, d, c, e,* ***b,*** *a* : Here we add *eb* and *ba* and remove *ea* with a total change of $5+14-10=9$.



From here the choice is clear and we choose option (iii) which gives a Hamilton circuit $\langle a,d,c,b,e,a\rangle$ of length **37**. This is in fact the minimal Hamilton circuit shown in Figure 6.19.

Comparing Methods: The Tourist Bus Problem

The Scenic valley Tourist Agency has a problem. It has launched a promotion to boost the number of tourists in Scenic Valley. They are about to offer a package tour to tourists who fly in and out of Geyser Town, which is the only airport with connection to towns outside the valley. They now offer a bus tour of the valley, visiting each town once, starting from and returning to GT. In order to estimate the costs of the tour, the agency needs to know the least number of miles the bus must travel in making the tour. We can easily see that this problem is another example of the "Travelling Salesperson Problem", and we will look at the solutions given by the **nearest neighbour method** and the **closest insertion method**. Below is the table containing the distances between each of the towns. We are assuming there are roads between each of the towns.

Table 6.1: Distances Between the Towns

		2	3	4	5	6	7	8	9	10
Geyser Town	1	8	5	9	12	14	12	16	17	22
Lakeside	2		9	15	17	8	11	18	14	22
Mountainview	3			7	9	11	7	12	12	17
Riverview	4				3	17	10	7	15	8
Twin Streams	5					8	10	6	15	15
Snowdonia	6						9	14	8	16
Skier's Paradise	7							8	6	11
Park City	8								11	11
Rolling Downs	9									10
Tumbling Basin	10									

The layout of these towns is shown in Figure 6.21

Figure 6.21: The Towns.

The Nearest Neighbour Solution

We recall that we start at town 1 and travel to the nearest town not yet visited. So we travel fro town 1 to town 3, from 3 to 4, from 4 to 5, from 5 to 8. The closest town to town 8 is 5, but we have already visited 5. The closest town to 8 not visited is 7. From 7 we go to 9, from 9 to 6, from 6 to 2. Now the only town not visited is 10. so we go from 2 to 10 and back to 1. The complete tour is $\langle 1,3,4,5,8,7,9,6,2,10\rangle$. It has a total distance of 95 km and is shown in Figure 6.22.

This example shows what can go wrong with the nearest neighbour method. It seems that town 10 is responsible for the creation of a disastrous tour. The journeys in and out of 10 both cross other segments of the tour. It is well known in the theory of the travelling salesperson problem that an optimal tour does not possess such crossings. We will try the closest insertion method to see if we can find a better solution.

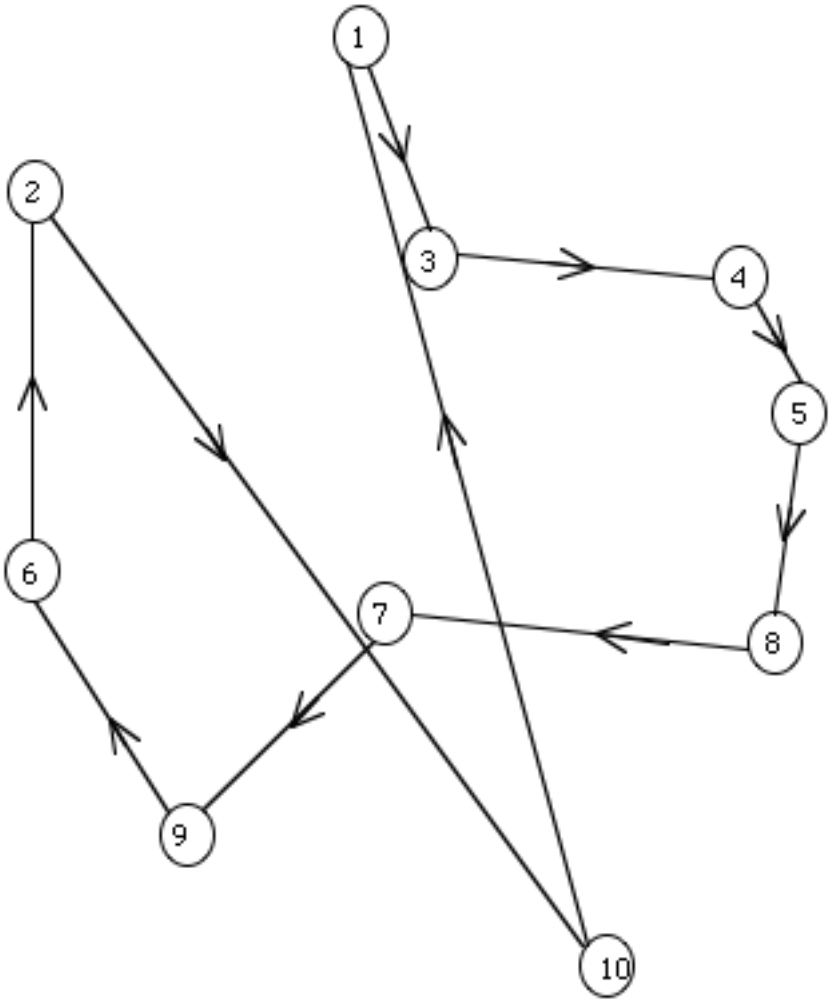


Figure 6.22: The nearest neighbour solution.

The Closest Insertion Solution.

We recall that this method builds up a closed tour of towns, beginning at town 1. The town closest to town 1 is town 3 and the first non-trivial tour is $\langle 1,3,1\rangle$.
We look for a town closest to one already in the tour. We find towns 4 and 7 are both 7 km from town 3. We settle the tie arbitrarily. Let us choose town 4 and produce a new tour: $\langle 1,3,4,1\rangle$.
At the next stage we find that town 5 is the closest to one of the towns already selected (3 km from town 4), but now we have a choice as to where to put town 5 in the tour. We could go 1, **5**, 3, 4, 1, or 1, 3, **5**, 4, 1 or 1, 3, 4, **5**, 1. As town 5 is closest to town 4 it would seem unlikely that the first choice would make the shortest distance, and this is what we discover. The shortest of these tours is $\langle 1,3,5,4,1\rangle$ with distance 26 km.

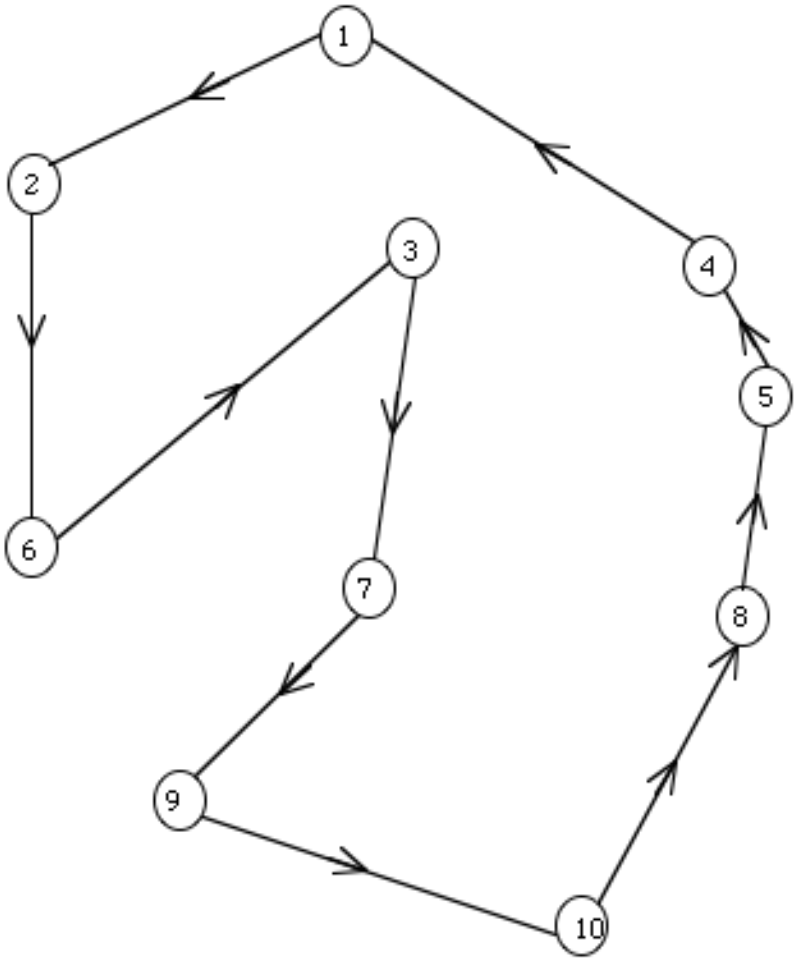


Figure 6.23: The Closest Insertion Method Solution.

We continue in this manner. Choices are settled arbitrarily, but we are careful about where we include the new member of the tour. The final tour is $\langle 1,2,6,3,7,9,10,8,5,4,1\rangle$ with a total distance of 79 km. This is a substantial improvement on the tour produces by the nearest neighbour method. It is shown in Figure 6.23.
This tour does not possess any crossings, but perhaps we can do even better.

The Geometric Method

Consider Figure 6.24 where a boundary, called the *convex hull*, has been drawn around the towns. It is well known that the towns on the boundary appear in the order in which they appear on the convex hull in any optimal solution.

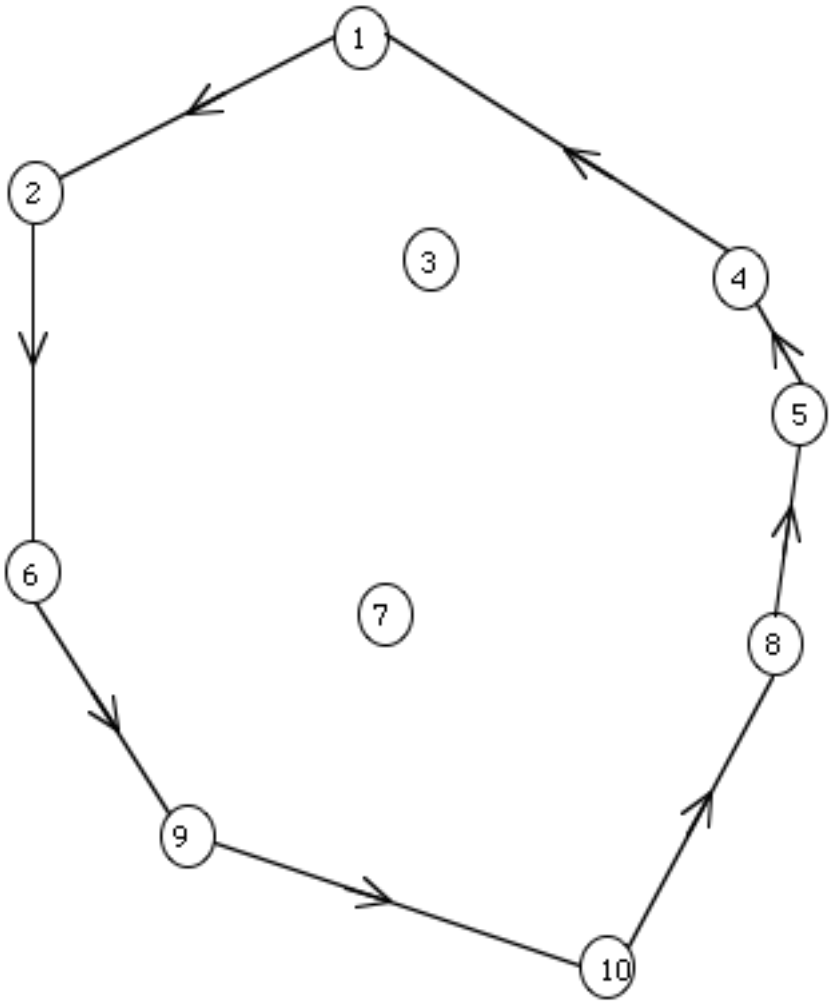


Figure 6.24: The convex hull.

Thus we can consider the partial tour $\langle 1,2,6,9,10,8,5,4,1\rangle$ as being a good starting point to finding a complete tour of relatively low distance. All we have to do is to somehow include the "free" towns 3 and 7. The following method inserts these towns with a view to minimising the extra distance incurred, but in a different manner from the previous method.

A line is drawn from each of these free towns to each of the towns on the present tour of $\langle 1,2,6,9,10,8,5,4,1\rangle$. These lines are shown below in Figure 6.25.

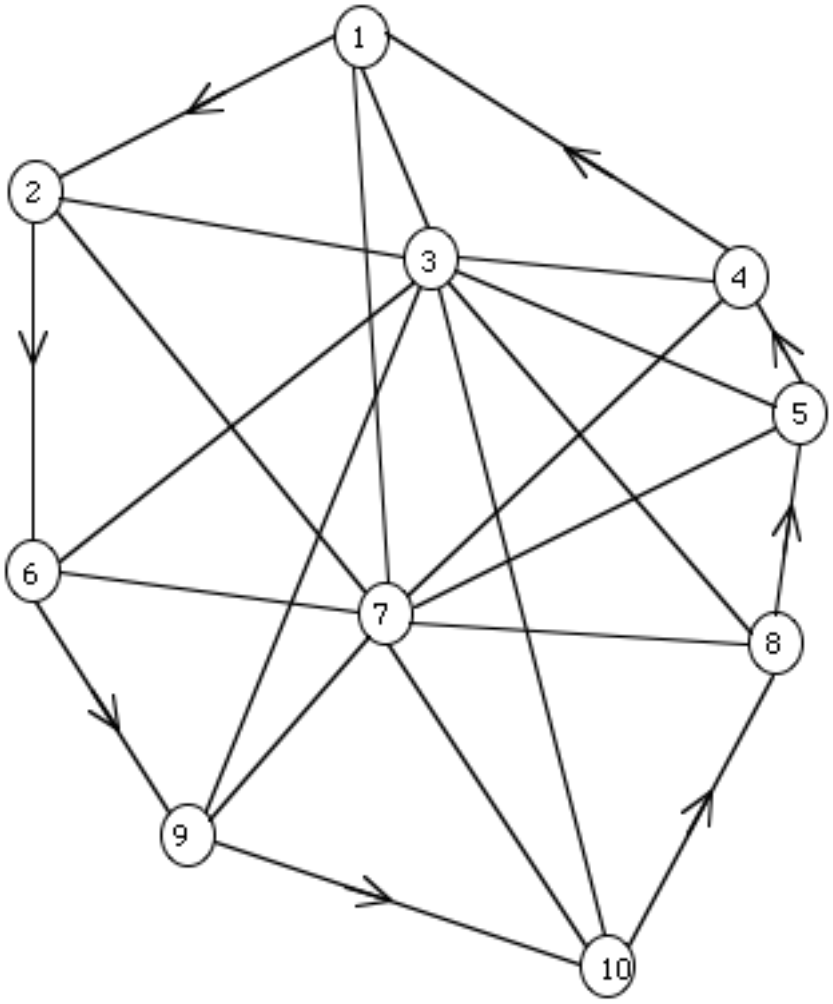


Figure 6.25: The angles subtended at towns 3 and 7.

The next step is to calculate, for each free town, the angle between the successive lines leaving it. The largest of these angles is the one subtended by the line $4 \to 1$. It seems

reasonable that the larger the angle the smaller the distance will be if we insert the town between the two towns at the other end of the lines making the angle.
So as the largest angle is subtended at town 3 it is inserted first. It is inserted in the intuitively obvious place between towns 1 and 4. This creates an enlarged tour $\langle 1,2,6,9,10,8,5,4,3,1\rangle$, which is shown in Figure 6.26.

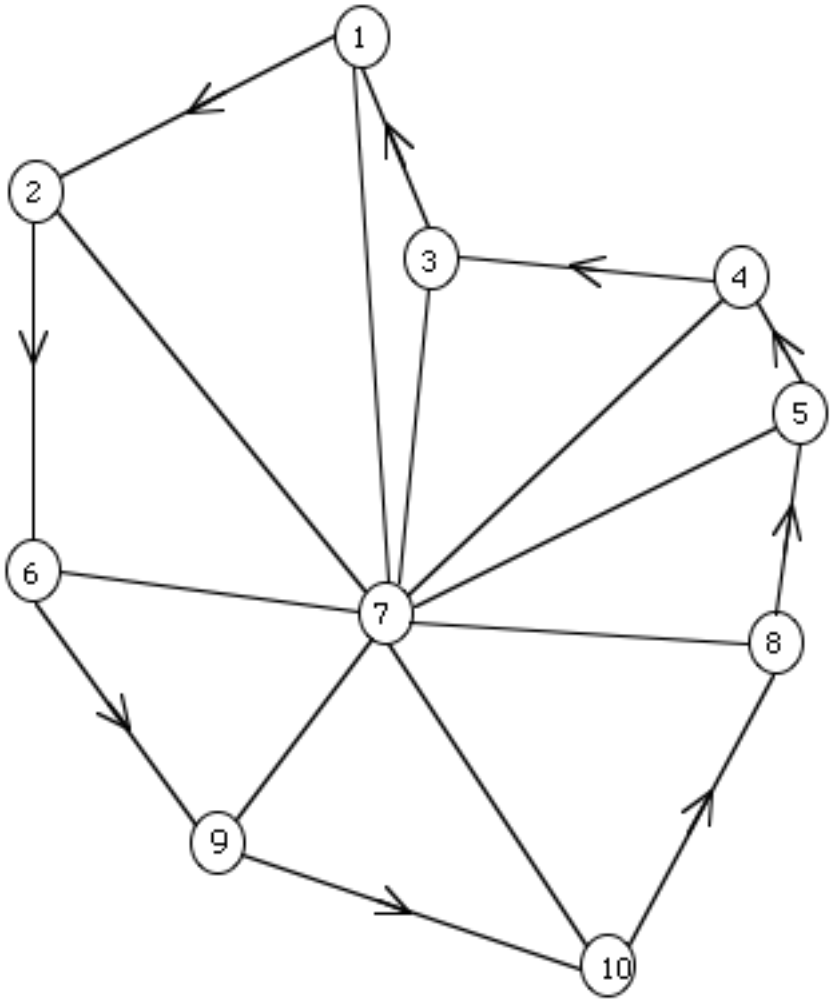


Figure 6.26: An enlarged tour.

The remaining free town 7 is linked by straight lines to the towns already on the tour and the largest angle is chosen. This is the angle subtended by $6 \rightarrow 9$ (in an accurate diagram). So town 7 is inserted in this journey between 6 and 9 and this produces the final tour $\langle 1,2,6,7,9,10,8,5,4,3,1\rangle$. It has length 67 km and is shown in Figure 6.27. It is a further improvement of the solution produced by the nearest neighbour method.

The geometric method does not refer directly to the inter town distance table, but relies on the spatial arrangement of the towns, and an accurate map!

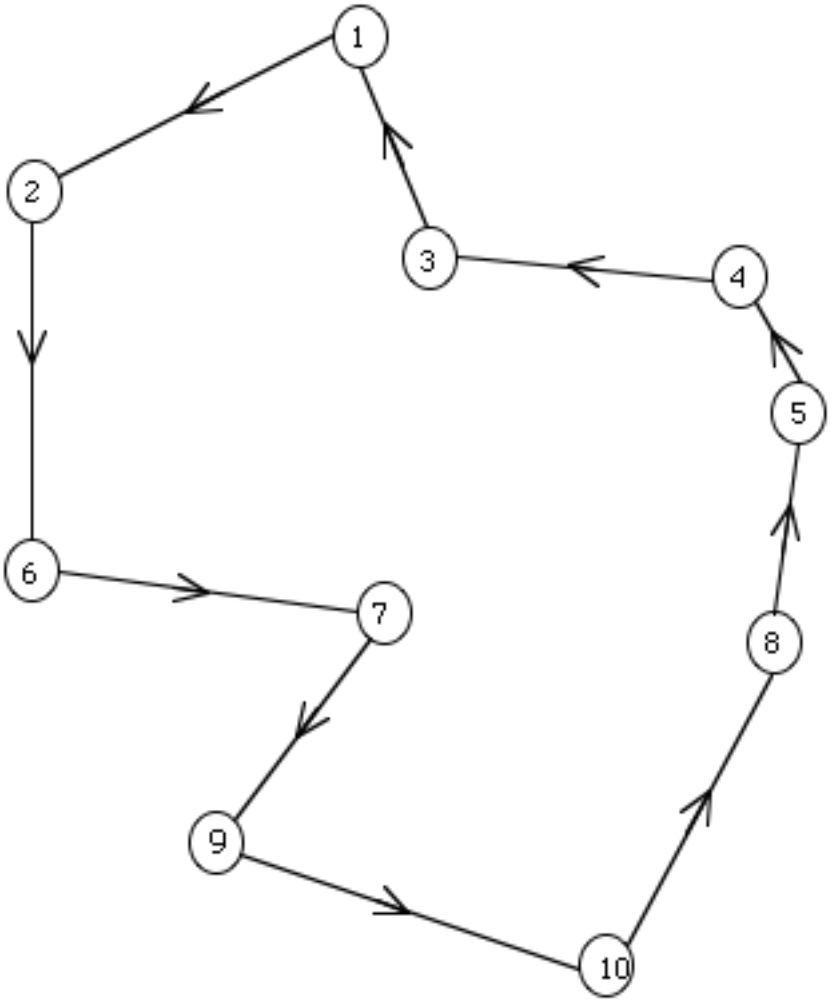


Figure 6.27: The final tour of length 67 km.

Some web links are
http://bits.bris.ac.uk/jimf/genetic/tsp.html,
http://www.nada.kth.se/theory/compendium/node103.html,
http://www.cseg.engr.uark.edu/~wessels/algs/notes/tsp.html,
http://www.npac.syr.edu/copywrite/pcw/node260.html,
http://www.ul.ie/~childsp/Elements/issue3/healy.html
http://kotn.ntu.ac.uk/daisy/chap1.htm,
http://www.tip.net.au/~dakin/tsp.htm.

Exercises

1. Consider the following 11 cities with coordinates: (1, 4), (0, 12), (9, 3), (4, 8), (6, 2), (10,10), (8, 4), (1, 10), (9, 6), (2, 5) and (3, 12). Calculate the straight line distance between each pair. Use these distances and the geometric method to find the shortest travelling salesperson tour through these cities.

2. Solve problem 1 using the nearest neighbour method.

3. Solve problem 1 using the closest insertion method.

END OF CHAPTER SIX

7. VERTEX COLOURING AND PLANAR GRAPHS

It is possible to associate a graph with a plane map in such a way that each vertex of the graph corresponds to a country of the map, and so that two vertices are adjacent if and only if the two corresponding countries are adjacent. Such a graph is a "planar graph" and we consider the problem of whether it is possible to assign one of four given colours to each vertex in such a way that any two adjacent vertices receive different colours.

This is a problem that has interested mathematicians since about the 1850's. In about 1850 Francis Guthrie (1831 - 1899) became interested in the general problem after showing how to colour the counties on a map of England with only four colours. Shortly after he showed the "Four Colour Problem" to his younger brother Frederick. At this time Frederick Guthrie (1833 - 1866) was a student of Augustus De Morgan (1806 - 1871), who communicated the problem in 1852 to William Hamilton (1805 - 1865). The problem didn't seem to interest Hamilton and lay dormant for about twenty-five years. It came into interest again to the the scientific community when Arthur Cayley (1821 - 1895) made an announcement at a meeting of the London Mathematical Society. In 1879 Cayley stated the problem in the first edition of the *Proceedings of the Royal Geographical Society*. Shortly after that the British mathematician Sir Alfred Kempe (1849 - 1922) devised a proof that remained unquestioned for over ten years. Then, in 1890, Percy Heawood (1861 - 1955), another British mathematician, found a mistake in Kempe's work.

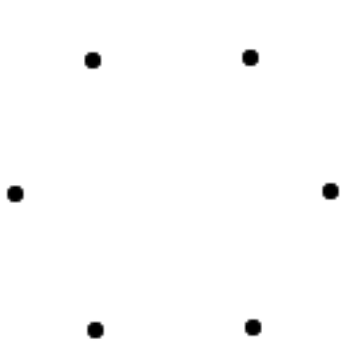
Many other proofs were attempted, but the proof was first given by Kenneth Appel and Wolgang Haken in 1976. They reduced the problem to a complicated algorithm and used high speed computers to solve the analysis of 1936 configurations.

7.1 Definitions and Examples

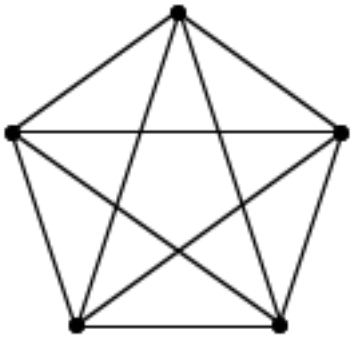
Let $G = (V, E)$ be a graph and $P = \{1, 2, ..., k\}$ be a set of k colours. We ask:
Does there exist a "colouring function" function $c : V \to P$ such that if u and v are distinct vertices and $uv \in E$, $c(u) \neq c(v)$?
The assignment c of colours to vertices is called a k-colouring of G. We say that G is k-**colourable**.
The idea of colouring of vertices of a graph also arises naturally in some kinds of scheduling problems. For example, suppose that a university is scheduling its courses for a given semester. There are many courses to be offered and a smaller number of periods in which the courses may be given. Two different courses may be intended for much the same audience, and should not be scheduled at the same time. This suggests modelling the situation with a graph G whose vertices correspond to the courses being offered, and where two vertices are adjacent if the courses they represent should not be offered at the same time. Suppose that a k-colouring of G exists. Since all the vertices of a given colour are mutually non-adjacent, the corresponding courses can all be offered at the same time. Hence all courses offered can be accommodated using k time periods.

104 Definition. *The minimum positive integer k for which G is k-colourable is called the* ***chromatic number*** *of G and is denoted* $\chi(G)$.

If we think about the definition we can see that if G has order n, then $1 \leq \chi(G) \leq n$. Also, $\chi(G) = 1$ if and only if G is empty ($G \cong \overline{K_n}$), and $\chi(G) = n$ if and only if G is complete.



(a) An Empty Graph



(b) A Complete Graph

Examples

1. (a) The chromatic number of a cycle is easily decided. Let C_p represent a cycle of order p. Then C_{2n} represents a cycle with an even number of vertices and C_{2n+1} represents a cycle with an odd number of vertices. We can see that $\chi(C_{2n}) = 2$ and $\chi(C_{2n+1}) = 3$

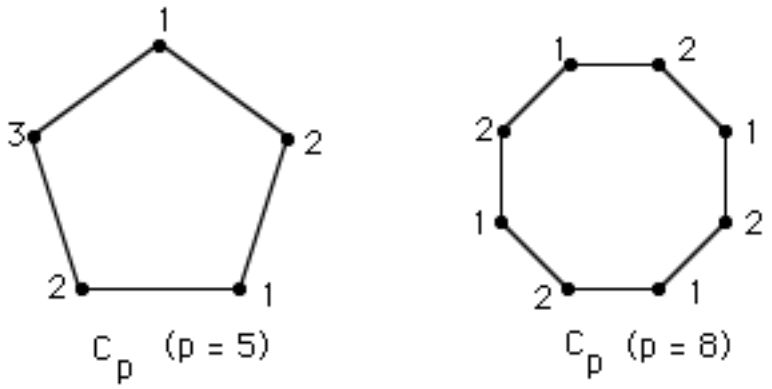


Figure 7.1: A cycle with an odd number of vertices and one with an even number of vertices.

(b) Let us consider the wheel graph of order 6 shown in Figure 7.2.

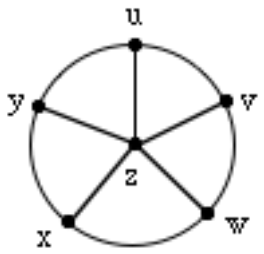


Figure 7.2: G is the wheel graph of order 6.

We notice that $G - z$ is isomorphic to the odd cycle C_5 and z is adjacent to each of these five vertices. $\chi(C_5) = 3$, and z requires its own fourth colour. It follows that $\chi(G) = 4$.

In general, if we let W_p denote the wheel of order p, then we can show that for $n \geq 2$, $\chi(W_{2n}) = 4$ and $\chi(W_{2n+1}) = 3$.

(c) Figure 7.3 shows a graph for which $\chi(G) = 4$. The colours are denoted by the numbers 1, 2, 3 and 4.

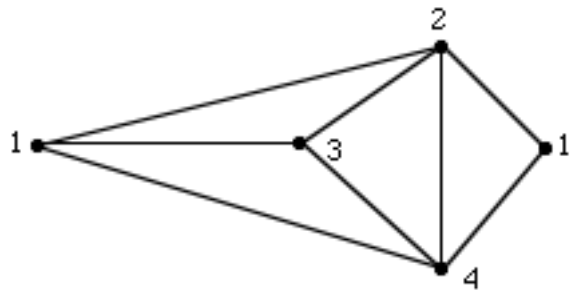


Figure 7.3: A graph for which $\chi(G) = 4$.

2. A standard technique for showing that $\chi(G) = k$ is to exhibit a k-colouring of the graph, and then to argue that G is not $(k-1)$-colourable.

   We find $\chi(G)$ for the graph shown in Figure 7.4.

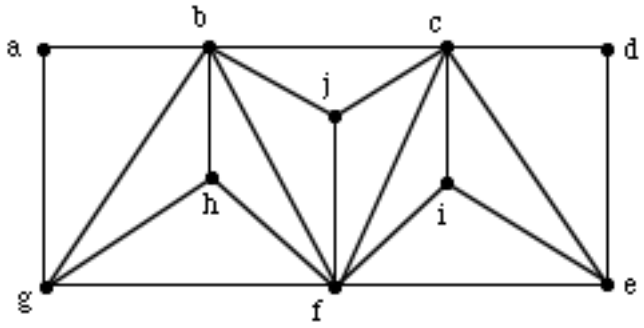


Figure 7.4: Example 7.1, part 2.

Let $U = \{b, f, h, g\}$. Then the subgraph of G induced by U is isomorphic to K_4. This means that $\chi(G) \geq \chi(K_4) \geq 4$. Therefore if we can show how to colour G with 4 colours we will have shown that $\chi(G) = 4$. One way to do this is to colour the vertices h, i and j with colour 1, b and e with colour 2, c and g with colour 3, and a, d, f with colour 4.

Suppose $G = (V, E)$ is k-colourable and a particular k-colouring of G is given. Define a relation on the vertices of G as follows:

For $x, y \in V$ $x\boldsymbol{R}y \leftrightarrow x$ and y have the same colour.

This relation $\boldsymbol{R}$ is an equivalence relation and partitions V into equivalence classes, with each class consisting of vertices of a different colour. These classes are called the colour classes of the colouring of G. If the classes are $V_1, V_2, \ldots, V_k$, then each V_i, $1 \leq i \leq k$, has the property that no two of its vertices are adjacent.
In Figure 7.5 below a 3-colouring is shown.

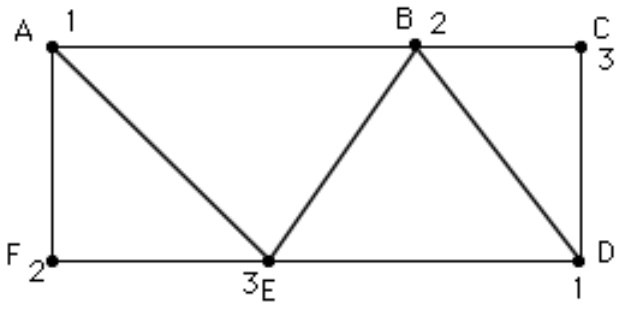


Figure 7.5: A graph with a 3-colouring.

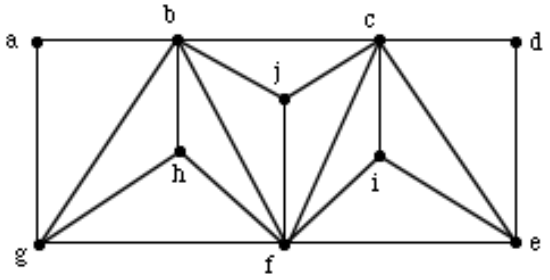
Here we can put $V_1 = \{A, D\}$, $V_2 = \{B, F\}$, and $V_3 = \{C, E\}$.

105 Definition. *Let $G = (V, E)$ be a graph. A non-empty subset U of V is called an independent set (of vertices) provided not two of its vertices are adjacent in G. An independent set U is called a maximal independent set provided U is not a proper subset of some other independent set of G.*

This means that the colour classes for a given k-colouring of G are independent sets in G. We note that the chromatic number of G is the minimum number of independent sets into which the vertex set of G may be partitioned.

Examples continued

3. The sets $V_1 = \{h, i, j\}$, $V_2 = \{b, e\}$, $V_3 = \{c, g\}$ and $V_4 = \{a, d, f\}$, which are the colour classes for the 4-colouring of Figure 7.4, are independent sets of the graph G shown again below.



In this case V_2, V_3 and V_4 are maximal independent sets, but V_1 is not since V_1 is contained in the larger independent set $U_1 = \{a,d,h,i,j\} = V_1 \cup \{a,d\}$. We can make an alternate partitioning of V and hence an alternate colouring if we put $U_1 = \{a,d,h,i,j\}$, $U_2 = V_2$, $U_3 = V_3$, and $U_4 = \{f\}$. It is clear that U_1, U_2 , U_3, and U_4 are independent sets in G whose union is V.

7.2 Graphs with Low Chromatic Number

Bipartite Graphs

We mentioned earlier that a graph has chromatic number 1 if and only if the graph is an empty graph (that is, a graph with vertices but no edges). We now want to consider graphs that have chromatic number 2. We first make a definition.

106 Definition. *A graph $G = (V,E)$ is called a **bipartite graph** if it is possible to partition the vertex set V as $V_1 \cup V_2$ in such a way that every edge of G joins a vertex of V_1 to a vertex of V_2.*

The graph in Figure 7.6 is a bipartite graph.If we put $V_1 = \{a,c,e,g\}$ and $V_2 = \{b,d,f,h\}$, then $V = V_1 \cup V_2$ and any edge of G joins a vertex in V_1 to a vertex in V_2. We can also note that V_1 and V_2 are independent sets. We can also see that by finding these independent sets we have found a 2-colouring of the graph.

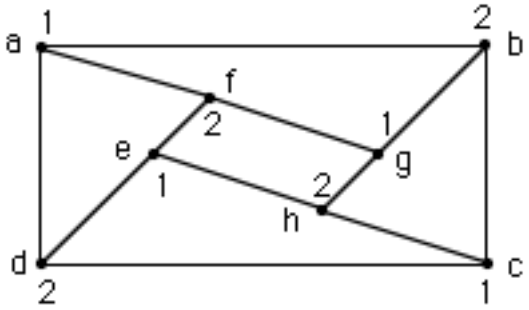


Figure 7.6: A bipartite graph.

107 Definition. *A graph $G = (V,E)$ is called a **complete bipartite graph** if it is bipartite and every vertex in the set V_1 is joined to every vertex in the set V_2. If V_1 has r vertices and V_2 has s vertices we usually denote a complete bipartite graph by $K_{r,s}$.*

Some complete bipartite graphs

Figure 7.7(a) represents $K_{4,3}$ and 7.7(b) represents $K_{1,5}$.

Figure 7.7: $K_{4,3}$ and $K_{1,5}$.

108 Theorem. *A graph $G = (V,E)$ has chromatic number 2 if and only if G is a non-empty bipartite graph.*

Proof. If G is a non empty bipartite graph, then $\chi(G) \geq 2$. If V_1 and V_2 are the partite sets of V, then V_1 and V_2 are independent sets in G whose union is V, and they supply a 2-colouring for G.
Conversely, suppose that $\chi(G) = 2$. T hen E is non-empty. For a given 2-colouring let B_1 and B_2 be the colour classes. These are both independent sets in G and so every edge in G must join a vertex of B_1 with a vertex of B_2. It follows that G is bipartite. □ □

We can also show that a graph G is bipartite if and only if it has no odd cycles. With this result we can restate Theorem 108 as follows:

A graph G has chromatic number 2 if and only if G is non-empty and has no odd cycles.

From here we can see that if G has an odd cycle, then $\chi(G) \geq 3$. The converse is also true. That is, if $\chi(G) \geq 3$ then G must contain an odd cycle. However, for $k \geq 3$ no nice characterisation is known for those graphs with chromatic number k.
There is a simple and efficient algorithm for deciding whether or not a given graph is bipartite. It produces either a 2-colouring of the graph or announces that the graph is not bipartite. It starts with a given vertex v_1 and colours it colour 0. It colours all the vertices adjacent to v_1 with colour 1, and adds these vertices to the set of vertices that have been coloured. It considers vertices adjacent to each of these vertices and colours them with the appropriate colour, checking that no vertex has already been coloured with a different colour. If there is a conflict the algorithm announces that the graph is not bipartite. Otherwise the algorithm continues until all vertices have been coloured.

The Algorithm BIPARTITE

The algorithm BIPARTITE is based on the method outlined above. The input is a graph. The output is a (0, 1) colouring of the graph if it is bipartite or else an announcement that the graph is not bipartite. We let the vertices of the graph G be the integers $1, 2, \ldots, n$.

BIPARTITE Algorithm

1. $\Sigma := \{1\}$ [initialising Σ]
2. $f(1) := 0$ [colouring the only vertex in Σ with colour 0]
3. WHILE $\Sigma \neq G$ [now pick out an adjacent vertex]
   (a) IF there is a vertex adjacent to Σ THEN
      i. $w :=$ the first vertex adjacent to Σ [now check the adjacent vertices in Σ all have the same colour.]
      ii. $v :=$ the first vertex in Σ adjacent to w
      iii. $f(w) = 1 - f(v)$ [changes 1 to 0 and 0 to 1]
      iv. IF (every u in which is adjacent to w satisfies $f(u) = 1 - f(w)$ THEN
         A. $\Sigma := \Sigma \cup \{w\}$ [w checks out and its colour has been properly assigned]
            OTHERWISE
         B. Output "Graph is not bipartite"
         C. End
   (b) IF there is no vertex adjacent to Σ THEN
      i. $w :=$ any vertex in $G - \Sigma$
      ii. $f(w) := 0$
      iii. $\Sigma := \Sigma \cup \{w\}$ [start a new component]

One use of bipartite graphs is in the theory of Hamilton circuits. Clearly any **circuit** in a bipartite graph (coloured red and blue, for example) must contain the same number of red and blue points. Therefore, if a Hamilton circuit exists in a **bipartite graph**, the graph must have the same number of red and blue points. Similarly, if a Hamilton **path** exists on a 2-coloured graph, and it is not a cycle, there is at most one more of one colour than the other.

Chromatic Number 3 and Higher

The problem of determining whether a given graph is **3-colourable** is much more difficult and is classified as NP-complete. The class of NP-complete problems is interesting in that no (deterministic) polynomial-time algorithms are known for any of them. Yet if a polynomial-time algorithm is found for any one of the problems in the class, then polynomial-time algorithms can be constructed for all of them. If it is proven that no polynomial-time algorithm exists for some problem in the class, then no polynomial-time algorithms exist for any of them.
There are a couple of theorems that help a little in determining the chromatic number of a graph, and we state them below.

109 Theorem. *Let G be a graph whose largest vertex degree is d. Then $\chi(G) \leq d+1$.*

110 Theorem. (Brookes, 1941) *Let G be a connected graph which is not a complete graph. If the largest vertex degree is $d \geq 3$, then $\chi(G) \leq d$.*

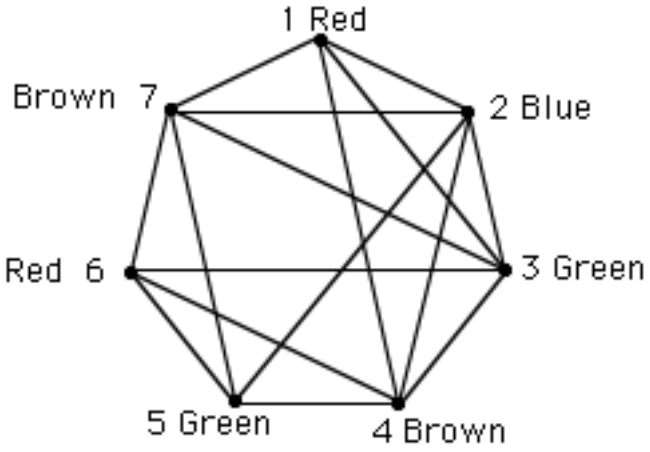
We just note here that the bound given by Brookes' Theorem is not always particularly helpful. If we consider the complete bipartite graph $K_{1,n}$, then the largest vertex degree is n, and so Brookes' Theorem tells us that $\chi(K_{1,n}) \leq n$, whereas we know that $\chi(K_{1,n}) = 2$.

7.3 An Application of Graph Colourings

Graph colouring has a variety of applications to problems involving scheduling and assignments.
Scheduling final exams. How can final exams be scheduled at a university so that no student has two exams at the same time?
Solution : This scheduling problem can be solved using a graph model, with vertices representing courses and with an edge between two vertices if there is a common student in the courses they represent. Each time slot for a final exam is represented by a different colour.
Suppose there are seven final exams to be scheduled and the courses are numbered $1, \ldots, 7$. The following pairs of courses have common students:

1 and 2, 1 and 3, 1 and 4, 1 and 7, 2 and 3, 2 and 4, 2 and 5, 2 and 7,
3 and 4, 3 and 6, 3 and 7, 4 and 5, 4 and 6, 5 and 6, 5 and 7, and 6 and 7.

The graph associated with this set of classes is shown below left.



Time Period	Courses
I	1, 6
II	2
III	3, 5
IV	4, 7

Figure 7.8: The solution to the scheduling problem.

A scheduling consists of a colouring of this graph. Since the chromatic number of this graph is four (you should check this), four colours (representing the time slots) are needed. A colouring using four colours is indicated on the graph, and the associated schedule is shown to the right of the graph.

7.4 Chromatic Polynomials

In this section we aim to associate with any graph a function which will tell us, among other things, whether or not the graph is 4-colourable.
We restrict our attention to simple graphs, and let $G = (V, E)$ be a simple graph. Let $P_G(k)$ denote the number of ways of colouring the vertices of G with up to k colours in such a way that no two adjacent vertices have the same colour. P_G will be called the **chromatic function** of G.

Some Examples

1. For the graph G shown in Figure 7.9, $P_G(k) = k(k-1)^2$ since the middle vertex can be coloured in k ways and each terminal vertex in $(k-1)$ ways.

Figure 7.9: Example 1.

2. If $G = (V, E)$ with $|V| = n$ and $E = \emptyset$, then G consists of n isolated points, and by the product rule $P_G(k) = k^n$.

Figure 7.10: As there are 5 vertices here and no edges, $P_G(k) = k^5$.

3. If $G = K_n$, the complete graph on n vertices, then at least n colours must be available for a proper colouring of G. Here, by the product rule

$$P_G(k) = k(k-1)(k-2\ldots(k-n+1).$$

Note that $P_G(k)$ is zero when $k < n$, correctly indicating that there is no proper k-colouring of K_n in this case. Moreover $P_G(k)$ becomes non-zero for the first time when $k = n = \chi(G)$.

For example, if $G = K_4$ (Figure 7.11) at least 4 colours must be used for a proper colouring of G, and $P_G(k) = k(k-1)(k-2)(k-3)$.

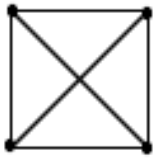


Figure 7.11: The complete graph K_4.

4. We can generalise the result of 1. by considering a path with more vertices. E.g.:

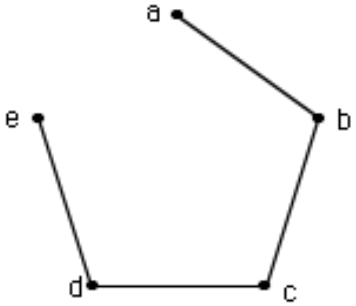


Figure 7.12: A Path with 5 vertices.

Vertex a can be coloured in k ways, and the rest of the vertices b, c, d and e can each be coloured in $(k-1)$ colours. So for this graph $P_G(k) = k(k-1)^4$. This means that $P_G(1) = 0$, but $P_G(2) = 2$ and $\chi(G) = 2$. If five colours are available, $P_G(5) = 5(4)^4 = 1280$.

In a similar way, if we have a path with n vertices, $P_G(k) = k(k-1)^{n-1}$. In fact we can take this further and deduce that if T is any tree with n vertices then $P_T(k) = k(k-1)^{n-1}$. For example:

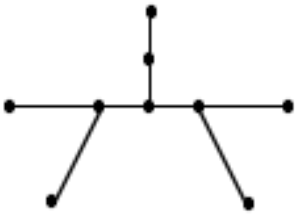


Figure 7.13: A tree with 9 vertices.

Here we have a tree T with 9 vertices. From what has been said above, we conclude that $P_T(k) = k(k-1)^8$.

The Decomposition Theorem for Chromatic Polynomials

We now look for some systematic way of finding the chromatic function for more general graphs. We want to do this by somehow breaking a graph down into two or more simpler parts. Let $G = (V,E)$ be a simple connected graph. For an edge $e = \{a,b\} \in E$, let $G\backslash e$ denote the subgraph of G obtained by deleting e from G, without removing the vertices a and b. From $G\backslash e$ a new graph is obtained by coalescing (the jargon word is "identifying") the vertices a and b. This second graph is denoted by $G\downarrow e$.

Figure 7.14 below shows the graphs $G\backslash e$ and $G\downarrow e$ for the graph G with the edge e as specified.

This may seem like quite an extraordinary construction and at this stage we would wonder why we might be interested in such subgraphs of a given graph. That question is answered by the next theorem.

111 Theorem. Decomposition Theorem for Chromatic Polynomials *If $G = (V,E)$ is a connected graph and $e \in E$, then*

$$P_{G\backslash e}(k) = P_G(k) + P_{G\downarrow e}(k).$$

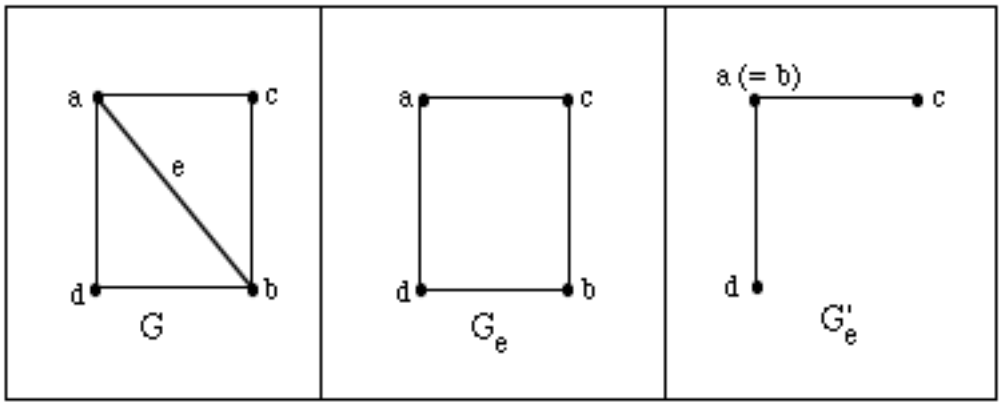


Figure 7.14: Finding $G_e = G\backslash e$ and $G'_e = G\downarrow e$.

Proof. Let $e = \{a,b\}$. The number of ways to properly colour the vertices in $G_e = G\backslash e$, with at most k colours is $P_{G\backslash e}(k)$. Those colourings where a and b have different colours are proper colourings of G. The colourings of $G\backslash e$ that are not proper colourings of G occur when a and b have the same colour. But each of these colourings corresponds with a proper colouring of $G'_e = G\downarrow e$.
This partition of the $P_{G\backslash e}(k)$ colourings into the two disjoint subsets described results in the equation

$$P_{G\backslash e}(k) = P_G(k) + P_{G\downarrow e}(k).$$

□

We usually use this result in the form

$$P_G(k) = P_{G\backslash e}(k) - P_{G\downarrow e}(k).$$

To find the polynomials $P_{G\backslash e}(k)$ and $P_{G\downarrow e}(k)$ we can, if necessary, repeat the procedure described above on each of the graphs $G\backslash e$ and $G\downarrow e$ and then repeat the procedure on all the new graphs obtained, and so on. The process terminates when we are left with trees or complete graphs, for which we know the chromatic polynomial.

Examples

1. We find $P_G(k)$ when G is a cycle of length 4.

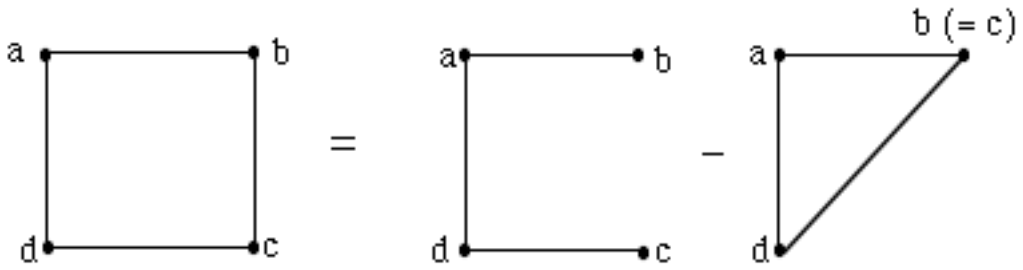


Figure 7.15: $P_G(k) = P_{G\setminus e}(k) - P_{G\downarrow e}(k)$.

Consider the graph and its decomposition in Figure 7.15. From here we know that $P_{G\setminus e}(k) = k(k-1)^3$ and $P_{G\downarrow e}(k) = k(k-1)(k-2)$, since $G\downarrow e$ is the complete graph on 3 vertices.

Therefore,

$$\begin{aligned} P_G(k) &= P_{G\setminus e}(k) - P_{G\downarrow e}(k) \\ &= k(k-1)^3 - k(k-1)(k-2) \\ &= k(k-1)[(k-1)^2 - (k-2) \\ &= k(k-1)[k^2 - 3k + 3] \\ &= k^4 - 4k^3 + 6k^2 - 3k. \end{aligned}$$

Since, $P_G(1) = 0$ and $P_G(2) = 2 > 0$, we conclude that $\chi(G) = 2$.

2. We find the chromatic polynomial for the graph shown in Figure 7.16.

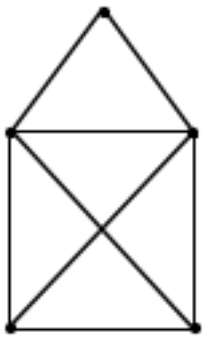
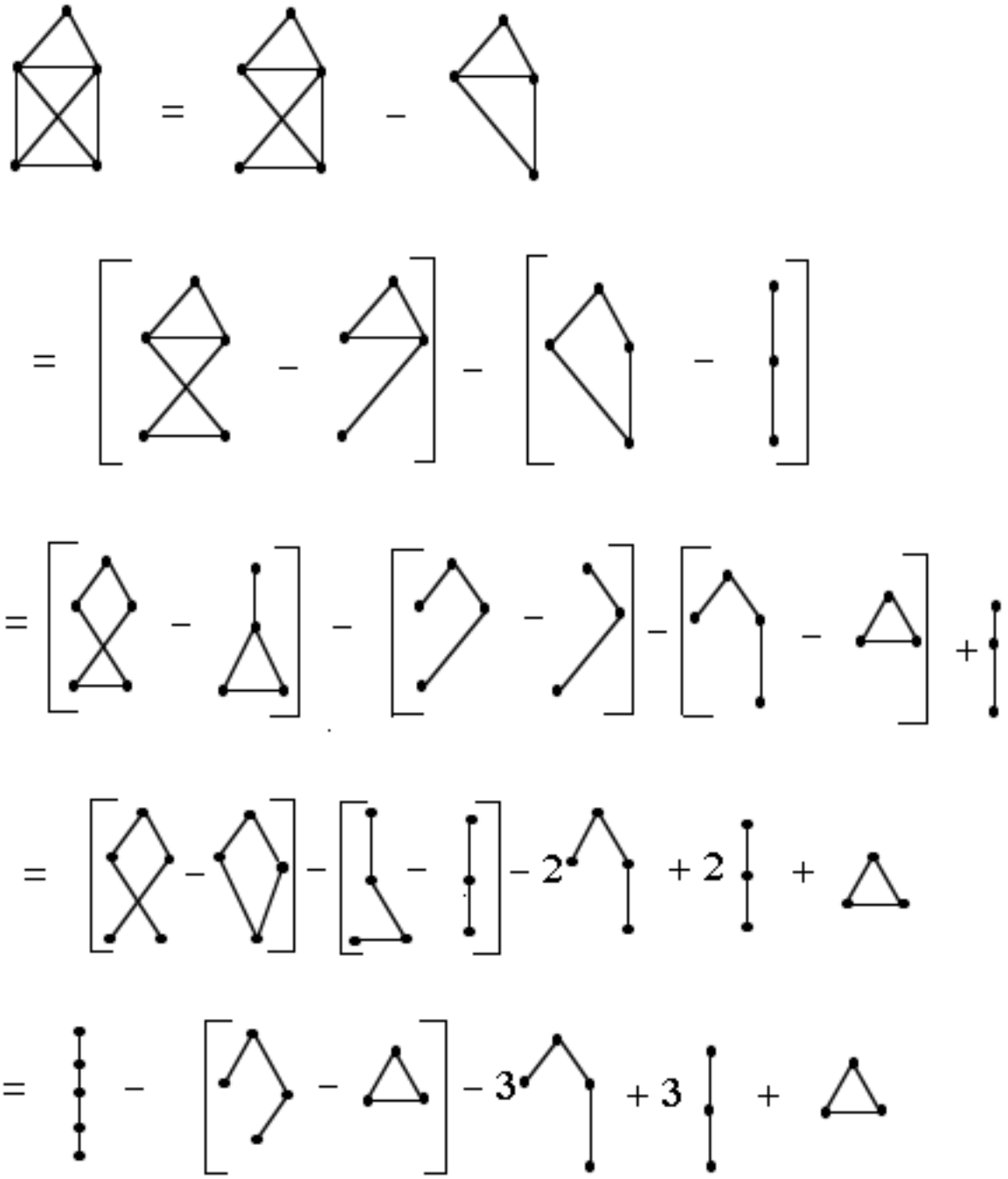
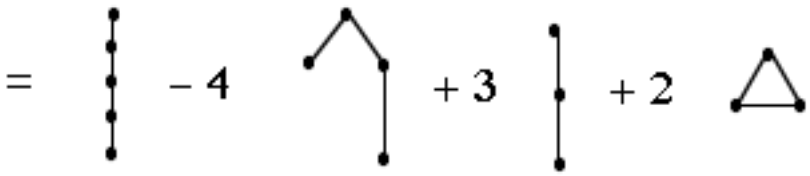


Figure 7.16: Example 2.

We decompose this graph as follows:





From here we can work out the chromatic polynomial for the original graph:

$$\begin{aligned} P_G(k) &= k(k-1)^4 - 4k(k-1)^3 + 3k(k-1)^2 + 2k(k-1)(k-2) \\ &= k(k-1)\{(k-1)^3 - 4(k-1)^2 + 3(k-1) + 2(k-2)\} \\ &= k(k-1)\{k^3 - 3k^2 + 3k - 1 - 4k^2 + 8k - 4 + 5k - 7\} \\ &= k(k-1)\{k^3 - 7k^2 + 16k - 12\} \\ &= k(k-1)(k-2)^2(k-3) \end{aligned}$$

This shows us that for $1 \leq k \leq 3$, $P_G(k) = 0$, and for $k \geq 4$, $P_G(k) > 0$. This graph has chromatic number 4.

We conclude this section with some results about chromatic polynomials.

112 Theorem. *For any graph G the constant term in $P_G(k)$ is 0.*

Proof. For any graph G, $\chi(G) > 0$ because the vertex set V is not empty. If $P_G(k)$ had constant term a, then $P_G(0) = a \neq 0$. This implies that there are a ways to colour G properly with 0 colours. □

113 Theorem. *Let $G = (V,E)$ with $|E| > 0$. Then the sum of the coefficients in $P_G(k)$ is 0.*

Proof. Since $|E| \geq 1$, $\chi(G) \geq 2$. This means that $P_G(1) = 0 =$ the sum of the coefficients in $P_G(k)$. □

7.5 Planar Graphs

In the introduction we talked about map colouring, and the association of a planar graph with a map. Now we need to talk about planar graphs so that we can look at the question of map colouring.

114 Definition. *A **planar graph** is one that can be drawn in the plane in such a way that the edges meet only at vertices. Such a diagram is called a **plane graph**.*

WARNING!: A plane graph is a type of diagram, not a type of graph. A diagram depicting a planar graph may or may not be a plane graph. So it may not be immediately obvious whether a given graph is planar or not, because the same graph can be drawn in many different ways.

Examples

1. The diagrams in Figure 7.17 below depict the same planar graph. Only the diagram on the right is a plane graph.

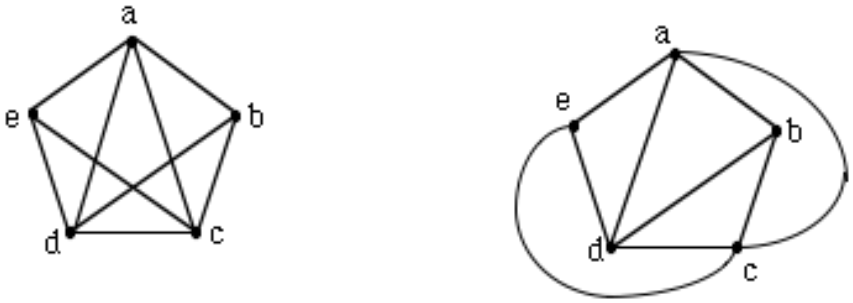


Figure 7.17: Two diagrams of a Planar Graph.

2. Are ALL graphs planar? No. Figure 7.18 illustrates a non-planar graph. The proof that the it is not planar follows later in the notes.

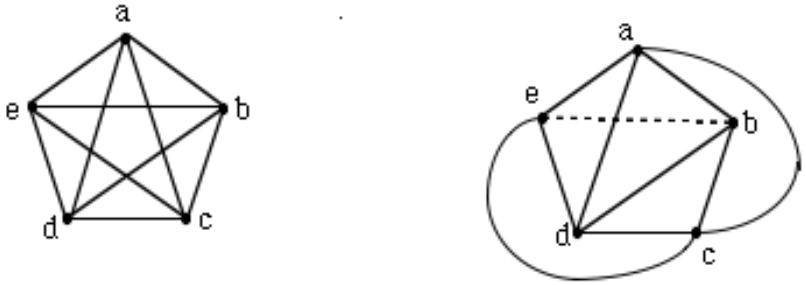


Figure 7.18: Two diagrams of the non-planar graph K_5.

We would like to be able to determine whether a graph is planar or not, independently of how it is depicted. Towards this goal we need to concept of duality.

115 Definition. *A **dual** G' of a plane graph G is any plane graph built from G as follows:*

- *Create a vertex of G' in each face (region) of G. (A face is a region of the plane containing no edges but entirely bounded by edges* - i.e. *the faces are the spaces between the lines. The unbounded region outside all the edges is also a face.)*
- *Create an edge of ε' of G' for each edge ε of G. The edge ε' crosses ε and its endpoints are the vertices pf G' located in the faces of G on either side of ε. (It is possible these faces are the same, in which case e' is a loop.)*

In Figure 7.19 we show a plane graph G with 5 vertices a, b, c, d, e, 6 faces and 9 edges. The dual plane graph shown, G', has 6 vertices A, B, C, D, E, F, 5 regions and 9 edges.

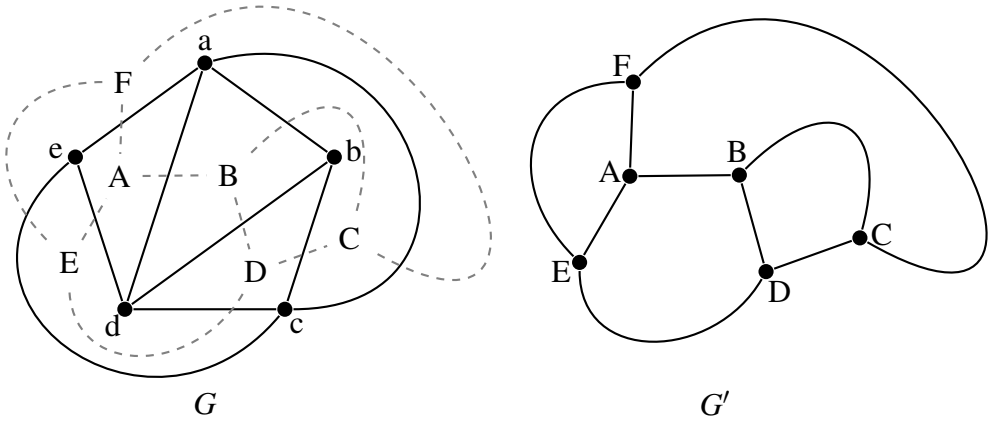


Figure 7.19: A plane graph and dual.

We consider all duals of plane graph G to be 'equivalent' in that G itself is clearly a dual of each of these duals. (This is the 'duality'.)
Note that, in general, we cannot speak of *the* dual of a *planar* graph; we can only speak of duals of its various *plane* graph depictions. These duals do not necessarily depict isomorphic graphs.

Next we introduce some convenient notation relating to a planar graph G.

$$\begin{aligned} v &= v(G) = \text{the number of vertices in } G, \\ e &= e(G) = \text{the number of edges in } G, \\ r &= r(G) = \text{the number of regions (faces) in a plane graph depicting } G. \\ &\qquad \text{(as the next theorem shows, this does not depend on the plane graph chosen)} \end{aligned}$$

Note that $e(G) = e(G'), v(G) = r(G')$, and $r(G) = v(G')$.

116 Theorem. (Euler) *Let G be a connected planar graph. Then*

$$v - e + r = 2.$$

Proof. The proof is by induction on the number of edges of a graph. For $e = 0$ there are no edges, and since the graph is connected it must consist of a single vertex. This means there is also a single region. So we have $v = 1, e = 0$ and $r = 1$. Hence $v - e + r = 2$.
We now assume that the theorem is true for all connected planar graphs with n edges, for $n \geq 0$. We want to consider graphs with $n+1$ edges. We note that graphs with $n+1$ edges can be constructed from graphs with n edges by one of the following processes:

(a) Adjoining a new vertex and connecting it by an edge to an existing vertex, without intersecting an existing edge,

(b) Joining two existing vertices with an edge, again without intersecting an existing edge,

(c) Joining a vertex to itself with a loop.

These situations are shown in Figure 7.20.

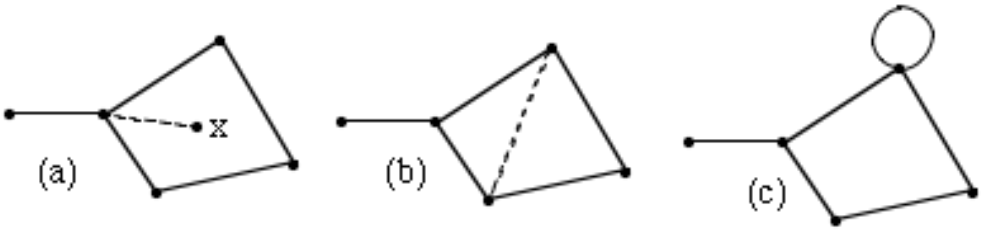


Figure 7.20: Adding an edge.

This procedure in case (a) adds a vertex and an edge, but doesn't change the number of regions. This means that v is increased by 1, e is increased by 1 and r is unchanged. Thus $v-e+r$ is unchanged.
In the situation (b), a new edge is added and a new region is formed, so that both e and r are increased by 1 and v is unchanged. Again$v-e+r$ is unchanged.
In the last case (c), e and r are again both increased by 1 making no change in $v-e+r$
We summarise this as follows:

	v	e	r	$v-e+r$
(a)	$v+1$	$e+1$	r	unchanged
(b)	v	$e+1$	$r+1$	unchanged
(c)	v	$e+1$	$r+1$	unchanged

This completes the proof.

□

We can now get back to the question of which graphs are planar and which aren't. We have already seen that K_5 is not planar, and we also know that K_5 can't be coloured with 4 colours. We give another example of a non-planar graph.

Another non-planar graph

This example relates to the following problem:
Three houses A, B and C are to be connected by cables or pipes to the three suppliers of (E)lectricity, (O)il and (W)ater. The connecting pipes and cables are to be laid at a depth of 3 metres and are not allowed to intersect. Show how this can be done, if possible.
Investigation: The graph in Figure 7.21 models the situation in which we are interested:
The second diagram represents just one attempt to draw the graph as a planar graph. It is clearly unsuccessful. Perhaps there are other ways! But it seems unlikely. Note that this graph is $K_{3,3}$, the complete bipartite graph with three vertices in one partite set and three in the other.
The next theorems give us some help in showing that in particular, the graphs K_5 and $K_{3,3}$ are not planar.

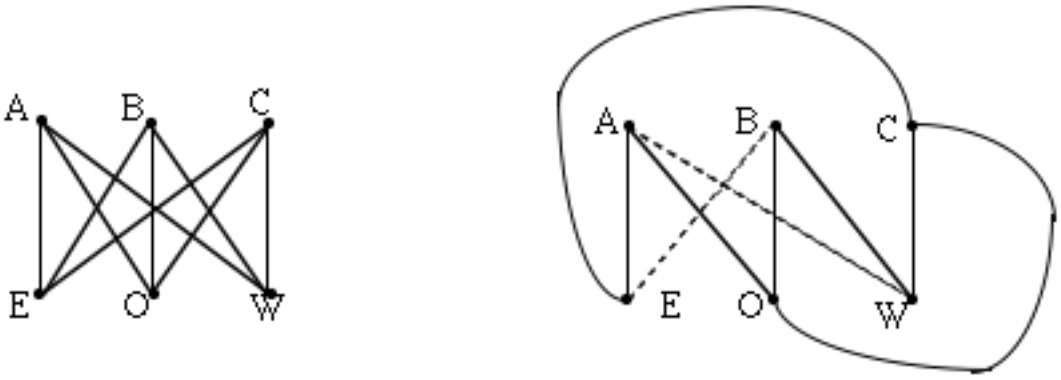


Figure 7.21: A non-planar situation.

117 Theorem. *Let G be a simple, connected planar graph with v vertices and e edges, where $v \geq 3$. Then*

$$e \leq 3v - 6.$$

Proof. In a plane graph depiction P of G, the plane will be divided into r regions. Let us first consider the case in which each of these regions is bounded by at least 3 edges.

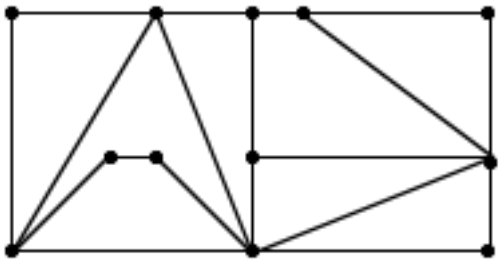


Figure 7.22: An example of a plane graph for P for a planar graph G.

This means that each vertex of a dual P' has degree 3 or more. Further this implies that the sum of the degrees of the vertices of P' is $3r$ or more. We also know that the sum of the degrees of the vertices is twice the number of edges (handshake lemma), which implies that

$$3r \leq 2e.$$

From Euler's formula, $r = e - v + 2$, which gives $3r = 3e - 3v + 6$. Substituting this into the inequality above gives

$$3e - 3v + 6 \leq 2e \qquad \text{or} \qquad e \leq 3v - 6.$$

This deals with the case where each region is bounded by 3 or more edges. If this is not the case and G is a connected simple graph with at least 3 vertices then P must look like the plane graph shown below.

Here there is one unbounded region, three vertices and two edges. So $e = 2$ and $3v - 6 = 9 - 6 = 3 > 2$. Hence the inequality is also satisfied in this special case. □

With this theorem we can immediately deduce that K_5 is not planar, since for K_5, $e = 10$ and $v = 5$ and $e \leq 3v - 6$ is not valid for theses numbers.
But this theorem is not strong enough for $K_{3,3}$ since for this graph $e = 9$ and $v = 6$ and the equality is valid. We find that for bipartite graphs there is a stronger result.

118 Theorem. *Let G be a simple, connected planar bipartite graph with v vertices and e edges, with $v > 3$. Then*

$$e \leq 2v - 4.$$

Proof. In a plane graph depiction of a simple bipartite graph, we first assume that the boundary of any region, being a cycle, must have 4 or more edges (even and not 0 or 2). Exception cases are dealt with at the end.

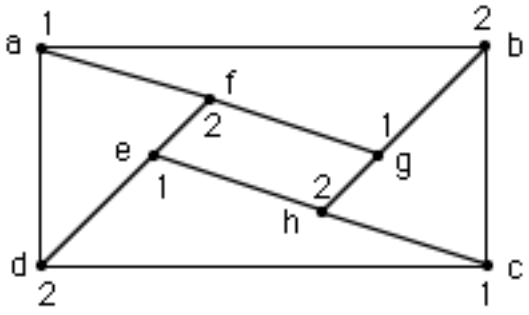


Figure 7.23: A plane graph for a sample simple bipartite graph.

If we follow the same reasoning as in the previous theorem we get the inequality

$$4r \leq 2e \qquad \text{or} \qquad 2r \leq e.$$

When we combine this with Euler's formula we get

$$2r = 2e - 2v + 4 \leq e \qquad \text{or} \qquad e \leq 2v - 4.$$

The only exceptions to the assumption that the boundary of each region has 4 or more edges are trees with 2 or 3 edges. In these cases it is easy to see that the inequality is true. □

As we mentioned above, for $K_{3,3}$, $e = 9$ and $v = 6$ and with these values the inequality $e \leq 2v - 4$ is not valid. Thus we conclude that $K_{3,3}$ is not planar.

Kuratowski's Theorem

These results that K_5 and $K_{3,3}$ are not planar turn out to be very important because we can characterise non planar graphs in terms of these two graphs. But first we need some additional terminology.

119 Definition. *An **elementary subdivision** of a non empty graph $G = (V,E)$ is obtained from G by removing an edge $e = uv$ and adding a new vertex $w \notin V$, along with edges uw and vw.*

If we start with the graph given in Figure 7.24(a), two possible elementary subdivisions are shown in figures (b) and (c).

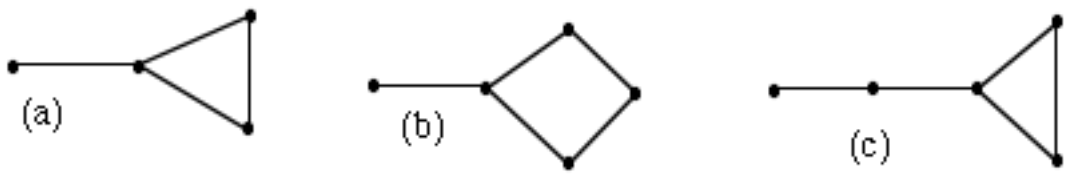


Figure 7.24: Elementary subdivisions of a graph.

120 Definition. *A **subdivision** of the graph G is a graph obtained from G by applying the recursive definition:*

1. *G is a subdivision of G*
2. *If H_1 is an elementary subdivision of G, and H_2 is an elementary subdivision of H_1, then H_2 is a subdivision of G.*

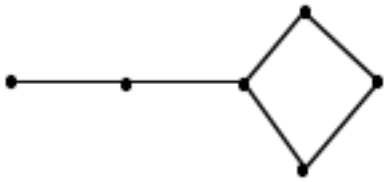


Figure 7.25: A division of the graph in Figure 7.24(a).

The graph shown in Figure 7.25 is a subdivision of the graph in Figure 7.24(a).

121 Theorem. (Kuratowski, 1930) *A graph G is planar if and only if G does not contain a subgraph H that is isomorphic to a subdivision of K_5 or $K_{3,3}$.*

A non-planar graph

The graph shown below in Figure 7.26 is Petersen's Graph. We show that it is non planar by finding a subgraph H which is isomorphic to a subdivision of $K_{3,3}$.

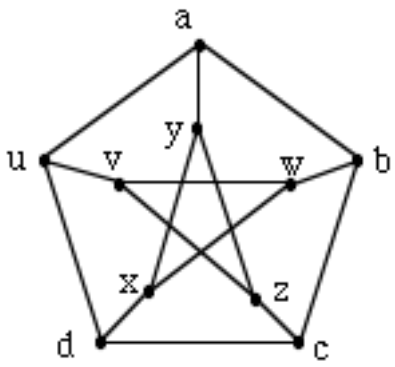


Figure 7.26: Petersen's Graph.

We first note that the graph is not bipartite. (Why?) It has 10 vertices and 15 edges and so satisfies the inequality $e \leq 3v - 6$. However, if we look at the graph in Figure 7.27 we see that it is a subgraph of the Petersen Graph and is also a subdivision of $K_{3,3}$.

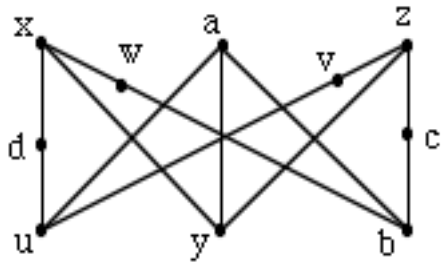


Figure 7.27: A subgraph of the Petersen graph.

We conclude that the Petersen Graph is not planar.

To finish, we give (without proof!) the famous theorem for planar graphs.

122 Theorem. (Appel, Haken and Koch, 1977) *For any planar graph G,* $\chi(G) \leq 4$.

END OF CHAPTER SEVEN

8. MATRIX GAMES

The *theory of games* considers situations in which there are two or more persons each of whom controls some, but not all, of the variables necessary to determine the outcome of a certain event. Usually the players will not agree as to which outcome should occur, so that their objectives in the game are different, usually conflicting.

Game theory is designed to help people who are in conflict situations determine the best choice of action out of several possible choices. The theory is applicable to some parlour games, but more importantly, it has been applied with some success to decision making problems in economics, business, psychology, sociology and other areas.

Roughly speaking, a *solution* to a game is a set of strategies, one for each player, that best approaches each player's specific objectives for the game. We will be more precise about solutions to one type of game, called a matrix game, the subject of this chapter.

We assume that all players have complete knowledge of all actions, moves or choices available to them and their opponents, and knowledge of the results of the conflict associated with any given selection of actions. We assume that each player acts rationally to maximise his or her gain. The basic problem, then, is to develop a theory that will help us to understand and predict human behaviour or economic phenomena.

Depending on which outcome of the game event that actually occurs, the players may receive various payments. If for each possible outcome the algebraic sum of these payments to all players is zero, the game is called *zero-sum*.

123 Definition. *A* two-person, zero-sum game *is a game played by two people, conventionally known as Alice and Bill. The game must end, and when it does each player receives an award, called their* payoff. *Bill's payoff is the negative of Alices's payoff.*

Because Bill's payoff is the negative of Alice's we usually just keep track of Alice's payoff. So if Alice's payoff is $100 then Bill's is -$100. In a game Alice tries to maximise her payoff and Bill tries to maximise his. This means in fact that Bill tries

to minimise Alice's payoff. In a two-person zero-sum game anything Alice does to improve her payoff necessarily reduces Bill's payoff, and vice-versa.

8.1 Payoff Matrices

A *matrix game* is a two-person zero-sum game in which each player has only a finite number of ways (choices) to influence the outcome of game event. It is called a matrix game because the payoff for all possible outcomes can be recorded in a matrix, called, naturally, the payoff matrix.

124 Definition. *The* payoff matrix *for a two-person zero-sum game in which the first player (Alice) has m choices and the second player (Bill) has n choices is the $m \times n$ matrix A whose i,j-th entry a_{ij} is the payoff to Alice that results from her making choice i and Bill making choice j.*

Some examples of payoff matrices are provided below. By convention, the solution to a matrix game is based on the principle that each player plays *conservatively* (you might say *pessimistically*). That is, each player's choice is designed to ensure that, considering all the possible choices for their opponent, their own *minimum* payoff is as great as possible. (By contrast, an optimistic player might want to ensure that the maximum payoff was as great as possible.) Depending on the game, this minimum may be achievable any time the game is played, or may only be achievable as an average or expected value if the game were played a number of times. Bare in mind that this minimum payoff may be negative; in that case the principle may be viewed as guarding against heavier losses than necessary. The following examples should help to clarify this conservative principle for the solution of matrix games.

Three Examples

Example 1: Alice and Bill have two cards each, an ace (*i.e.* a One) and a Two. Each player selects one of their cards, with their choice unknown to their opponent. They then compare the two selected cards. If the sum of the two face values of the selected cards is even, Alice wins that sum from Bill. If the sum is odd, Bill wins that amount from Alice.

This is an example of a *two-person, zero-sum game*, because what one player wins, the other loses. It is a matrix game because the information critical to the game, the various payoffs, can be simply recorded using matrix notation:

$$\begin{array}{cc} & \text{Bill} \\ & \begin{array}{cc} \text{Ace} & \text{Two} \end{array} \\ \text{Alice} \begin{array}{c} \text{Ace} \\ \text{Two} \end{array} & \begin{bmatrix} 2 & -3 \\ -3 & 4 \end{bmatrix} \end{array}$$

In this payoff matrix the entry in the 2nd row and the 1st column (for example) represents the amount Player 2 pays Player 1 if Player 1 plays her two and Player 2 plays her ace. The negative sign indicates that Player 2 actually wins that amount from Player 1.

Let us first focus on what Bill should do. If he plays his Ace he stands to lose 2 or win 3. If he plays his Two he stands to win 3 or lose 4. Playing conservatively, at first sight it seems obvious that he should play his Ace, to minimise his potential loss, or, in game theory jargon, to achieve the greatest minimum payoff (since $-2 > -4$). Indeed, if the game is to be played only once, Bill *should* play his Ace. But what if the game is to be played many times? If Bill always plays his Ace, surely Alice will 'catch on' and always play *her* Ace, ensuring she always wins 2 at Bill's expense. Would it not be better for Bill to occasionally, and randomly, play his Two? Although this risks a greater loss on an individual game, it has two potential advantages: Firstly it provides him the opportunity to win 3 on each game that Alice continues to play her Ace, and secondly this in turn may cause Alice to stop always playing her Ace so that then he will occasionally win another 3 when he plays his Ace and she her Two. Do these potential advantages outweigh the extra risk? If so what proportion of plays should be allocated to each card and can we quantify the long term effect? And what about Alice — what is *her* best strategy against a strategic opponent? If they both play optimally, who does best in the long term, or, given that $2-3-3+4=0$, will things inevitably even out? We will return to these questions later in the chapter.

Example 2: Tire Town and its principal competitor, Tire City, are both planning sales for the big Sales Weekend. They both usually discount 10 or 20%, and each knows the other's practice. Tire Town is somewhat more popular and will receive 65% of the trade if they both discount by the same amount. On the other

hand, if Tire Town discounts more than Tire City, then it will receive 80% of the business. Conversely, Tire City will receive 55% of the business if it discounts more than Tire Town. Assuming each manager is free to choose either discount, and volume of business is the only issue, which should each choose?

Solution: We first display the percentage split of the customers for each case in a table. Since the split always sums to 100, we need only show the portion obtained by one of the companies. In the following table we give the percentage of the business going to Tire Town for each of the discount possibilities.

		Tire City	
		10%	20%
Tire Town	10%	65	45
	20%	80	65

We can now analyse the information. Since both managers know the options of their competitor, each will assume the worst when viewing the possible actions. Consider Tire Town first. From the first row of the table we see that if Tire Town discounts by 10%, then Tire City could discount by 20% and Tire Town would obtain only 45% of the Trade. From the second row of the table if Tire Town discounts by 20%, then the worst outcome would be that Tire City also discounts by 20% and Tire Town would receive 65% of the Trade. With this analysis Tire Town would choose row 2 since the 20% discount guarantees the greater percentage of the trade.

From Tire City's perspective the analysis is similar. Since the table is given in terms of gains to Tire Town, tire City would want an action that keeps the percentage as low as possible. From column 1, if Tire City discounts by 10% then Tire Town could discount by 20% and get 80% of the business. Similarly, from column two, if Tire City discounts by 20% then Tire Town could discount by 20% and get 65% of the trade. So Tire City chooses column 2 since the 20% discount guarantees the smaller percentage for Tire Town.

In this game of retail competition each company should advertise a 20% discount and Tire Town would obtain 65% of the trade. Unlike the situation in the previous example, if either manager sticks to this plan then their is never any advantage in the other changing plan, even occasionally and randomly (*e.g.* only on a sales weekend when the manager's mother sneezed before breakfast.) So there is no element of uncertainty or surprise over the outcome.

The 'game' of Example 2 is a **two-person**, **constant-sum** game, with (non-zero) constant of 100. But a constant-sum game can be considered to be zero-sum if one of the players is thought to have that constant sum in his possession and the other player wins some portion of it. In the example above we may think of Tire City having 100% of the trade until Tire Town wins some of that percentage. In such a game we need only to explicitly consider the gains of one of the players since the gains of the other can be deduced. For these reasons the term 'zero-sum' is often used loosely to include 'constant sum'.

When it is not appropriate to call the players 'Alice' and 'Bill', they are referred to generically as the **row player** and the **column player** respectively. The row player's possible choices or actions, listed down the left side of the payoff matrix, are referred to generically as $R_1, R_2, \ldots, R_m$ and the column player's, listed across the top, as $C_1, C_2, \ldots, C_n$. So in Example 2 Tire Town is the row player, Tire City the column player, $R_1 = C_1 = 10\%$ and $R_2 = C_2 = 20\%$. In generic terms the entry a_{ij} of the payoff matrix in the payoff **to the row player** resulting from choices C_i and R_j. For the column player, the payoff resulting from these choices is $S - a_{ij}$, where S is the constant sum of the game. ($S = 0$ for a zero-sum game.) So the lower the payoff, hopefully negative in fact, the better things are for the column player. For this reason the column player is called the **minimising player** whereas the row player is naturally called the **maximising player**.

Our final introductory example is another case of non-zero constant sum:

Example 3: Two competing lunch wagons, Pete's and Dave's, can each park by the town hall or the police station down the block, but for stocking reasons the decisions must be made in advance. There is a total of \$1200 worth of business at the two locations and it will split evenly if they park at different locations. However, if they both park at the town hall Pete will sell only \$500 worth, whereas he will sell \$700 worth if they both park at the police station. Where should each of the vendors park their wagon?

Solution: We arbitrarily assign Pete the row player, and Dave the column player. Then each has two possible actions: park at the town hall or park at the police

station. The payoff matrix is given as follows:

		Dave	
		Town Hall	Police Station
Pete	Town Hall	500	600
	Police Station	600	700

Each of the vendors is sharp and can work out his best position. Pete reasons that if parks at the town hall, then Dave could also park at the town hall restricting his (Pete's) payoff to \$500. If he parks at the police station then Dave could park at the town hall restricting his (Pete's) payoff to \$600. So Pete will achieve the maximum *guaranteed* payoff by parking at the police station.

Now for Dave's decision. If he parks at the town hall, then Pete could park at the police station with payoff \$600, restricting his own payoff to $\$1200 - \$600 = \$600$. If Dave parks at the police station, then Pete could also park at the police station and realise a payoff of \$700, restricting his own payoff to \$500. Since Dave wants to keep Pete's payoff as small as possible so as to gain the most for himself, he will park at the town hall and guarantee a payoff of at most \$600 to Pete and therefore at least \$600 for himself.

We see that each vendor's action is clear. Pete should park at the police station and Dave should park at the town hall for a \$600 - \$600 split of the revenue. The analysis has led to a similar situation to Example 2, where, unlike that of Example 1, there is no advantage to any randomisation of choices.

8.2 Strictly Determined Games

In Examples 2 and 3 that we have just looked at, each player was able to reliably avoid the most unfavourable action of the other, any time the game was played. These games are examples of a type of game that has an especially simple solution.

125 Definition. *Let $A = (a_{ij})$ be the payoff matrix of a game. A game is said to be* ***strictly determined*** *if there is an entry a_{hk} that is simultaneously the minimum in its row and the maximum in its column. When this happens the number a_{hk} is called the* ***value of the game****, R_h is called the* ***optimal action (choice) of the row player****, and C_k the* ***optimal action (choice) of the column player****.*

It is easy to tell if a game is strictly determined and when it is to find the value of the game as well as the optimal actions for both players.

Optimal Action for the Row (Maximising) Player.

1. Find the smallest number in each row and write it to the right of the row.
2. Find the greatest of these row minima. If his number falls in the hth row, then R_h may be the optimal action for the row player.

Optimal Action for the Column (Minimising) Player

1. Find the largest number in each column and write it below the column.
2. Find the least of these column maxima. If this number falls in the kth column, then C_k may be the optimal action for the column player.

Course of Action

Case 1. If the greatest of these row minima is equal to the least of the column maxima, then these numbers occur in the same position. In this case the game is strictly determined with value a_{hk} and optimal actions R_h and C_k for the row and column player respectively.

Case 2. If the greatest of the row minima is *not* equal to the least of the column maxima, then the game is not strictly determined. We deal with this more interesting situation a little later.

Two more Examples

Example 4: Determine whether each of the games with payoff matrices below is, or is not, strictly determined. If it is strictly determined, find the value of the game and the optimal action for the players.

$$A = \begin{bmatrix} 2 & -1 & -5 \\ -3 & 1 & 5 \\ 4 & 2 & 4 \end{bmatrix}, \qquad B = \begin{bmatrix} -4 & 3 & 2 \\ 1 & -1 & 5 \\ -2 & 6 & -3 \end{bmatrix}$$

Solution: First we write A with each row minimum to the right of the row and each column maximum below the column. We have also bracketed the greatest of the row minima and the least of the column maxima.

$$A = \begin{bmatrix} 2 & -1 & -5 & -5 \\ -3 & 1 & 5 & -3 \\ 4 & 2 & 4 & (2) \\ 4 & (2) & 5 & \end{bmatrix}$$

The greatest of the row minima and the least of the column maxima are both equal to 2. Therefore, the game is strictly determined. This number occurs in row 3 and column 2, so R_3 and C_2 are the optimal actions for the row and column players respectively. The value of the game is $a_{32} = 2$.

We repeat the process for matrix B obtaining

$$B = \begin{bmatrix} -4 & 3 & 2 & -4 \\ 1 & -1 & 5 & (-1) \\ -2 & 6 & -3 & -3 \\ (1) & 6 & 5 & \end{bmatrix}$$

Here the greatest row minimum is -1. This is less than 1, the least column maximum. The game is not strictly determined.

Remark: When a game is strictly determined, the optimal action pair (R_h, C_k) is sometimes called a **saddle point** and the payoff, $a_{hk,}$ the saddle value. In this context, the row player's action R_h is called the **maximin** (i.e. maximise the minimums) and the column players action C_k is called the **minimax** (i.e. minimise the maximums) action. We give another application that shows that strictly determined games, while not suspenseful, are of some interest.

Example 5: The two banking companies, United Savings and State Savings, are vying for Golden Passbook accounts. Each company is in the process of deciding on a gift to use as an enticement to new customers. United will choose among a television set, cd player and video camera, whereas State will choose from among a video recorder, home computer and a video camera. Each knows the other's possible gifts and has knowledge of the market research table to be presented. This table shows the split (States vs United) of the potential 12,000 new customers under the various combinations.

		United		
		Television	CD Player	Videocam
State	Recorder	5000 – 7000	7000 – 5000	9000 – 3000
	Computer	6000 – 6000	8000 – 4000	7000 – 5000
	Videocam	5000 – 7000	6000 – 6000	4000 – 8000

Which gift should each give to insure the most new customers for their organisation?

Solution: We form the payoff matrix P with State as the row player, and the entries given in terms of thousands of accounts.

$$P = \begin{matrix} \begin{bmatrix} 5 & 7 & 9 \\ 6 & 8 & 7 \\ 5 & 6 & 4 \end{bmatrix} & \begin{matrix} 5 \\ (6) \\ 4 \end{matrix} \\ \begin{matrix} (6) & 8 & 9 \end{matrix} & \end{matrix}$$

The row minima are found, and the greatest of these is 6. The column maxima are found and the smallest of these is 6, and a_{21} is the matrix entry which is simultaneously the minimum in its row and the maximum in its column. The game is strictly determined with value 6, and R_2 and C_1 are the optimal actions. This means that State should offer a computer and United a television for a 6000–6000 split of the accounts.

8.3 Non-Strictly-Determined Games

Strategies and expected values

We start with an example somewhat similar to Example 1, but with simpler payoffs.

Example 6: This is the familiar game of coin matching that can be played by two players. Let us call the players Alice and Bill. If both players show heads or both tails, then Alice wins \$1. If one shows heads and the other tails, then Bill wins \$1. Set up the payoff matrix for this game and discuss the implications of the given strategies.

Solution: Let Alice be the row player and Bill the column player. Then the payoff for both heads or both tails is \$1. The payoff for either heads-tails combination is –\$1 since Alice will lose that amount. The payoff matrix is

$$\begin{array}{cc} & \text{Bill} \\ & \begin{array}{cc} \text{H} & \text{T} \end{array} \\ \text{Alice} \begin{array}{c} \text{H} \\ \text{T} \end{array} & \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \end{array}$$

This is not a strictly determined game, so we can't use the method from the previous section. We look carefully at the situation.

If either player shows the same side of the coin all the time, then the other player will catch on and win every time. Clearly, both should show each side of the coin some proportion of the time, in a random fashion. Let us examine a possible pair of strategies. Suppose that Alice decides to show heads 1/4 of the time and tails 3/4 of the time, whereas Bill settles on showing heads 1/8 of the time and tails 7/8 of the time. What would each player expect to gain over a series of plays?

First, we form a table showing the relative proportion of the time that each combination occurs. We have

Alice - Bill	**Relative proportion of the Time**
Heads - heads	$\frac{1}{4} \times \frac{1}{8} = \frac{1}{32}$
Heads - tails	$\frac{1}{4} \times \frac{7}{8} = \frac{7}{32}$
Tails - heads	$\frac{3}{4} \times \frac{1}{8} = \frac{3}{32}$
Tails - tails	$\frac{3}{4} \times \frac{7}{8} = \frac{21}{32}$

We see that the proportion of time that Alice would expect to win \$1 is

$$\frac{1}{32} + \frac{21}{32} = \frac{22}{32},$$

while the proportion of time she would expect to lose \$1 is

$$\frac{7}{32} + \frac{3}{32} = \frac{10}{32}.$$

Thus the expected payoff for Alice will be

$$\frac{22}{32}(+1) + \frac{10}{32}(-1) = \frac{12}{32}.$$

This means that, on the average, Alice would expect to win \$12 every 32 plays.

It would appear that Alice is playing heads and tails wisely. But perhaps this is not so. Let us see what happens when Bill changes his strategy. Suppose that Bill observes that his money is dwindling and decides that his strategy is backwards. So he decides to show heads 7/8 of the time and tails 1/8 of the time. This gives the following combination table:

Alice - Bill	**Relative proportion of the Time**
Heads - heads	$\frac{1}{4} \times \frac{7}{8} = \frac{7}{32}$
Heads - tails	$\frac{1}{4} \times \frac{1}{8} = \frac{1}{32}$
Tails - heads	$\frac{3}{4} \times \frac{7}{8} = \frac{21}{32}$
Tails - tails	$\frac{3}{4} \times \frac{1}{8} = \frac{3}{32}$

With these competing strategies we see that Alice would expect to win \$1

$$\frac{7}{32} + \frac{3}{32} = \frac{10}{32}.$$

of the time, and lose \$1

$$\frac{1}{32} + \frac{21}{32} = \frac{22}{32}.$$

of the time. Hence her expected payoff would be

$$\frac{10}{32}(+1) + \frac{22}{32}(-1) = \frac{-12}{32}.$$

This means that with this strategy Bill would expect to win \$12 every 32 plays.

In this second case Alice plays the same strategy as before, but the payoff is reversed. What it makes us ask is "What should Alice and Bill do to ensure their best payoff?" We will answer this question after introducing and demonstrating some relevant terminology and notation, and a little matrix algebra.

Let A_{mn} be the payoff matrix of a game and r_i the relative proportion of the time that the row player plays the ith action, that is, chooses row i. Then the collection $r_1, \ldots, r_i, \ldots, r_m$ is called a **strategy for the row player.** We denote the row player's strategy by R and represent it explicitly as the row vector

$$R = \begin{bmatrix} r_1 & \cdots & r_i & \cdots & r_m \end{bmatrix}.$$

In a similar way, if we make c_j the relative proportion of the time that the column player plays the jth action, then the collection $c_1, \ldots, c_j, \ldots, c_n$ is called a **strategy for**

the column player. We denote the strategy of the column player by C and represent it as a column vector

$$C = \begin{bmatrix} c_1 \\ \vdots \\ c_j \\ \vdots \\ c_n \end{bmatrix}.$$

An important characterisation of a strategy vector is that each entry is between 0 and 1, and the entries add to 1. Thus the strategy vectors can be interpreted as probability vectors where the ith row action is to be played with probability r_i, and the jth column action with probability c_j.

The actions must be chosen in some random fashion within the framework of the strategy, or else the pattern of play might be detected with disastrous resulting consequences. Some simple devices for random selection of an action might be used.

The term **pure strategy** is used when the actions are restricted to a single one. This was the case in the strictly determined games of the previous section. The ith row action R_i then becomes the ith pure strategy of the row player and is represented in vector form as

$$R_i = \begin{bmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}$$

where the 1 is in the ith position. The jth column action C_j becomes the jth pure action for the column player and is represented in vector form as

$$C_j = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

with a 1 in the jth position and 0 elsewhere. The term **mixed strategy** is often used to emphasise that more than one action may be employed.

We now consider the effect of one strategy pitted against another. If the row player makes the ith action with relative proportion r_i, and the column player makes the jth action with relative proportion c_j then the two occur together with relative proportion $r_i c_j$. Further since the r_i, c_j payoff is a_{ij} we call the product $r_i a_{ij} c_j$ the **expected payoff**

of the combined r_i, c_j actions. If the products are summed over all combinations of row and column actions, then we obtain the expected payoff or **expected value** for the strategy pair R , C of the game A. We write this

$$E_A(R,C) = \Sigma r_i a_{ij} c_j.$$

This is more easily written in matrix form as

$$E_A(R,C) = RAC.$$

R is a $1 \times m$ matrix, A is $m \times n$ and C is $n \times 1$, so the product is 1×1 which is treated as a real number. Here are some calculations of expected values:

Example 7: Consider the matrix A and the strategies given below. Find the expected value for each pair of row and column strategies.

$$A = \begin{bmatrix} 3 & -3 \\ -2 & 6 \\ 2 & -6 \end{bmatrix}, \quad R = \begin{bmatrix} 1/6 & 1/2 & 1/3 \end{bmatrix}, \quad R_2 = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}, \quad C = \begin{bmatrix} 6/7 \\ 1/7 \end{bmatrix}, \quad C_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Solution: The expected values are:

(a) $E_A(R,C) = RAC = \begin{bmatrix} 1/6 & 1/2 & 1/3 \end{bmatrix} \begin{bmatrix} 3 & -3 \\ -2 & 6 \\ 2 & -6 \end{bmatrix} \begin{bmatrix} 6/7 \\ 1/7 \end{bmatrix} = 9/42 = 3/14.$

With this strategy pair this game is slightly favourable to the row player.

(b) $E_A(R,C_1) = RAC_1 = \begin{bmatrix} 1/6 & 1/2 & 1/3 \end{bmatrix} \begin{bmatrix} 3 & -3 \\ -2 & 6 \\ 2 & -6 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1/6 = 7/42.$

This is still favourable to the row player, but slightly less so than the previous pair of strategies.

(c) $E_A(R_2,C) = R_2AC = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 3 & -3 \\ -2 & 6 \\ 2 & -6 \end{bmatrix} \begin{bmatrix} 6/7 \\ 1/7 \end{bmatrix} = -6/7.$

This pair of opposing strategies is favourable to the column player.

(d) $E_A(R_2,C_1) = R_2AC_1 = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 3 & -3 \\ -2 & 6 \\ 2 & -6 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = -2.$

This pair of strategies is even more favourable to the column player.

What is a 'solution' to a non-strictly-determined game?

The situation of the previous example leads us to ask whether there is a best strategy for each player, and before we can answer that we need some criterion to determine when a best strategy has been obtained.
Let A be the payoff matrix for a game and suppose that there are strategies R' and C' for the row and column players respectively, and a number v such that

$$R'AC \geq v \text{ for all strategies } C \text{ of the column player,} \tag{8.1}$$

and

$$RAC' \leq v \text{ for all strategies } R \text{ of the row player.} \tag{8.2}$$

Then v is called the **value** of the game, and R' and C' are **optimal strategies** for the row and column players respectively. The triple, (v, R', C') is a **solution** of the game.
If such a solution exists this means that the row player can find a strategy R' such that the expected value is at least v no matter what strategy the column player plays, and the column player can find a strategy C' such that the expected value is at most v no matter what strategy the row player uses. Each player can push the expected value to an equilibrium point.
An important theorem is:

126 Theorem. *Every matrix game has a solution.*

The problem is now to find the solution.

A game is called **fair** if its value is 0. For a fair game this means that the row player has a strategy R' and the column player a strategy C' such that

$$R'AC \geq 0 \text{ for all strategies } C \text{ of the column player,}$$

and

$$RAC' \leq 0 \text{ for all strategies } R \text{ of the row player.}$$

Here each player can find a strategy such that the other player is not expected to win anything, or that neither player has an advantage.

Guessing a Solution

For some simple games we can guess a solution and then verify that the guess is correct. We do this for the heads and tails game of Example 6 and show that the game is fair.

Example 6 completed: Recall that the payoff matrix is

$$\begin{array}{cc} & \text{Bill} \\ & \begin{array}{cc} \text{H} & \text{T} \end{array} \\ \text{Alice} \begin{array}{c} \text{H} \\ \text{T} \end{array} & \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \end{array}$$

From the symmetry of the game we can guess that the value of the game is $v = 0$ and that

$$R' = \begin{bmatrix} 1/2 & 1/2 \end{bmatrix}, \qquad C' = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}.$$

This strategy would mean that each player should show heads and tails with equal frequency. Let

$$R = \begin{bmatrix} r_1 & r_2 \end{bmatrix}, \qquad C = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}.$$

be arbitrary strategies of the row and column players respectively. Then

$$R'AC = \begin{bmatrix} 1/2 & 1/2 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = 0 \geq 0.$$

and

$$RAC' = \begin{bmatrix} r_1 & r_2 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} = \begin{bmatrix} r_1 & r_2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0 \leq 0.$$

This verifies that the guess is correct.

Finding the solution to a 2×2 game

We've learned how to identify the solution of a game, but not how to find such a solution. In general solving these games involves finding special solutions to the system of linear inequalities that occur in the definition of the solution of a game. We first find a formula for the solution of a 2×2.

To solve a 2×2 game we examine two systems of linear inequalities in the plane. The procedure is illustrated with an example but then a formula will be given for any non strictly determined 2×2 game.

We first give a theorem, which makes finding the solution to a game a little easier than is obvious from the definition.

127 Theorem. *Let A be the payoff matrix of a game. Further, let R' and C' be strategies of the row and column players respectively, and v be a number such that R', C' and v satisfy the inequalities*

$$R'AC \geq v \text{ for all pure strategies } C_i \text{ of the column player,} \tag{8.3}$$

and

$$RAC' \leq v \text{ for all pure strategies } R_j \text{ of the row player.} \tag{8.4}$$

Then (v, R', C') is a solution of the game.

This theorem says that when a player wants to check for optimality that player doesn't need to worry about *all* the strategies of the opposing player, but only the pure strategies. What is most important about this is that there is only a finite number of pure strategies.

Example 8: We solve the game

$$A = \begin{bmatrix} 2 & -3 \\ 1 & 5 \end{bmatrix}$$

A quick check shows that the game is not strictly determined.
Then, since the entries in the strategy vectors must add to 1 for each player, we can let the unknown optimal strategies be

$$R' = \begin{bmatrix} r & 1-r \end{bmatrix} \qquad \text{and} \qquad C' = \begin{bmatrix} c \\ 1-c \end{bmatrix}.$$

where $0 \leq r \leq 1$, and $0 \leq c \leq 1$. Let v be the unknown value of the game. Then, according to the definition of R', C' and v, playing optimal strategies against the possible pure strategies gives the following four inequalities:

$$R'AC_1 = \begin{bmatrix} r & 1-r \end{bmatrix} \begin{bmatrix} 2 & -3 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \geq v \tag{8.5}$$

$$R'AC_2 = \begin{bmatrix} r & 1-r \end{bmatrix} \begin{bmatrix} 2 & -3 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \geq v \tag{8.6}$$

and

$$R_1AC' = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & -3 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} c \\ 1-c \end{bmatrix} \leq v \tag{8.7}$$

$$R_2AC' = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & -3 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} c \\ 1-c \end{bmatrix} \leq v. \tag{8.8}$$

Moreover, by Theorem 127, if (R', C', v) satisfies these inequalities then it is indeed the solution we seek.

From the inequality 8.5 we get

$$2r + (1 - r) \geq v \qquad \text{or} \qquad r + 1 \geq v,$$

and from 8.6 we get

$$-3r + 5(1 - r) \geq 5 \qquad \text{or} \qquad -8r + 5 \geq v.$$

Figure 8.1 illustrate the area on the plane that satisfies both these inequalities, and the constraint that $0 \leq r \leq 1$. This area is called the **feasibility region**.

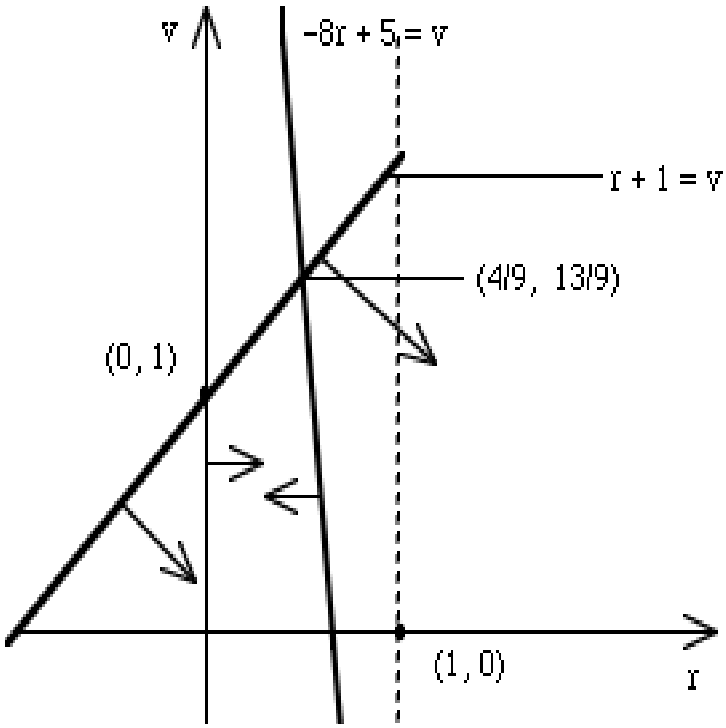


Figure 8.1: The feasibility region for the row strategy.

The greatest value of v that satisfies both constraints is given at the upper corner of the feasibility region. To find this point we can solve the system of equations given by the boundary lines

$$\begin{aligned} r + 1 &= v \\ -8r + 5 &= v. \end{aligned}$$

The point at the corner is given by $r = 4/9$ and $v = 13/9$. This means that the row strategy is

$$R' = \begin{bmatrix} 4/9 & 5/9 \end{bmatrix}$$

and v is at most 13/9.

We investigate the second group of matrix inequalities to determine C' and a lower bound for v. From the inequalities 8.7 and 8.8 we get

$$2c - 3(1-c) \leq v \qquad \text{or} \qquad 5c - 3 \leq v,$$

and

$$c + 5(1-c) \leq v \qquad \text{or} \qquad -4c + 5 \leq v.$$

The feasibility region for this system is shown in Figure 8.2.

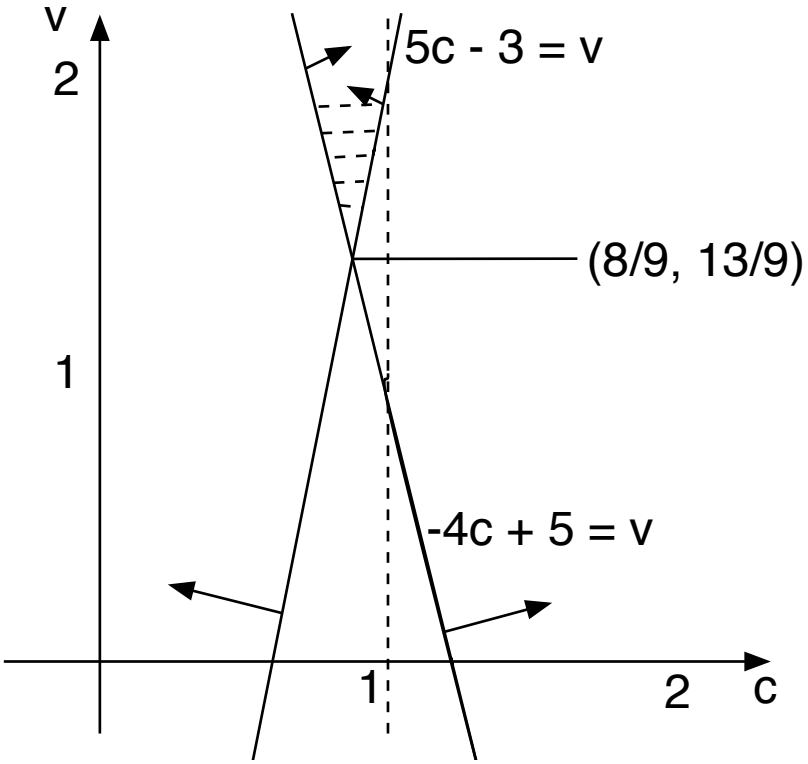


Figure 8.2: The feasibility region for the column strategy.

The least value of v that satisfies both constraints is given at the lower corner of the feasibility region. To find this we solve the system of equations given by the

boundary lines.

$$\begin{aligned} 5c-3 &= v \\ -4c+5 &= v. \end{aligned}$$

The point at the corner is given by $c = 8/9$ and $v = 13/9$. The column strategy is given by

$$C' = \begin{bmatrix} 8/9 \\ \\ 1/9 \end{bmatrix}$$

and v is at least 13/9.

From the two strategies we have $v \leq 13/9$ and $v \geq 13/9$, so we conclude that $v = 13/9$. This means that the game has been solved.

Using a similar method to that used in this example we can actually find a formula for the any 2×2 game that is not strictly determined. We first present a convenient criterion for deciding whether a 2×2 game is strictly determined or not.

128 Lemma. *The game*

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

is non strictly determined if and only if

$$a > b, a > c, d > b, d > c,$$

or

$$b > a, b > d, c > a, c > d.$$

This can be restated: A 2×2 game is not strictly determined if and only if both entries on one diagonal are greater than both entries on the other diagonal.

Proof. If either of the sets of inequalities holds, then none of the numbers can be the minimum in its row and the maximum in its column. Thus the game is not strictly determined.

Conversely, suppose that the game is non strictly determined. If $a = b$, it is a good exercise to show that the game is strictly determined. Similarly, if any two entries in the same row or the same column of A are equal, the game is strictly determined. So we can assume that no two entries in the same row or the same column are equal.

Therefore, $a > b$ or $b > a$. If $a > b$ it follows that $d > b$, or otherwise b is a row minimum and a column maximum. Then $d > c$, or else d is a row minimum and column maximum. Finally $a > c$ or else c is a row minimum and column maximum. This means that the first set of inequalities holds. If $b > a$ then the second set of inequalities must hold. □

We can now give the solution of a non strictly determined game.

129 Theorem. *Let*

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

be the payoff matrix for a non strictly determined game, and define

$$\begin{aligned} \det(A) &= ad - bc \quad \textit{(the determinant of A)} \quad \textit{and} \\ \operatorname{den}(A) &= (a+d) - (b+c) \quad \textit{(the denominator for the formulæbelow).} \end{aligned}$$

Then the solution is given by

$$\begin{aligned} v &= \frac{\det(A)}{\operatorname{den}(A)} \\ r' &= \frac{d-c}{\operatorname{den}(A)} \qquad R' = [r' \quad 1-r'] \\ c' &= \frac{d-b}{\operatorname{den}(A)} \qquad C' = \begin{bmatrix} c' \\ 1-c' \end{bmatrix} \end{aligned}$$

Proof. The formulas could be derived by geometric means as in the previous example, but since we have them we need only verify that they work.

First note that since the game is non strictly determined, Lemma 128 tells us that $\operatorname{den}(A) \neq 0$ and so R', C' and v are all well defined.

So all we need do is to verify each of the inequalities

$$R'AC_1 \geq v, \quad R'AC_2 \geq v, \quad R_1AC' \leq v, \quad R_2AC' \leq v.$$

In fact each of them is true by virtue of actually being an *equality*. We show this for the first of them and leave the others as an exercise.

In the first inequality we have.

$$\begin{aligned} R'AC_1 &= [r' \quad 1-r'] \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} \dfrac{d-c}{\text{den}(A)} & \dfrac{\text{den}(A)-(d-c)}{\text{den}(A)} \end{bmatrix} \begin{bmatrix} a \\ c \end{bmatrix} \\ &= \begin{bmatrix} \dfrac{a(d-c)+c(a-b)}{\text{den}(A)} \end{bmatrix} \qquad [\ \text{since } \text{den}(A) = (a-b)+(d-c)\ ] \\ &= \frac{ad-bc}{\text{den}(A)} = \frac{\det(A)}{\text{den}(A)} = v. \end{aligned}$$

□

Theorem 8.2 makes the solution of a 2×2 game very easy.

Example 9: Solve the game with payoff matrix $A = \begin{bmatrix} 5 & -6 \\ -2 & 3 \end{bmatrix}$.

Solution: In the matrix we note that 5 and 3 are both greater than -2 and -6. This means that the game is non strictly determined, and we can use the formulas from Theorem 129. We have

$$\begin{aligned} \det(A) &= ad-bc = 5\times 3-(-6)\times(-2) = 3, \\ \text{den}(A) &= a+d-b-c = 5+3-(-6)-(-2) = 16, \end{aligned}$$

and so from Theorem 129 we get the solutions

$$\begin{aligned} v &= \frac{\det(A)}{\text{den}(A)} = \frac{3}{16} \\ R' &= \begin{bmatrix} \dfrac{d-c}{\text{den}(A)} & \dfrac{a-b}{\text{den}(A)} \end{bmatrix} = \begin{bmatrix} \dfrac{3+2}{16} & \dfrac{5+6}{16} \end{bmatrix} = \begin{bmatrix} \dfrac{5}{16} & \dfrac{11}{16} \end{bmatrix} \\ C' &= \begin{bmatrix} \dfrac{3+6}{16} \\ \dfrac{5+2}{16} \end{bmatrix} = \begin{bmatrix} \dfrac{9}{16} \\ \dfrac{7}{16} \end{bmatrix}. \end{aligned}$$

Note that despite the fact that the sum (5+3) of the possible rewards to the row player is equal to that of the column player (6+2), the game favours the row player (since the payoff is positive).

Example 10 (Continuation of Example 1): We return to Alice and Bill's game where each can choose an Ace (a One) or a Two, and Alice pays Bill the sum of the card values if the sum is odd and *vice-versa* if the the sum is even. With Alice as row player the payoff matrix is $A = \begin{bmatrix} 2 & -3 \\ -3 & 4 \end{bmatrix}$, where playing Ace is choice 1 and playing Two is choice 2. As in Example 9, Lemma 128 tells us the game is non strictly determined so we can use the formulæ of Theorem 129 to solve the game. We find that $v = (2 \times 4 - -3 \times -3)/(2 + 4 - -3 - -3) = -1/12$, meaning that the game very slightly favours Bill (to the extent of one point per twelve games on average) when both players play optimally. The formulæ tell us that the optimum strategy for both Alice and Bill is to randomly play their Ace with probability $(4 - -3)/12 = 7/12$, and consequently to play their Two with probability $5/12$.

The next example considers a chase from historical fiction as a 2×2 matrix game. The incident is taken from *The Final Problem, Memoirs of Sherlock Holmes,* by Sir Arthur Conan Doyle, and was treated by Von Neumann and Morgenstern in *Theory of Games and Economic Behaviour.*

Example 11: Sherlock Holmes is leaving London on the Continental Express when he spies on the platform at Victoria Station, his evil nemesis, Professor Moriarty, a misdirected mathematician. Moriarty is bent on murder, and Holmes wants to avoid a confrontation until the following week when a trap for the entire gang will be sprung. Holmes allows that Moriarty, being equally intelligent as himself, will rent a special train and pursue.

The Continental stops only at Canterbury before speeding on to Dover for connection with the boat to France. Holmes must get off at Canterbury in an attempt to temporarily evade Moriarty, or go on to Dover in the hope of escaping to France. Moriarty may stop at Canterbury, missing the boat at Dover and risk losing Holmes altogether, or go on to Dover, chancing that Holmes is not in Canterbury.

Set the problem up as a matrix game and solve the game.

Solution: This is a version of the matching game with Moriarty being the one to match. We establish some point payoffs for the possibilities. If both stop at the same station we assign 100 points to Moriarty since he has achieved his purpose. If Holmes goes to Dover and Moriarty stops at Canterbury, then we shall give 80 points to Holmes since he can escape to France, but this means less to him

than the attempted murder means to Moriarty. If Holmes gets off at Canterbury and Moriarty goes on to Dover, we shall assign 20 points to Holmes since he has temporarily avoided the confrontation but is still in England. The payoff matrix is then

		Holmes	
		Canterbury	Dover
Moriarty	Canterbury	100	−80
	Dover	−20	100

As a game it is not strictly determined so we can use the formulæ from Theorem 129. We find that

$$v = \frac{10000-1600}{300} = \frac{8400}{300} = 28$$

$$R' = \begin{bmatrix} \frac{120}{300} & \frac{180}{300} \end{bmatrix} = \begin{bmatrix} \frac{2}{5} & \frac{3}{5} \end{bmatrix}$$

$$C' = \begin{bmatrix} \frac{180}{300} \\ \\ \frac{120}{300} \end{bmatrix} = \begin{bmatrix} \frac{3}{5} \\ \\ \frac{2}{5} \end{bmatrix}.$$

This is a mixed strategy solution for a game that is to be played only once. Here, we interpret $r_2 = 3/5$ to mean that Moriarty should go to Dover since $3/5 > 2/5$, and $c_1 = 3/5$ to mean that, on the same basis, Holmes should get off at Canterbury to achieve his aim of avoiding Moriarty.

This is, in fact, what the Conan Doyle characters do. So Holmes watches with satisfaction as Moriarty roars past the platform at Canterbury. Even thought the game is biased towards Moriarty ($v = 28$) the fact that the non strictly determined game is played with pure strategies makes Holmes the winner.

8.4 Using 'Dominance' to Solve some Games with $m > 2$ and/or $n > 2$.

Many games carry an element of chaff in their structure that is in the form of actions never played. Their presence may be used to conceal the true bias of the game, or to

present a threat that is never invoked. We can sometimes strip away these mathematically superfluous actions and reveal the true structure of the game.
Let A be the payoff matrix of a game and suppose that the h th row

$$\begin{bmatrix} a_{h1} & \cdots & a_{hj} & \cdots & a_{hn} \end{bmatrix}$$

and the kth row

$$\begin{bmatrix} a_{k1} & \cdots & a_{kj} & \cdots & a_{kn} \end{bmatrix}$$

have the property that each entry of row h is less than or equal to the corresponding entry of row k , that is,

$$a_{h1} \leq a_{k1}, \ldots, a_{hj} \leq a_{kj}, \ldots, a_{hn} \leq a_{kn}.$$

In these circumstances row h is said to be **dominated** by row k. In this situation the row player need never play the hth action in any optimal strategy because he could always do equally well by playing the kth action. This means that we can remove the h action from the game and row h from the payoff matrix without changing the value of the game.
In the same way suppose that

$$\begin{bmatrix} a_{1h} \\ \vdots \\ a_{ih} \\ \vdots \\ a_{mh} \end{bmatrix} \qquad \text{and} \qquad \begin{bmatrix} a_{1k} \\ \vdots \\ a_{ik} \\ \vdots \\ a_{mk} \end{bmatrix}$$

have the property that each entry in column h is less than or equal to the corresponding entry in column k , that is,

$$a_{1h} \leq a_{1k}, \ldots, a_{jh} \leq a_{jk}, \ldots, a_{mh} \leq a_{mk}.$$

In this situation column h is said to be **dominated** by column k. In this case the column player need never play the kth action in any optimal strategy because he could do at least as well by playing the hth action. Hence we may remove the kth action from the game and row k from the payoff matrix without altering the value of the game.

Example 12: Solve the game

$$A = \begin{bmatrix} 3 & -3 \\ -2 & 6 \\ 2 & -6 \end{bmatrix}$$

Solution: If we don't look too closely at this game we may think that the game might be fair since each positive number is matched with a negative one. However, on solving the game we see that this is not the case.

We haven't learnt to solve a 3×2 matrix game by direct technique. However, we can observe that row 1,

$$\begin{bmatrix} 3 & -3 \end{bmatrix}$$

dominates row 3,

$$\begin{bmatrix} 2 & -6 \end{bmatrix}.$$

We see that the row player would have a better payoff with the first row than with the third, no matter what action the column player takes. We can remove the third row and solve the game

$$A' = \begin{bmatrix} 3 & -3 \\ -2 & 6 \end{bmatrix}.$$

This new matrix makes it look as though the game is biased towards the row player. Using the formulas of Theorem 129 we find

$$\begin{aligned} \det(A) &= ad - bc = 3 \times 6 - (-3) \times (-2) = 12 \\ \text{den}(A) &= a + d - b - c = 3 + 6 + 3 + 2 = 14 \end{aligned}$$

so

$$\begin{aligned} v &= \frac{12}{14} = \frac{6}{7} \\ R' &= \begin{bmatrix} \frac{6+2}{14} & \frac{3+3}{14} \end{bmatrix} = \begin{bmatrix} \frac{4}{7} & \frac{3}{7} \end{bmatrix} \\ C' &= \begin{bmatrix} \frac{6+3}{14} \\ \\ \frac{3+2}{14} \end{bmatrix} = \begin{bmatrix} \frac{9}{14} \\ \\ \frac{5}{14} \end{bmatrix}. \end{aligned}$$

The solution to the original game is

$$\begin{aligned} v &= = \frac{6}{7} \\ R' &= \begin{bmatrix} \frac{4}{7} & \frac{3}{7} & 0 \end{bmatrix} \\ C' &= \begin{bmatrix} \frac{9}{14} \\ \\ \frac{5}{14} \end{bmatrix}. \end{aligned}$$

The third row disguised the fact that the game was not fair.

We close this section with an analysis of a simple 'sucker' game.

Example 13: A young traveller is taking a train ride in the country when a friendly stranger sits down and strikes up a conversation. After a time the stranger proposes the following card game to while away the hours.

The traveller is to be given a red 2, black 10 and a black ace, whereas the stranger will hold a red 2, black 10 and red ace. At a signal each person will show one card. The payoff is as follows:

If both show aces the play is a draw. If the stranger shows his ace against the two of the traveller the stranger wins \$4, but if the traveller shows his ace against the 2 of the stranger then the traveller wins \$8! Otherwise, if both show the same colour, the stranger wins \$1, whereas the traveller wins \$1 if the colours are different.

Set up the payoff matrix and find the solution of the game.

Solution: The payoff matrix is

$$\begin{array}{cc} & \textit{Traveller} \\ & \begin{array}{ccc} 2R & 10B & 1B \end{array} \\ \textit{Stranger} \begin{array}{c} 2R \\ 10B \\ 1R \end{array} & \begin{bmatrix} 1 & -1 & -8 \\ -1 & 1 & 1 \\ 4 & -1 & 0 \end{bmatrix} \end{array}$$

This payoff matrix appears to have a strong streak of generosity on the part of the stranger. However, let's look more carefully at the situation. We see that the

first row is dominated by the third row and can be removed. We get

$$\begin{array}{c} \\ 10B \\ 1R \end{array} \begin{array}{c} \begin{array}{ccc} 2R & 10B & 1B \end{array} \\ \begin{bmatrix} -1 & 1 & 1 \\ 4 & -1 & 0 \end{bmatrix} \end{array}.$$

Now we can also see that the second column is dominated by the third column. Therefore we can remove column three to get the 2×2 payoff matrix

$$\begin{array}{c} \\ 10B \\ 1R \end{array} \begin{array}{c} \begin{array}{cc} 2R & 10B \end{array} \\ \begin{bmatrix} -1 & 1 \\ 4 & -1 \end{bmatrix} \end{array}.$$

It now appears that the stranger is not so friendly. We use the formulas to find the solution:

$$\begin{aligned} v &= \frac{1-4}{-7} = \frac{3}{7} \\ R' &= \begin{bmatrix} \frac{-5}{-7} & \frac{-2}{-7} \end{bmatrix} = \begin{bmatrix} \frac{5}{7} & \frac{2}{7} \end{bmatrix} \\ C' &= \begin{bmatrix} \frac{-2}{-7} \\ \\ \frac{-5}{-7} \end{bmatrix} = \begin{bmatrix} \frac{2}{7} \\ \\ \frac{5}{7} \end{bmatrix}. \end{aligned}$$

The solution to the game as proposed is

$$\begin{aligned} v &= \frac{3}{7} \\ R' &= \begin{bmatrix} 0 & \frac{5}{7} & \frac{2}{7} \end{bmatrix} \\ C' &= \begin{bmatrix} \frac{2}{7} \\ \\ \frac{5}{7} \\ \\ 0 \end{bmatrix}. \end{aligned}$$

In the long run, the stranger would expect to win \$3 on every 7 plays. He would do this by playing his black 10 five sevenths of the time and his red ace the other

two sevenths of the time. Since he never plays his two the enticement payoff of \$8 will never occur. The traveller's best play is to play his red 2 two sevenths of the time and his black 10 the remaining five sevenths of the time. He should never play his ace in the hopes of a big payoff.

The very best thing for the traveller to do is to stay out of the game with the stranger!

There are of course games with payoff matrices that are larger than 2×2 matrices and which cannot be reduced, or can only be partially reduced, because of dominance. The best way to handle these matrices is to use an algebraic method based on the simplex algorithm. It is quite efficient and covers more general situations. However, a discussion of the simplex method is beyond what we wish to do here. Instead, we will extract a little more juice from the methods we already have.

8.5 Solving $2 \times n$ Matrix Games when Dominance is *Not* Present

It turns out that a $2 \times n$ game can be reduced to a 2×2 subgame by restricting the column players choices to just two. But which two? We illustrate with the next example, which extends the graphical method demonstrated in Example 8.

Example 14: Consider the 2×3 matrix game

$$G = \begin{bmatrix} 1 & -1 & 3 \\ 3 & 5 & -3 \end{bmatrix}.$$

Here A will be the row player and B the column player. The game is not strictly determined. As A has only two choices, suppose a strategy for A is determined by

$$R' = \begin{bmatrix} r & 1-r \end{bmatrix}.$$

Checking this against the pure strategies for B we get

$$R'GC_1 = \begin{bmatrix} r & 1-r \end{bmatrix} \begin{bmatrix} 1 & -1 & 3 \\ 3 & 5 & -3 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \geq v, \tag{8.9}$$

$$R'GC_2 = \begin{bmatrix} r & 1-r \end{bmatrix} \begin{bmatrix} 1 & -1 & 3 \\ 3 & 5 & -3 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \geq v, \tag{8.10}$$

$$R'GC_3 = \begin{bmatrix} r & 1-r \end{bmatrix} \begin{bmatrix} 1 & -1 & 3 \\ 3 & 5 & -3 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \geq v, \tag{8.11}$$

This gives the inequalities

$$r+3(1-r) = -2r+3 \geq v \tag{8.12}$$

$$-r+5(1-r) = -6r+5 \geq v \tag{8.13}$$

$$3r-3(1-r) = 6r-3 \geq v \tag{8.14}$$

where $0 \leq r \leq 1$ and we require v to be maximal. In the same way as for the 2×2 game we can draw a graph which summarises A's situation. This is shown in Figure 8.3 below.

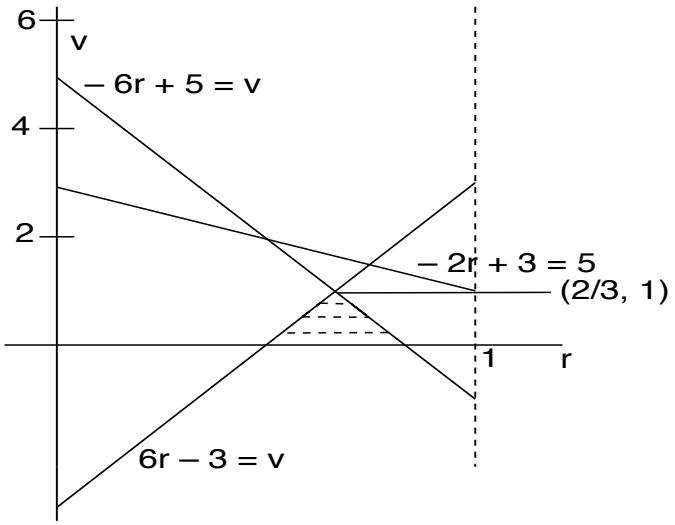


Figure 8.3: The feasible region for the row player.

Here the line $-2r+3=v$ represents the payoff to A as r varies from 0 to 1, assuming B uses only the strategy C_1. The line $-6r+5=v$ represents the payoff to A if B uses the strategy C_2, and the line $6r-3=v$ represents the payoff if B uses the strategy C_3. From the graph it is apparent that the strategy $\begin{bmatrix}2/3 & 1/3\end{bmatrix}$ is optimal for A and gives $v=1$.

To find B's optimal strategy we observe from the graph, that if A uses the strategy $\begin{bmatrix}2/3 & 1/3\end{bmatrix}$, the payoff for B using either C_2 or C_3 is $v=1$, but the payoff if B uses C_1 is $v=5/3$. Thus B stands to lose more than 1 if he employs strategy C_1. In this circumstance, where a pure strategy played against an opponent's optimal strategy yields less than the value of the game, we may ignore it. So we may ignore the C_1 strategy since it will appear with 0 probability in any optimal mixed strategy for B.

We therefore select from the original 2×3 game the 2×2 subgame in which B employs C_2 with probability c and C_3 with probability $1-c$, that is

$$C'=\begin{bmatrix}c\\1-c\end{bmatrix},$$

and the matrix is

$$\begin{bmatrix}-1 & 3\\5 & -3\end{bmatrix}.$$

We can use the formulas from Theorem 129 to get $v=1$ and $c=1/2$. Thus the optimal strategy for B in this subgame is $\begin{bmatrix}1/2\\1/2\end{bmatrix}$ and the strategy for B for the whole game is $\begin{bmatrix}0\\1/2\\1/2\end{bmatrix}$.

For the 2×3 game in the example above we needed to examine a diagram with three lines, one for each column of the matrix. The solution of the game involved finding the 'highest' point in the feasible region, which was the intersection of two of those three lines. For a $2\times n$ game the diagram will have n lines and similarly the solution probability r' and the value v of the game will always be obtainable as the coordinates of the intersection of just two of these n lines. So the values of r' and v can be calculated by solving the corresponding 2×2 subgame as we did in the example above. However

a complication arises if the relevant intersection point is a common intersection point for more than two of the lines. While this does not affect the calculation of v or r', it does affect the calculation of c'. We illustrate this in our final example below.

Example 15: The game

$$G = \begin{bmatrix} 1 & 2 & 4 & 0 \\ 0 & -2 & -3 & 2 \end{bmatrix}$$

has the graph shown in Figure 8.4 below.

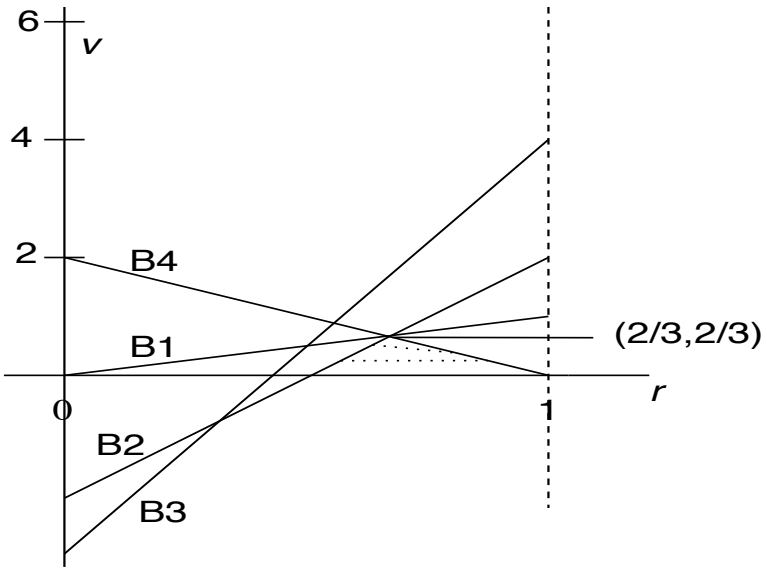


Figure 8.4: The feasible region for the row player.

The line $B1$ comes from the inequality $R'GC_1 \geq v$ and is the line $r = v$. The line $B2$ comes from the inequality $R'GC_2 \geq v$ and is the line $4r - 2 = v$. The line $B3$ comes from the inequality $R'GC_3 \geq v$ and is the line $7r - 3 = v$. Finally, the line $B4$ comes from the inequality $R'GC_4 \geq v$ and is the line $3 - 2r = v$. Each line represents the payoff to A when B plays the corresponding pure strategy C_i, $i = 1,2,3,4$. In this game an optimal strategy for A is $\begin{bmatrix} 2/3 & 1/3 \end{bmatrix}$ and $v = 2/3$. This information is not very helpful to B. The point $r = 2/3, v = 2/3$ lies on the intersection of the three lines $BI, B2$ and $B4$. We can therefore ignore $B3$ because the payoff against A's optimal strategy is 5/3, a worse yield for B than the value of the game. Hence three possible 2×2 subgames arise, one for each pair of $BI, B2$ and $B4$. Perhaps surprisingly, it turns out that these three subgames do *not* all have the same solution.

The game involving BI and $B2$ has the matrix $\begin{bmatrix} 1 & 2 \\ 0 & -2 \end{bmatrix}$. This is the matrix of a strictly determined game, with value $v' = 1$. Since this is not the value of the original 2×4 game we don't need to examine its solution. So it's only necessary to solve two 2×2 subgames. They are

$$\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \qquad \text{and} \qquad \begin{bmatrix} 2 & 0 \\ -2 & 2 \end{bmatrix}.$$

The value of each of these games is $v = 2/3$, which is the same as the value of the original 2×4 game and the first gives an optimal strategy for B as $C_1' = \begin{bmatrix} 2/3 \\ 1/3 \end{bmatrix}$, while the second gives an optimal strategy for B as $C_2' = \begin{bmatrix} 1/3 \\ 2/3 \end{bmatrix}$. Putting these in the context of the original game we get two optimal strategies for B:

$$C_1^\star = \begin{bmatrix} 2/3 \\ 0 \\ 0 \\ 1/3 \end{bmatrix} \qquad \text{and} \qquad C_2^\star = \begin{bmatrix} 0 \\ 1/3 \\ 0 \\ 2/3 \end{bmatrix}.$$

From here we ask whether B can use any other mixed strategy in optimal play? The answer is yes, for we can verify that any combination $\lambda C_1^\star + (1-\lambda)C_2^\star$ with $0 \le \lambda \le 1$ (this is called a **convex combination**), is also an optimal strategy for B. Firstly, note that

$$C = \lambda C_1^\star + (1-\lambda)C_2^\star = \begin{bmatrix} \frac{2}{3}\lambda \\ \frac{1}{3}(1-\lambda) \\ 0 \\ \frac{2}{3} - \frac{1}{3}\lambda \end{bmatrix}.$$

To show that this is an optimal strategy for B we need to show that

$$R_1 GC = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 & 0 \\ 0 & -2 & -3 & 2 \end{bmatrix} \begin{bmatrix} \frac{2}{3}\lambda \\ \frac{1}{3}(1-\lambda) \\ 0 \\ \frac{2}{3} - \frac{1}{3}\lambda \end{bmatrix} \ge v \quad (= \frac{2}{3})$$

and

$$R_2 GC = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 & 0 \\ 0 & -2 & -3 & 2 \end{bmatrix} \begin{bmatrix} \frac{2}{3}\lambda \\ \frac{1}{3}(1-\lambda) \\ 0 \\ \frac{2}{3} - \frac{1}{3}\lambda \end{bmatrix} \ge v \quad (= \frac{2}{3})$$

In the first case we get

$$\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 & 0 \\ 0 & -2 & -3 & 2 \end{bmatrix} \begin{bmatrix} \frac{2}{3}\lambda \\ \frac{1}{3}(1-\lambda) \\ 0 \\ \frac{2}{3}-\frac{1}{3}\lambda \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 & 0 \end{bmatrix} \begin{bmatrix} \frac{2}{3}\lambda \\ \frac{1}{3}(1-\lambda) \\ 0 \\ \frac{2}{3}-\frac{1}{3}\lambda \end{bmatrix}$$

$$= \frac{2}{3}\lambda + \frac{2}{3}(1-\lambda)$$

$$= \frac{2}{3}$$

We get a similar result in the second case. So we conclude that we have found one optimal strategy for the row player, and an infinite number of optimal strategies for the column player.

END OF CHAPTER EIGHT

9. COMBINATORIAL GAMES

Combinatorial games are two-person games with perfect information and no chance moves, and with a win-or-lose outcome. Familiar games of this sort are Chess, Go, Checkers, Noughts and Crosses, Dots-and-Boxes and Nim. Such games can be played perfectly in the sense that either one player can force a win or both can force a draw. In reality, games like Chess and Go are too complex to find an optimal strategy, and they derive their attraction from the fact that it is not known (yet) how to play them perfectly. We will however, learn to play Nim perfectly.
We study combinatorial games because they have a rich and interesting mathematical theory. We will focus on the central role of the game of Nim for impartial games. This is nontrivial mathematics, and it should be fun, and something you have not seen before.

130 Definition. *A **non random perfect information** game is a game in which there is no element of chance in the play and nothing is hidden from the players.*

Examples

1. **Go-moku**

   This is another game in the Noughts and Crosses family. This game is of Japanese origin and can be described as "five in a row". The game may be played on a Go board which is equivalent to a 19×19 array. Instead of 0's and $\times$'s black and white stones are put on the intersections of the lines. By convention black goes first. Of course the game can be played on a checkerboard, or with 0's and $\times$'s. It can be quite complicated, even though it is a perfect information game. It is a game of win, lose or draw.

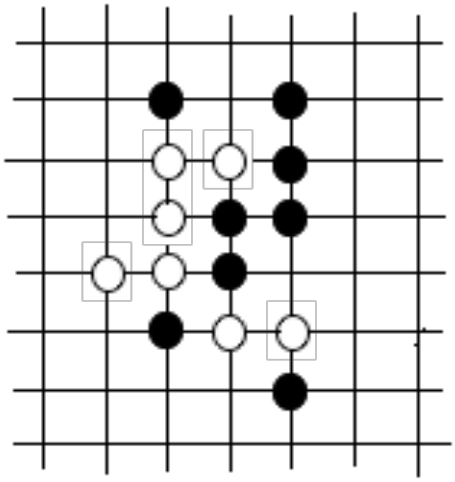


Figure 9.1: Go- moku, White to play.

2. **The Game of Ten**

   The **game of ten** is sometimes taught to children when they are learning to add. Alice goes first and chooses the number 1 or 2. Bill then adds 1 or 2 to it and obtains an answer. The A adds 1 or 2 again, and the game continues in this way. The first person to reach 10 wins. A position is simply given by an integer between 0 and 10. It is not too hard to work out a strategy for this game, and there can be many variations.

3. **Backgammon**

   Backgammon is a two-person, zero-sum game. Since it uses a dice a random element enters the game. It can still be considered as a perfect information game, as it is a game of skill.

Card games are usually not perfect information games, as most of the time certain cards are hidden. One of the challenges of a zero-sum, two-person, non random perfect information game is that it is a battle of wits. All the information is available and there are no chance elements to surprise the players. We analyse some games of this type that are played sequentially.

131 Definition. *An **impartial combinatorial game** is a game that satisfies the following conditions.*

1. *There are two players.*
2. *There is a set, usually finite, of possible positions of the game, and sometimes a particular starting position.*
3. *There are clearly defined rules that specify the moves that either player can make from the given position to possible new positions.*
4. *The players alternate in making moves.*
5. *In the **normal play** convention a player who is unable to move loses.*
6. *The rules are such that play will always come to an end. This is called an **ending condition**.*
7. *There is complete information.*
8. *There are no chance moves such as rolling dice or shuffling cards.*
9. *The game is **impartial** in the sense that the possible moves of a player depend only on the position but not on the player.*

As an alternative to condition 5, there is also the **misère play** convention under which the last player to move loses. In contrast to condition 9, games where the available moves depend on the player are called **partisan** games. The game Chess is a partisan game because one player can only move the white pieces and the other the black pieces. We do not consider partisan games, because the analysis becomes too complex, even for something like noughts and crosses. Chess and noughts and crosses also fail condition 5 since they may end in a tie or draw.

9.1 Games and Digraphs

We can equivalently describe an impartial combinatorial game as a game played on a directed graph. We represent the positions of the game as **vertices** of a graph **G**. If it is possible to move from position P_1 to position P_2, we draw a directed edge from P_1 to P_2. Insisting that the game end after a finite number of moves means that there are **no cycles** on **G**. We usually assume that the game is finite and that there are only finitely many positions.

132 Definition. *An **acyclic** digraph is a digraph with no cycles on it other than the trivial one vertex ones. A **game graph G** is a finite, directed, acyclic graph. A vertex with out-degree zero is called an **end vertex**.*

We illustrate the game of ten in Figure 9.2. It is quite easy to picture since it has only 11 positions.

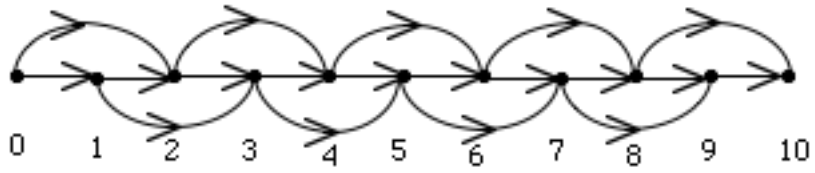


Figure 9.2: The Game of Ten.

It is interesting to note that even a simple game like Noughts and Crosses has a very large graph. If we take as possible positions any configuration of 0's and $\times$'s and blanks in a 3 by 3 array, there are $3^9 = 19,683$ positions. Not all these positions are used as the rules of the game require that there be either an equal number of 0's and $\times$'s, or one more $\times$ than 0. This reduces the number of possible positions, but it is still very large. For more complicated games the figures get very large indeed.

9.2 Win/Lose Games and Strategic Labelling

We want to consider games that can be pictured as directed graphs and if possible, determine a strategy for winning.

Example 1

Let G be the game with graph shown in Figure 9.3. Player A starts at vertex v and moves to one of three choices. Player B then moves according to the possibilities allowed. The players move alternately. The first person who has *no* moves to make *loses*. The other person wins. Thus, players A and B each want to move into one of the positions marked **X**. The game is at position P and it is player A 's turn. What should she do?

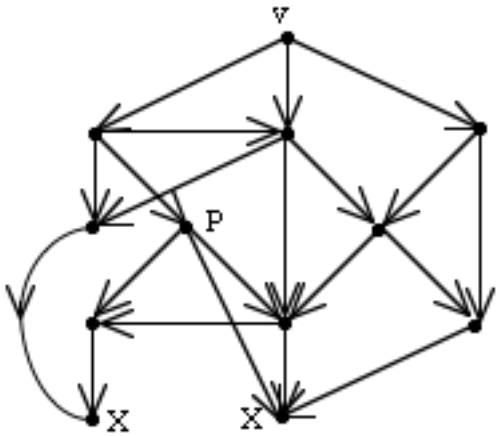


Figure 9.3: Example 1

Method: A possible strategy is to identify each position as a winning or a losing position. You can be in a winning position and end up losing the game if you don't play well. If you are in a losing position, then whatever you do you must move into a winning position - for your opponent - and then you will lose if your opponent plays well. If you are in a winning position your goal is to move into a *losing* position, because it will be your opponents turn. This means that the positions marked **X** are losing positions. You want to move into one of them so that your opponent will not be able to move.

We will mark each of the vertices with a 0 or a 1: 0 for a losing position and 1 for a winning position. We want the following to be true for a labelling.

Rule 1: *All moves* from a 0-vertex go to a 1-vertex.

Rule 2: *There is always a move* from a 1-vertex to a 0-vertex.

The method is to work backwards from the known losing positions to find winning positions and then to discover new winning or losing positions until all the positions are labelled. We use the rules just stated and start labelling the end vertices with 0. We call this set of vertices V_0. We then find the set V_1 of all the vertices from which the next move **must** end in V_0, and we label these as 1-vertices. We now find V_2 the set of all the vertices from which the next move must end in V_0 or V_1. We label these according to the Rules 1 and 2. We continue in this way until all vertices are labelled. Figure 9.4 is helpful in this labelling process.

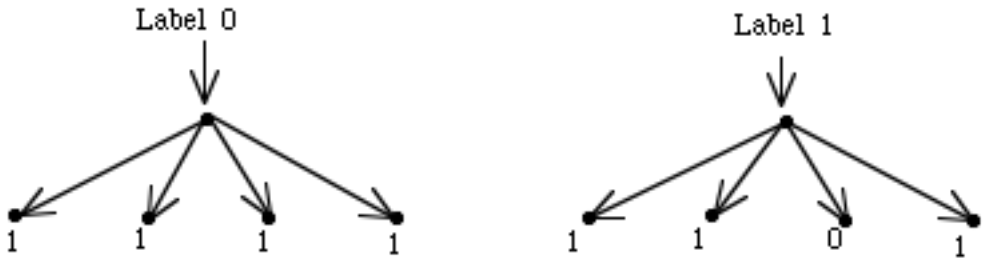


Figure 9.4: The labelling process.

In this way we systematically find all winning and losing positions by working backwards from the known losing positions. Figure 9.5 gives a complete labelling of the game in Figure 9.3.

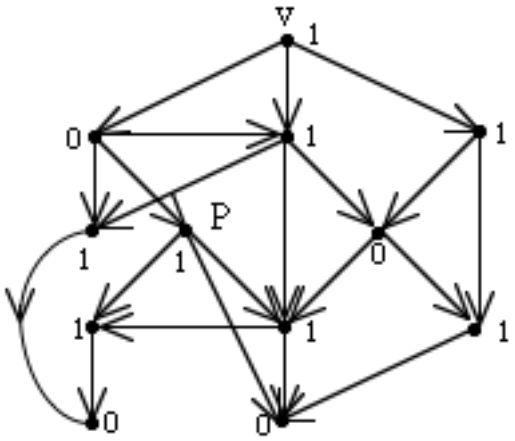


Figure 9.5: A strategic Labelling.

The rule to win is that you must always move to a 0 vertex, if this is possible. If you start at a 1-vertex this will always be possible. In the game above the first player is at a 1-vertex and can move to 0-vertex. Hence the first player can always win if he or she plays correctly.

From here we define a strategic labelling for any game graph. We almost seem to be assuming that such a labelling always exists when we make this definition. We will consider this question later.

133 Definition. *A **labelling** of a game graph **G** is a function f from the vertices of **G** into $\{0,1\}$, where $f(v)$ is called the **label** of v. If $f(v) = 0$, we say that v is a 0-vertex, and similarly for 1-vertices. A **strategic labelling** of **G** is a labelling such that*

1. *All edges with an initial 0-vertex have a terminal 1-vertex.*
2. *If v is a 1-vertex, there is an edge with initial vertex v whose terminal vertex is a 0-vertex.*

Example 2

We give a strategic labelling for the game of ten. If you think about the game you will realise that if you get to 7 then you must win. With a little further thinking you will realise that 4 is a pretty good spot to aim for, as is 1. When we follow the procedure given above we get the labelling shown in Figure 9.6.

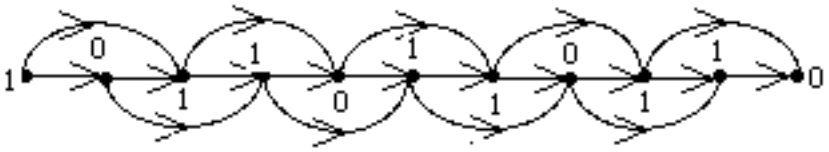


Figure 9.6: A strategic labelling for the game of ten.

Now we can think about the question as to whether a game graph always has a strategic labelling. We need the following definition.

134 Definition. *A **lower set L of** vertices in a game graph G is a set of vertices with the property that if an edge of G has its initial vertex in L, then its terminal vertex must be in L also.*

In the graph G below (Figure 9.7) the initial vertex is a, and l and m are end vertices. Some lower sets for G are $\{l\},\{m\},\{l,m\},\{k,l,m\},\{j,k,l,m\}$. Note that $\{j,l,m\}$ is not a lower set of vertices for G because the vertex j has an edge that terminates at k and $k \notin \{j,l,m\}$. Another lower set is $\{e,f,g,i,j,k,l,m\}$.

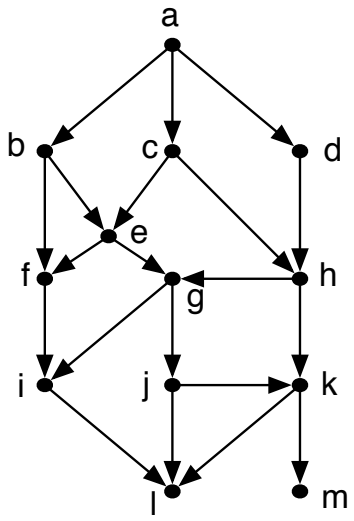


Figure 9.7: A game graph G

There are three important issues for the theorem below.

1. The first is that the set T of end vertices always makes a lower set. In the example above, we have $T = \{l, m\}$.

2. The second is that if we have a lower set L for a graph G, and $L \neq G$ then it is possible to find a vertex $v \in G \setminus L$ with the property that $\{v\} \cup L$ is a lower set for G. For the graph G above, take $L = \{e, f, g, i, j, k, l, m\}$. Then we can add either vertex b or vertex h to L to obtain a new lower set L'.

3. The third is that G itself satisfies the definition for a lower set.

135 Theorem. *Any game graph **G** has a strategic labelling.*

Proof. We show by induction that any lower set of G has a strategic labelling. We begin with the lower set T that consists of all the end vertices of G. We label these with 0, and this clearly provides a strategic labelling for T. We assume as the *inductive hypothesis* that any lower set L with, say n vertices can be strategically labelled. If $L \neq G$, we show how to extend the strategic labelling to one more vertex.
Suppose L is a lower set where $L \neq G$. By the inductive hypothesis, we assume that L has a strategic labelling. We claim there is a vertex $v \in G \setminus L$ such that $v \cup L$ is a lower set of G. We call such a vertex v *suitable*. We find a suitable vertex as follows.

Choose any vertex $v_0 \in G \setminus L$. If v_0 is not suitable, there is an edge starting at v_0 whose terminal vertex v_1 is also not in L. If v_1 is not suitable, there is an edge starting at v_1 whose terminal vertex v_2 is not in L. This process continues indefinitely if there are no suitable vertices. But there are no cycles in a game graph, and the graph is finite, so the process cannot continue indefinitely.
This means that at some point we will find a suitable vertex v. We use conditions 1 and 2 of Definition 133 to decide the labelling of the suitable vertex v. When we label v accordingly we have a strategic labelling for the lower set $\{v\} \cup L$, which has $n+1$ vertices. By the principle of induction this means that we can label any lower set with a strategic labelling. As the game graph G is itself a lower set, this means that we can label any game graph G with a strategic labelling.

□

We analyse a game using strategic labelling. This game shows the advantage of using this technique.

9.3 Examples Using Strategic Labeling

Wythoff's Game

Wythoff's game is played with two piles of checkers. Player A and player B alternate moves according to the following rules. Any player can remove any positive number of checkers from either pile, including the whole pile, or an equal number from each pile. The last person to make a move - and remove everything - wins.
Suppose that one pile has 12 checkers and the other 15. What move should you make to ensure that you win? Is it possible to make a move that will ensure you will win?
Method: We look for a strategic labelling of the positions. In the above situation you would hope that you are at a 1-vertex, so that you can move to a 0-vertex. But a labelling is required.
Let us represent the piles $(m,\ n)$ as a point on the plane, or perhaps a checker on a checker board.
The rules are geometrically stated as moving left, down or diagonally down to the left. Let us suppose the game is at position $(13,9)$ with 13 representing the number of checkers in the left hand pile, and 9 the number in the right hand pile. If we remove 5 checkers from the left hand pile, we move to position $(8,9)$. This corresponds to moving left 5 units from the coordinate $(13,9)$ to the coordinate $(8,9)$. Removing 4 checkers from the right hand pile now takes us to position $(8,5)$ which corresponds to

a move down 4 units. Finally if we take 3 checkers from each pile we move to position $(5,2)$ and this corresponds to moving diagonally to the left for 3 units. These moves are illustrated in Figure 9.8.

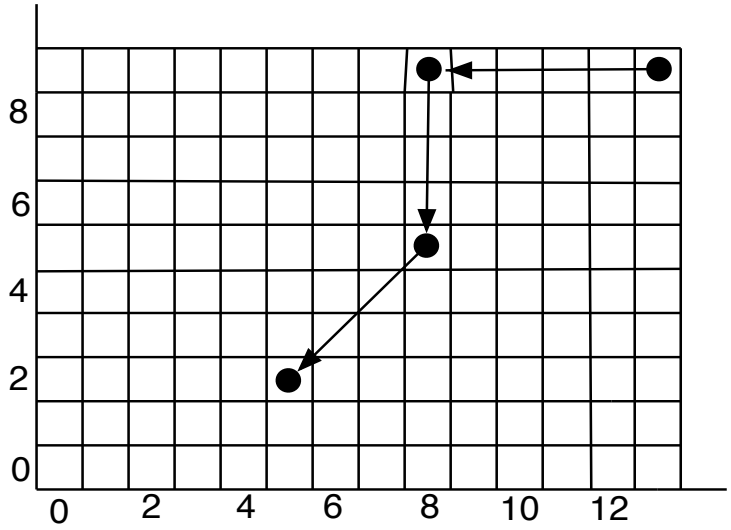


Figure 9.8: Analysis of moves for Wythoff's game .

We start labelling the positions on the checker board. It is important to remember that each square represents a position of the game (a vertex if we drew the game as a graph). If we look at a particular square (m,n), it is possible to move to that position from any square above it, from any square to the left of it, or from any square diagonally (to the right) above it. If we represented all these moves by arrows the diagram would have so many arrows that it would be totally confusing. So we don't put in the arrows, but we need to keep them in mind.

The end vertex is the $(0,0)$ square as the person who makes the last legal move (the move to $(0,0)$) is the winner. We label $(0,0)$ as a 0-vertex. All vertices that can get to this position in 1 move are going to be 1-vertices. These are the vertices directly above the (0, 0) position, directly to the right of the (0, 0) position, and the vertices in the diagonal position from the (0, 0) position.

Consider now the (1,2) position. From it the only possible moves are to the (0, 2) position, the (1,1) position or diagonally to the (0, 1) position. As these are all 1-vertices, we label (1,2) with a 0, as it is a losing vertex. This is shown in Figure 9.9. A similar argument holds for the (2,1) position.

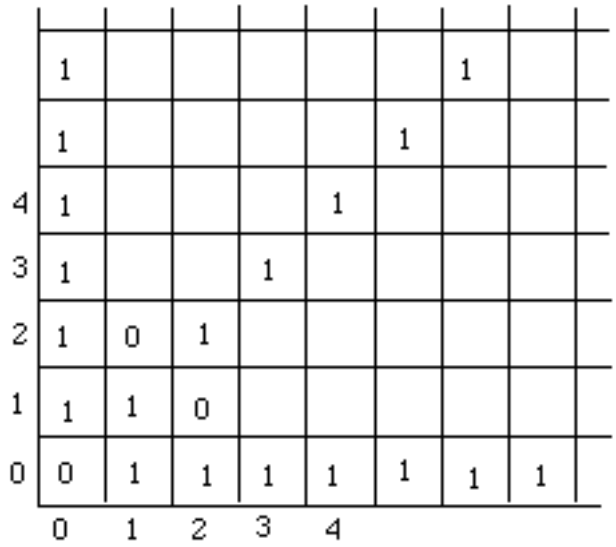


Figure 9.9: Beginning the labelling.

From here we can label all the vertices, by working backwards and using the rules we have already stated. We could go on indefinitely, but that is not very helpful. Perhaps by completing a little more of the graph we can work out precisely which of the vertices are 0-vertices.

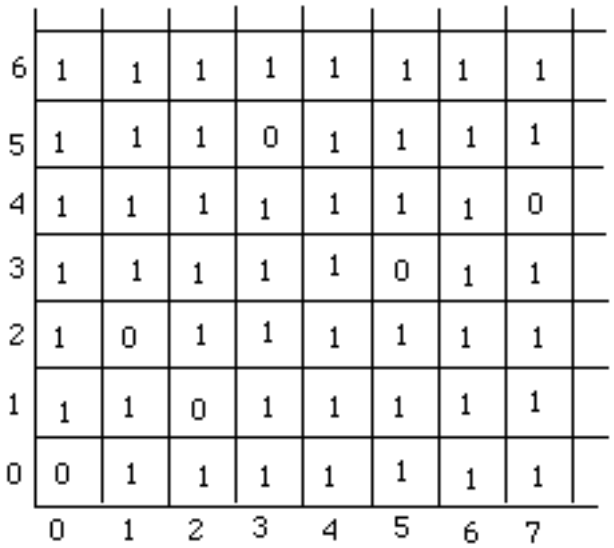


Figure 9.10: A complete labelling for this region.

This is possibly enough to read off the 0-vertices. The situation is quite symmetric so the piles (m,n) are labelled in the same way as (n,m). So only the 0-vertices (m,n) where $m \leq n$ are given.

$$(0,0),(1,2),(3,5),(4,7),(6,10),(8,13),(9,15),\ldots$$

Perhaps you can find a pattern for these numbers?
Let's recall the initial problem. The game is at position (12, 15) and it is your turn. What is your move? From the list give above we can see that this is a 1-vertex and for you to win you want to move to a 0-vertex, using a legal move. It's really quite easy. If you move 3 from the first pile you will get to the (9, 15) position which is a losing position (for your opponent). If you continue to play well, you will win the game.

Nim

Nim is an ancient game, similar to Wythoff's game. It is played with several piles of checkers and players must remove a positive number of checkers (possibly all) from any one pile of his or her choosing. Play alternates and the player who picks up the last checker wins.
As there would usually be more than two piles we can't easily use a 2-dimensional diagram to analyse the game. However, the game is perhaps simpler than Wythoff's as the possible moves are more restricted.
Method: The 2-pile game is quite easy to play. The strategy is to level the piles, if possible, and then do in one pile whatever your opponent does in the other. Using an analysis similar to that for Wythoff's game it is easy to see that the 0-positions are the positions (n,n) for any $n \geq 0$. So if the piles are even and it is your turn you possibly won't win, as this copycat strategy is pretty easy to see. What happens when there are more than two piles?
Suppose that there are 3 piles and that these contain 3, 3 and 7 checkers. What is the best play? The strategy is to write these numbers in binary notation and to add each column, doing the addition modulo 2. This is equivalent to counting the number of 1's in each column. If there is an odd number of 1's in a column, put a 1 beneath the column; if there is an even number of 1's put a 0 below that column.

3			1	1		3			1	1
3			1	1	$\longrightarrow$	3			1	1
7		1	1	1		0		0	0	0
7		1	1	1		0			0	0

The *strategy* is to remove checkers from one of the piles so that each column sum is zero. Such positions are the 0-positions. The easiest (and only) move here is to change the 7 to 0. This then leaves you in the 2-pile game and it is easy to finish from there.
Let's try another example. Suppose that the piles are 5, 6 and 8. We follow the same procedure:

5			1	0	1	5			1	0	1
6			1	1	0	6			1	1	0
9		1	0	0	1	3				1	1
10		1	0	1	0	0			0	0	0

If we change the 9-pile (1001) to 3 (11 in binary notation) we get the required situation. Thus (5, 6, 3) is a 0-position.
The strategic labelling in Nim can be defined as follows: write the numbers in each pile in binary notation (underneath each other). If each column sum is zero, the position is a 0-position. Otherwise, it is a 1-position.
If you would like to try your hand against the computer have a look at the website
`http://www.gamedesign.jp/flash/nim/nim.html`
The strategy is fairly easy to state, but perhaps we need to think about why? Any 0-position must lead to a 1-position. If the game is at a 0-position, which means all the column sums add to an even number, and some checkers are removed from one pile. This means that the binary number representing that pile is lessened and in that process at least one of the 1's must change to a 0. Hence the even sum in that column becomes odd.
From any 1-position it is possible to move to a 0-position. Choose the column furthest to the left and take any of the piles with a 1 in that position. Change this to a 0 and change the rest of the bits so that all column sums become even. In this way it is possible to change from a 1-position to a 0-position.

9.4 Nim Addition

The addition described above is known as *Nim addition*. If a and b are two non-negative integers, their **Nim sum** is another integer which we will denote by $a \oplus b$. In the addition described above: $3 \oplus 3 \oplus 7 = 7$, where 7 is the binary number represented by the sum of the columns. Also $5 \oplus 6 \oplus 9 = 10$ where again 10 is the binary number represented by the sum of the columns.
Nim addition shares some of the properties of ordinary addition:

$$\begin{aligned} a \oplus 0 &= 0 \oplus a = a \\ a \oplus b &= b \oplus a \\ (a \oplus b) \oplus c &= a \oplus (b \oplus c), \end{aligned}$$

where a, b and c are non-negative integers. However, Nim addition also has some unusual properties:

$$a \oplus a = 0,$$

for every non-negative integer a. Also

$$4 \oplus 8 = 12, \quad 1 \oplus 4 = 5, \quad 3 \oplus 5 = 6, \quad 5 \oplus 3 \oplus 7 \oplus 3 \oplus 1 = 3.$$

With a little investigation you will find that the Nim sum of two *different* powers of 2 is the same as their ordinary sum. So for example: $16 \oplus 32 = 48$. This property, together with the properties listed above give an alternative method for calculating any Nim sum for two integers a and b. To use this method for calculating a Nim sum, write each number as a sum of distinct powers of 2. Then use the commutative and associative laws to cancel any repetitions in pairs. For example:

$$11 \oplus 16 \oplus 18 = (8+2+1) \oplus 16 \oplus (16+2) = (8 \oplus 2 \oplus 1) \oplus 16 \oplus (16 \oplus 2) = 8 \oplus 1 = 8+1 = 9.$$

A short Nim addition table is given below:

Table 9.1: Nim addition

$\oplus$	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
4	3	2	1	0	7	6	5	4
5	5	4	7	6	1	0	3	2
6	6	7	4	7	2	3	0	1
7	7	6	5	4	3	2	1	0

9.5 Grundy Labelling

We introduce **Grundy numbers** and **Grundy labelling**. A Grundy labelling is a generalisation of a strategic labelling and is helpful in playing Nim like games. Each vertex of a game graph is assigned a non negative integer n, called its Grundy number and the vertex is called an n -vertex. The definition is recursive. We first define the *mex* or Minimum Excludant.

136 Definition. *Let A be a set of non-negative integers. Then $mex(A)$ is the smallest non-negative integer not included in A.*

Some examples are

1. for $A = \{0,1,2,4,5\}$, $mex(A) = 3$,
2. for $B = \{1,2,4,6,7\}$, $mex(B) = 0$,
3. for $C = \{0,1,2,3,4\}$, $mex(C) = 5$.

137 Definition. *The vertices of a game graph are labelled with Grundy numbers in the following way:*
The end vertices are all 0-vertices. If v is not an end vertex, and if the possible moves from v lead to vertices $v_1, v_2, \ldots, v_k$, with Grundy numbers $g_1, g_2, \ldots, g_k$, then the Grundy number of v is $mex(A)$ where $A = \{g_1, g_2, \ldots, g_k\}$.

In a similar way to the proof of Theorem 135 we can show that any game graph G has a Grundy labelling. We would show that this is true by showing that it is true for the lower sets of G and then use induction. I leave this for you to do.

9.6 Examples of Grundy Labelling

Example 1: A small game graph

We make a Grundy labelling of the game graph given in Figure 9.11

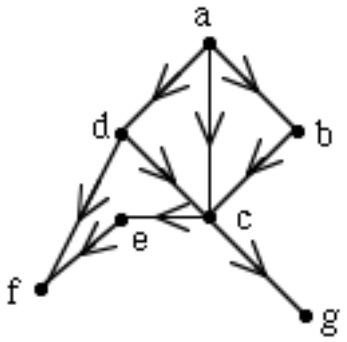


Figure 9.11: Example 1

It is clear that vertices f and g are the end vertices and so we label those 0. The next vertex that can be labelled is vertex e. Vertex e has only one arrow going from it and that arrow goes to vertex f. Since vertex f is labelled with 0, $A_e = \{0\}$ and $mex(A_e) = 1$. Hence e will be labelled with a 1.
It is now possible to label vertex c. The possible moves from vertex c go to e and g with Grundy numbers 1 and 0 respectively. hence $A_c = \{0,1\}$ and $mex(A_c) = 2$, so c is labelled with a 2. The remaining vertices are labelled as follows as in Table 9.2 below. The Grundy labelling for the game is shown in Figure 9.12

Table 9.2: Finding the Grundy Labelling for Example 1

Vertex x	possible moves to	A_x	Grundy label
f,g	end vertices		0
e	f	$A_e = \{0\}$	1
c	e,g	$A_c = \{0,1\}$	2
d	f,c	$A_d = \{0,2\}$	1
b	c	$A_b = \{2\}$	0
a	b,c,d	$A_a = \{0,1,2\}$	3

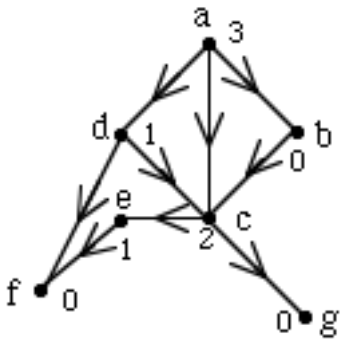


Figure 9.12: Grundy Labelling for Example 1

Example 2: The Game of Ten

We find the Grundy labelling for the graph for the game of Ten.

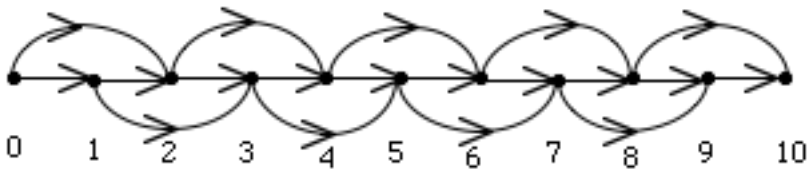


Figure 9.13: Game graph for Example 2: Game of Ten.

Method: We again start from the end vertex which is the 10 vertex. Its Grundy number will be 0. The next vertex it is possible to consider is the 9 vertex, and the only possible move is to the 10 vertex which has Grundy number 0. So $A_9 = \{0\}$ and $mex(A_9) = 1$. Hence vertex 9 has Grundy number 1.
It is possible now to consider vertex 8. The possible moves from the 8 vertex are to 9 and 10, so $A_8 = \{0,1\}$ and $mex(A_8) = 2$. Hence vertex 8 has Grundy number 2.
Vertex 7 allows possible moves to vertices 8 and 9, so $A_7 = \{1,2\}$, $mex(A_7) = 0$ and vertex 7 has Grundy number 0. This establishes the pattern and the Grundy labelling for this graph is shown in Figure 9.14.

Example 3: A single Nim pile

Consider the digraph P_n whose vertices are the integers from 0 through to n, with an edge from x to y if $x > y$. This is the graph of the relation $xRy \leftrightarrow x > y$.

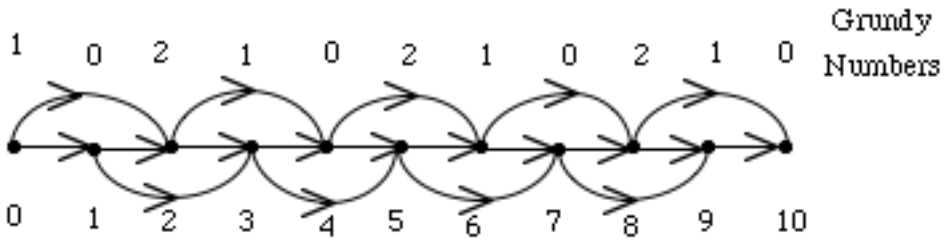


Figure 9.14: Grundy Labelling for Game of Ten.

The graph P_4 is shown in Figure 9.15 below.

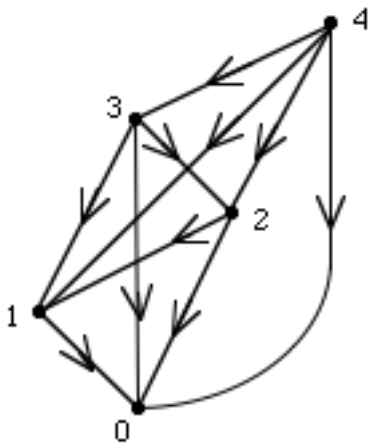


Figure 9.15: Graph for Example 3: P_4

We can see that P_4 is a game graph as it is a finite directed graph with no cycles. If we work out the Grundy numbers for each vertex of P_4 we find that vertex i has Grundy number i. The same is true for the digraph P_n. In fact we can think of P_n as a single Nim pile with n checkers. This further means that a single Nim pile of size n has Grundy number n.

Parallel Games

Example 3 above is quite helpful to us as it gives us a means of understanding a game that is played on several different game graphs at the same time. Suppose that a game is played on k game graphs $G_1, G_2, \ldots, G_k$ as follows. Each digraph has a checker at one of its vertices. A player can play on any digraph and move the checker there using a legal move. Play alternates and the player who has no more moves loses. Each graph

represents a game in its own right, so we could call this "playing games in parallel". We can use Grundy numbers to find the strategy for this game of parallel games. We find the Grundy number of the position in each of the component games, and then treat these numbers as Nim piles. By using the strategy of Nim we decide which number to reduce. We can then go to the corresponding component game and move to a position with that Grundy number. By the definition of Grundy numbers that will always be possible. The principle will be illustrated in the next section.

9.7 Hackenbush

We discuss a relative of Nim, the game **Hackenbush.** The game is played with pencil and paper and a drawing like the one in Figure 9.16.

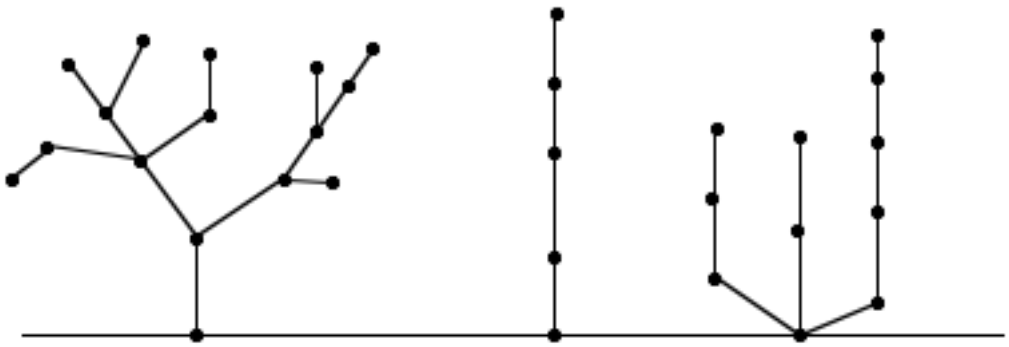


Figure 9.16: An example of a Hackenbush game.

The game consists of a horizontal line, called the **ground**, and a number of trees (simple undirected graphs with no circuits), each connected to the ground by a single **ground vertex**. The legal move is to *chop* one edge of one of the trees, which then causes that edge and all other edges no longer connected to the ground to disappear. The game ends when some player is unable to move because all the edges have disappeared.

How can we find a good move from the position of Figure 9.16 ? Well, the game can be thought of as playing several one-tree Hakenbush games in parallel. At our turn, each tree, in its current state, represents a position in the one-tree game played on its unpruned original, and so has a Grundy number. If we can compute this Grundy number for each of our current trees, we can play the game as a game of Nim.

There is a simple rule for computing the current Grundy value of a Hackenbush tree. It is best explained by first considering two very simple kinds of trees called **snakes** and **bushes** as in Figure 9.22.

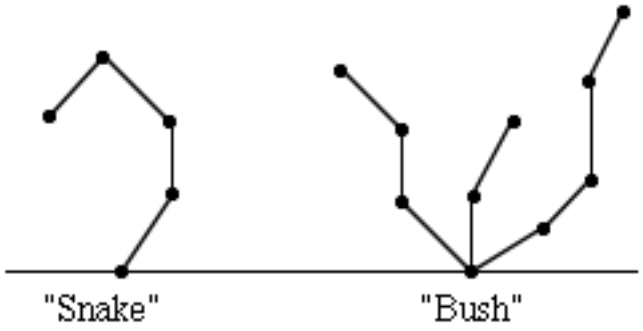


Figure 9.17: Snakes and Bushes.

A snake is just a chain of edges with one edge touching the ground, and a bush is just a number of snakes joined together at the ground. A Hackenbush snake is really just the same as a Nim pile of the same size as the number of edges. So the Grundy value of a snake is just the number of edges. A bush isn't much harder, since despite the fact that the snakes are connected at the ground, players chop the snakes independently, so we can disconnect them and treat each snake as a Nim pile. For the situation in Figure 9.22 we get the equivalence shown in Figure 9.18

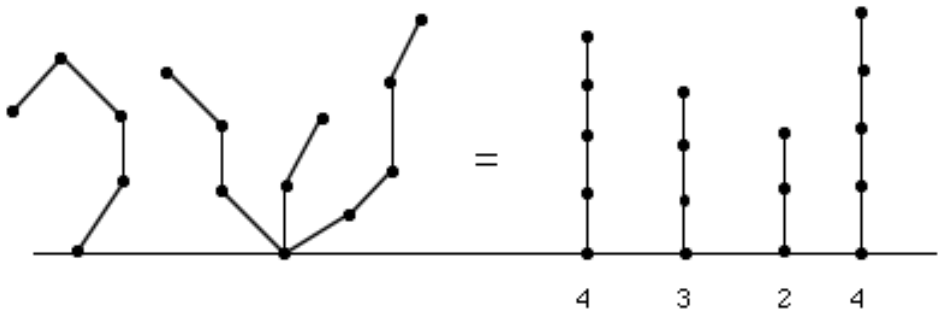


Figure 9.18: Nim equivalence

So this is the same as having Nim piles of 4, 3, 2 and 4. We then write these is binary notation and add the columns without carrying:

$$\begin{array}{cccc} 4 & 1 & 0 & 0 \\ 3 & & 1 & 1 \\ 2 & & 1 & 0 \\ 4 & 1 & 0 & 0 \\ \hline & 0 & 0 & 1 \end{array}$$

To make all the columns add to zero we may reduce the second snake by 1, that is to cut off the top edge of the second snake. We will than have snakes of sizes 4, 2, 2 and 4, which is a losing position and one with an easy strategy.
It turns out that knowing about snakes and bushes we can compute the Grundy value of any tree. Consider, for example, the tree in Figure 9.19. By inspecting this tree carefully we see that it consists of several bushes "glued together".

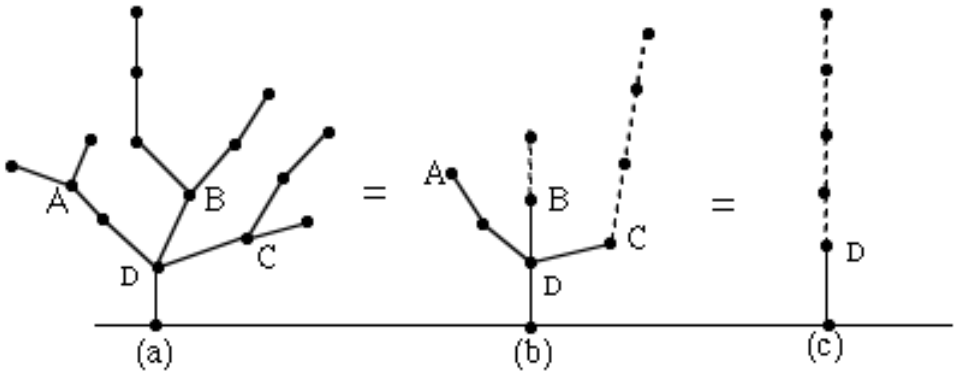


Figure 9.19: From a bush to a snake.

There is a bush of two snakes of length 1 growing from vertex A. This bush is equivalent to a snake of length $1 \oplus 1 = 0$, so it is replaced by this snake of length zero! The bush growing from vertex B is equivalent to a snake of length $3 \oplus 2 = 1$ so we replace it by a snake of length 1. Finally the bush at C is equivalent to a snake of length $2 \oplus 1 = 3$ and we replace it by a snake of length 3. This leads us to part (b) of the diagram in Figure 9.19. From here we now look at vertex D, and in the same way the bush growing from vertex D is equivalent to a snake of length $2 \oplus 2 \oplus 4 = 4$. Replacing the bush with the equivalent snake we get the tree of Figure 9.19(c). This is just a snake of length 5 and has Grundy value 5.

Instead of redrawing the tree as we have done in Figure 9.19 we can keep track of the book-keeping without erasing anything. We attach a number to each edge, which represents the Grundy value of that part of the tree consisting of the given edge and all edges *above* it. We call these numbers the *edge labels* and they can be calculated from the top down. We start by labelling each of the edges in the three bushes growing at vertices A, B and C. Each of these bushes consists of a number of snakes, and the edges in a snake are labelled 1, 2, 3,... , starting at the top. This is illustrated in Figure 9.20(a).

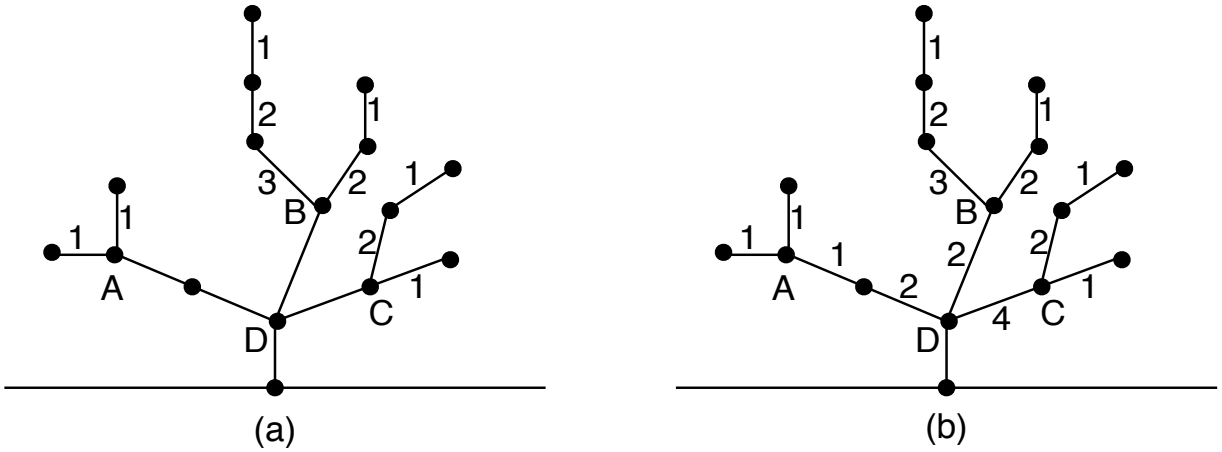


Figure 9.20: Finding the Grundy values.

Next, we assign labels to the edges just below the vertices A, B and C. These values are equal to *one more* than the length of the (equivalent) snake above the corresponding vertex. So at the edge below A we give the label $(1 \oplus 1) + 1 = 0 + 1 = 1$. At the edge below B we give the label $(3 \oplus 2) + 1 = 1 + 1 = 2$, and at the edge below C we put the label $(2 \oplus 1) + 1 = 3 + 1 = 4$. Now we can label the vertex two below A with 1 more than the edge above it, that is with a 2. This is shown in Figure 9.20 (b)

At D the edges above it now have the labels 2, 2 and 4. We label the edge below D with the label $(2 \oplus 2 \oplus 4) + 1 = 5$. This final label is the Nim value of the original tree. We now find a good move, if there is one, from the Hackenbush position of Figure 9.16 redrawn with numbering as Figure 9.21 below. .

Each of the component trees is labelled in the way we have just described and the Grundy values of the component trees are 5, 4 and 4. Note that the second 4 comes

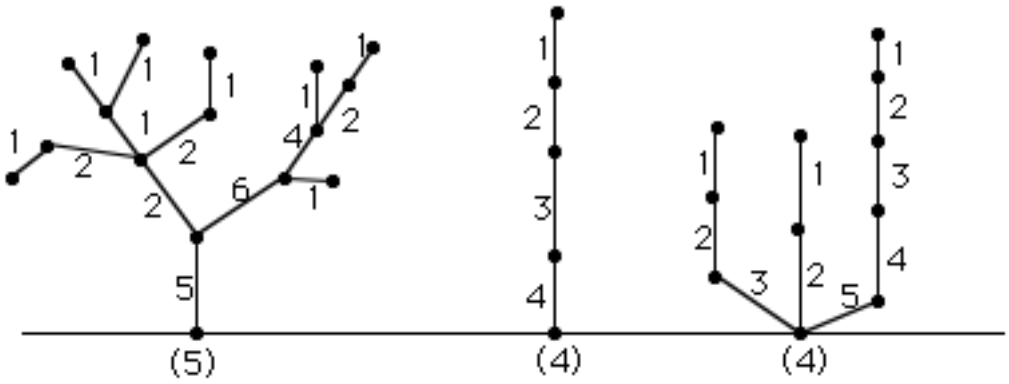


Figure 9.21: Grundy values for Figure 9.16.

as the Grundy sum: $3 \oplus 2 \oplus 5 = 4$. If we use our Nim strategy now we write these numbers in binary notation and add the columns modulo 2:

$$\begin{array}{cccc} 5 & 1 & 0 & 1 \\ 4 & 1 & 0 & 0 \\ 4 & 1 & 0 & 0 \\ \hline & 1 & 0 & 1 \end{array}$$

We want all these columns to be zero and the way to do this is to remove the pile (tree) of size 5. This means we cut off the tree of Grundy number 5 right at the bottom and we are left with the two trees of Grundy numbers 4 and 4. We might note that we have made the Grundy sum of the component trees equal to 0.

Let's see where we might go from here. Suppose the next player removes the top branch from the snake of length 4, so that it has length 3. See figure 9.22 below. What is your next move?

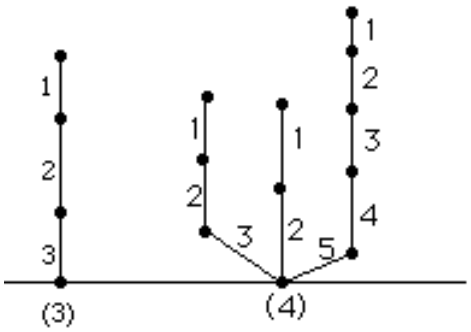


Figure 9.22: What is the next move?

We again use Nim strategy and write the Grundy numbers in binary notation.

$$\begin{array}{cccc} 3 & & 1 & 1 \\ 4 & 1 & 0 & 0 \\ \hline & 1 & 1 & 1 \end{array}$$

Each of the columns adds (mod 2) to 1, and the only way to make them all 0 is to cut the tree with Grundy number 4 in such a way that it will have Grundy number 3. We recall that the Grundy number 4 comes as the sum: $3 \oplus 2 \oplus 5 = 4$. We need to be able to chop something off one of the three snakes in this bush so that the Grundy sum of the three components will be 3. After some experimentation we find that $3 \oplus 2 \oplus 2 = 3$ so that if we chop the top 3 edges from the snake of length 5 the whole bush will have Grundy number 3:

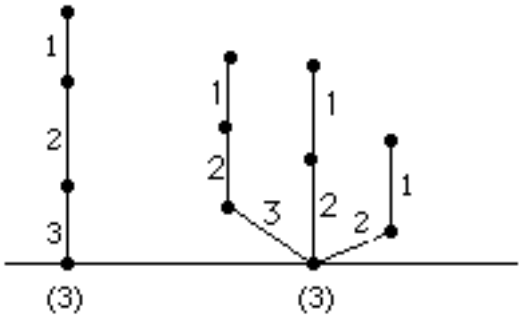


Figure 9.23: Position of the game after our second move.

It is a good exercise to follow this game through to the end. The Nim strategy is easy now that there are only two trees, but applying it may take some thought.

END OF CHAPTER NINE