Algorithmic Problem Solving

May 28, 2008

Contents

Chapter 1

Introduction

In historical terms, the digital computer is very, very new. The science of computing is yet newer. Compared to its older sister —mathematics— which is thousands of years old, it is hardly in the embryonic stage of development. Yet, computing science is already having a major influence on our problem-solving skills, amounting to a revolution in the art of effective reasoning.

Because of the challenges of *programming* (which means instructing a dumb machine how to solve each instance of a problem) and the unprecedented *scale* of programming problems, computing scientists have had to hone their problem-solving skills to a very fine degree. This has led to advances in logic, and to changes in the way that mathematics is practised. These lectures form an introduction to problem-solving using the insights that have been gained in computing science.

1.1 Algorithms

Solutions to programming problems are formulated as so-called *algorithms*. An algorithm is a well-defined procedure, consisting of a number of instructions, that are executed in turn in order to solve the given problem.

A concrete example may help to understand better the nature of algorithms and their relation to problem solving. Consider the following problem, which is typical of some of the exercises we discuss. You may want to tackle the problem before reading further.

Four people wish to cross a bridge. It is dark, and it is necessary to use a torch when crossing the bridge, but they only have one torch between them. The bridge is narrow and only two people can be on it at any one time. The four people take different amounts of time to cross the bridge; when two cross together they must proceed at the speed of the slowest. The first person takes 1 minute to cross, the second 2 minutes, the third 5 minutes and the fourth 10 minutes. The torch must be ferried back and forth across the bridge, so that it is always carried when the bridge is crossed.

Show that all four can cross the bridge within 17 minutes.

The solution to this problem —which we won't disclose just yet!— is clearly a sequence of instructions about how to get all four people across the bridge. A typical instruction will be: "persons x and y cross the bridge" or "person z crosses the bridge". The sequence of instructions solves the problem if the total time taken to execute the instructions is (no more than) 17 minutes.

An algorithm is typically more general than this. Normally, an algorithm will have certain *inputs*; for each input, the algorithm should compute an *output* which is related to the input by a certain so-called *input-output relation*. In the case of the bridge-crossing problem, an algorithm might input four numbers, the crossing time for each person, and output the total time needed to get all four across the bridge. For example, if the input is the numbers 1, 3, 19, 20, the output should be 30 and if the input is the numbers 1, 4, 5, 6 the output should be 17. The input values are called the *parameters* of the algorithm.

Formulating an algorithm makes problem-solving decidedly harder, because it is necessary to formulate very clearly and precisely the procedure for solving the problem. The more general the problem, the harder it gets. (For instance, the bridge-crossing problem can be generalised by allowing the number of people to be variable.) The advantage, however, is a much greater understanding of the solution. The process of formulating an algorithm demands a full understanding of *why* the algorithm is correct.

The key to effective problem-solving is economy of thought and of expression — the avoidance of unnecessary detail and complexity. The mastery of complexity is especially important in computing science because of the unprecedented size of computer programs: a typical computer program will have hundreds, thousands or even millions lines of code. Coupled with the unforgiving nature of digital computers, whereby a single error can cause an entire system to abruptly "crash", it is perhaps not so surprising that the challenges of algorithm design have had an immense impact on our problem-solving skills.

This book aims to impart these new skills and insights to a broad audience, using an *example-driven* approach. It aims to demonstrate the importance of mathematical calculation, but the chosen examples are typically not mathematical; instead, like the bridge-crossing problem above, they are problems that are readily understood by a lay person, with only elementary mathematical knowledge. The book also aims to challenge; most of the problems are quite difficult, at least to the untrained or poorly trained practitioner.

1.2 Bibliographic Remarks

I first found the bridge problem in [Lev03]. Rote [Rot02] gives a comprehensive bibliography. The problem is also known as the “flashlight” problem and the “U2” problem; it is reputed to be used by at least one major software company in interviews for new employees.

Chapter 2

Invariants

"Invariant" means "not changing". An invariant of some process is thus some attribute or property of the process that doesn't change. Other names for "invariant" are "constant", "law" and "pattern".

The recognition of invariants is an important problem-solving skill, possibly the most important. This chapter introduces the notion of an invariant, and discusses a number of examples of its use.

We begin as we mean to go on. We first present a number of problems for you to tackle. Some you may find easy, but others you may find difficult or even impossible to solve. If you can't solve one, move on to the next. To gain full benefit, however, it is important that you try the problems first, before reading further.

We then return to each of the problems individually. The first problem we discuss in detail, showing how an invariant is used to solve the problem. Along the way, we introduce some basic skills related to computer programming — the use of assignment statements, and how to reason about assignments. The second problem, which otherwise would be quite hard, is now straightforward. We leave it to you to solve, but, because the techniques are new, we suggest a sequence of steps which lead directly to the solution. The third problem is quite easy, but involves a new concept, which we discuss in detail. Then, it is your turn again. From a proper understanding of the solution to these initial problems, you should be able to solve the next couple of problems. This process is repeated as the problems get harder; we demonstrate how to solve one problem, and then leave you to solve some more. You should find them much easier to solve.

1. **Chocolate Bars.**

   A rectangular chocolate bar is divided into squares by horizontal and vertical grooves, in the usual way. It is to be cut into individual squares. A cut is made by taking a single piece and cutting along one of the grooves. (Thus each cut splits one piece into two pieces.)

Figure 2.1 shows a 4×3 chocolate bar that has been cut into five pieces. The cuts are indicated by solid lines.

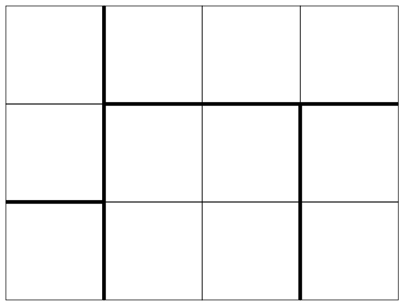


Figure 2.1: Chocolate-Bar Problem.

How many cuts in total are needed to completely cut the chocolate into all its pieces?

2. **Empty Boxes.**

   Eleven large empty boxes are placed on a table. An unknown number of the boxes is selected and, into each, eight medium boxes are placed. An unknown number of the medium boxes is selected and, into each, eight small boxes are placed.

   At the end of this process there are 102 empty boxes. How many boxes are there in total?

3. **Tumblers.**

   Several tumblers are placed in a line on a table. Some tumblers are upside down, some are the right way up. (See fig. 2.2.) It is required to turn all the tumblers the right way up. However, the tumblers may not be turned individually; an allowed move is to turn any *two* tumblers simultaneously.

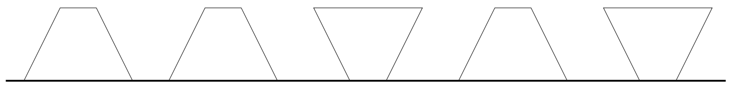


Figure 2.2: Tumbler Problem.

   From which initial states of the tumblers is it possible to turn all the tumblers the right way up?

4. **Black and White Balls**

   Consider an urn filled with a number of balls each of which is either black or white. There are also enough balls outside the urn to play the following game. We want to reduce the number of balls in the urn to one by repeating the following process as often as necessary.

Take any two balls out of the urn. If both have the same colour, throw them away, but put another black ball into the urn; if they have different colours then return the white one to the urn and throw the black one away.

Each execution of the above process reduces the number of balls in the urn by one; when only one ball is left the game is over. What, if anything, can be said about the colour of the final ball in the urn in relation to the original number of black balls and white balls?

5. **Dominoes**

   A chess board has had its top-right and bottom-left squares removed so that there are 62 squares remaining. (See fig. 2.3.) An unlimited supply of dominoes has

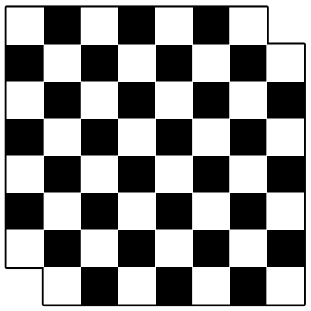


Figure 2.3: Mutilated Chess Board

been provided; each domino will cover exactly two squares of the chessboard. Is it possible to cover all 62 squares of the chessboard with the dominoes without any domino overlapping another domino or sticking out beyond the edges of the board?

6. **Tetrominoes**

   A *tetromino* is a figure made from 4 squares of the same size. There are five different tetrominoes, called the O-, Z-, L-, T- and I-tetrominoes. (See fig. 2.4.)

   The following exercises all concern covering a rectangular board with tetrominoes. Assume that the board is made up of squares of the same size as the ones used to make the tetrominoes. Overlapping tetrominoes or tetrominoes that stick out from the sides of the board are not allowed.

   (a) Suppose a rectangular board is covered with tetrominoes. Show that at least one side of the rectangle has an even number of squares.

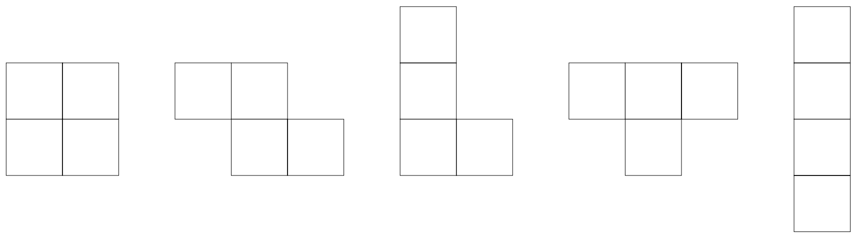


Figure 2.4: O-, Z-, L-, T- and I-tetromino

(b) Suppose a rectangular board can be covered with T-tetrominoes. Show that the number of squares is a multiple of 8.

(c) Suppose a rectangular board can be covered with L-tetrominoes. Show that the number of squares is a multiple of 8.

(d) An $8{\times}8$ board cannot be covered with one O-tetromino and fifteen L-tetrominoes. Why not?

2.1 Chocolate Bars

Recall the problem statement:

> A rectangular chocolate bar is divided into squares by horizontal and vertical grooves, in the usual way. It is to be cut into individual squares. A cut is made by taking a single piece and cutting along one of the grooves. (Thus each cut splits one piece into two pieces.)
>
> How many cuts in total are needed to completely cut the chocolate into all its pieces?

2.1.1 The Solution

The solution to the chocolate-bar problem is as follows. Whenever a cut is made, the number of cuts increases by one, and the number of pieces increases by one. Thus, the number of cuts and the number of pieces both change. What *doesn't* change, however, is the *difference* between the number of cuts and the number of pieces. This is an "invariant", or a "constant", of the process of cutting the chocolate bar.

Now, we begin with one piece and zero cuts. So, the difference between the number of pieces and the number of cuts, at the outset, is one. It being a constant means that it will always be one, no matter how many cuts have been made. That is, the number of pieces will always be one more than the number of cuts. Equivalently, the number of cuts will always be one less than the number of pieces.

We conclude that to cut the chocolate bar into all its individual pieces, the number of cuts needed is one less than the number of pieces.

2.1.2 The Mathematical Solution

Once the skill of identifying invariants has been mastered, this is an easy problem to solve. For this reason, we have used English to describe the solution, rather than formulate the solution in a mathematical notation. For more complex problems, mathematical notation helps considerably, because it is more succinct and more precise. Let us use this problem to illustrate what we mean.

Abstraction The mathematical solution begins by introducing two *variables*. We let variable p count the number of pieces, and we let variable c count the number of cuts. The values of these variables describe the *state* of the chocolate bar.

This first step is called *abstraction*. We "abstract" from the problem a collection of variables (or "parameters") that completely characterise the essential elements of the problem. In this step, inessential details are eliminated.

One of the inessential details is that the problem has anything to do with chocolate bars! This is totally irrelevant and, accordingly, has been eliminated. The problem could equally well have been about cutting postage stamps from a sheet of stamps. The problem has become a "mathematical" problem, because it is about properties of numbers, rather than a "real-world" problem. Real-world problems are very hard, if not impossible, to solve; in contrast, problems that succumb to mathematical analysis are relatively easy.

Other inessential details that have been eliminated are the sequence of cuts that have been made, and the shapes and sizes of the resulting pieces. That is, the variables p and c do not completely characterise the state of the chocolate bar, or the sequence of cuts that have been made to reach that state. Knowing that, say, four cuts have been made, making five pieces, does not allow us to reconstruct the sizes of the individual pieces. That is irrelevant to solving the problem.

The abstraction step is often the hardest step to make. It is very easy to fall into the trap of including unnecessary detail, making the problem and its solution over-complicated. Conversely, deciding what is essential is far from easy —there is no algorithm for doing this!— . The best problem-solvers are probably the ones most skilled in abstraction.

(Texts on problem-solving often advise drawing a figure. This may help to clarify the problem statement —for example, we included fig. 2.1 in order to clarify what is meant by a cut— but it can also be a handicap! There are two reasons. The first is that extreme cases are often difficult to capture in a figure. This is something we return to later. The second is that figures often contain much unnecessary detail, as exemplified by fig. 2.1. Our advice is to use figures with the utmost caution; mathematical formulae are most often far more effective.)

Assignments The next step in the problem's solution is to model the process of cutting the chocolate bar. We do so by means of the *assignment statement*

$$p,c \ := \ p+1,c+1 \ .$$

An assignment statement has two sides, a *left* side and a *right* side. The two sides are separated by the *assignment symbol* " $:=$ ", pronounced "becomes". The left side is a comma-separated list of variables (in this case, p,c). No variable may occur more than once in the left side. The right side is a comma-separated list of expressions (in this case, $p+1,c+1$). The list must have length equal to the number of variables on the left side.

An assignment effects a change of state. An assignment is executed by evaluating, in the current state, each expression on the right side. The state is then changed by replacing the value of each variable on the left side by the value of the corresponding expression on the right side. In our example, the state —the number of pieces and the number of cuts— is changed by evaluating $p+1$ and $c+1$, and then replacing the values of p and c by these values, respectively. In words, p "becomes" $p+1$, and c "becomes" $c+1$. This is how the assignment statement models the process of making a single cut of the chocolate bar[1].

An *invariant* of an assignment is some function of the state whose value remains constant under execution of the assignment. For example, $p-c$ is an invariant of the assignment $p,c \ := \ p+1,c+1$.

Suppose E is an expression depending on the values of the state variables. (For example, $p-c$ is an expression depending on variables p and c .) We can check that E is an invariant simply by checking for equality between the value of E , and the value of E after replacing all variables as prescribed by the assignment. For example, the equality

[1] ***A word of warning*** (for those who have already learnt to program in a language like Java or C): The assignment statements we will be using are often called *simultaneous assignments* because several variables are allowed on the left side, their values being updated *simultaneously* once the right side has been evaluated. Some programming languages do not allow simultaneous assignments, restricting the programmer to a *single* variable on the left side in all assignments. Java is an example. Instead of a simultaneous assignment, one has to write a sequence of assignments. This is a nuisance, but only that. Much worse is that the equality symbol, " $=$ ", is used instead of the assignment symbol, Java being again an example. This is a major problem because it causes confusion between assignments and equalities, which are two quite different things. Most novice programmers frequently make the mistake of confusing the two, and even experienced programmers sometimes do, leading to difficult-to-find errors. If you do write Java or C programs, always remember to pronounce an assignment as "left side ***becomes*** right side", and not "left side equals right side", even if your teachers do not do so.

$$p{-}c \;=\; (p{+}1)-(c{+}1) \quad ,$$

holds whatever the values of p and c. This checks that $p{-}c$ is an invariant of the assignment $p\,,c \;:=\; p{+}1\,,c{+}1$. The left side of this equality is the expression E and the right side is the expression E after replacing all variables as prescribed by the assignment.

As another example, suppose we have two variables m and n, and we consider the assignment

$$m\,,n \;:=\; m{+}3\,,n{-}1$$

We check that $m+3{\times}n$ is invariant by checking that

$$m+3{\times}n \;=\; (m{+}3)+3{\times}(n{-}1) \quad .$$

Simple algebra shows that this holds. So, increasing m by 3, simulaneously decreasing n by 1, does not change the value of $m+3{\times}n$.

Given an expression, E, and an assignment, $ls \;:=\; rs$,

$$E[ls := rs]$$

is used to denote the expression obtained by replacing all occurrences of the variables in E listed in ls by the corresponding expression in the list of expressions rs. Here are some examples:

$$(p{-}c)[p\,,c \;:=\; p{+}1\,,c{+}1] \;=\; (p{+}1)-(c{+}1)$$

$$(m+3{\times}n)[m\,,n \;:=\; m{+}3\,,n{-}1] \;=\; (m{+}3)+3{\times}(n{-}1)$$

$$(m{+}n{+}p)[m\,,n\,,p \;:=\; 3{\times}n\,,m{+}3\,,n{-}1] \;=\; (3{\times}n)+(m{+}3)+(n{-}1)$$

The invariant rule for assignments is then the following.

E is an invariant of the assignment $ls \;:=\; rs$ if, for all instances of the variables in E,

$$E[ls := rs] \;=\; E \quad .$$

Induction The final step in the solution of the chocolate problem is to exploit the invariance of $p-c$.

Initially, $p=1$ and $c=0$. So, initially, $p-c=1$. But, $p-c$ is invariant. So, $p-c=1$ no matter how many cuts have been made. When the bar has been cut into all its squares, $p=s$, where s is the number of squares. So, at that time, the number of cuts, c, satisfies $s-c=1$. That is, $c=s-1$. The number of cuts is one less than the number of squares.

An important principle is being used here, called the *principle of mathematical induction*. The principle is very simple. It is that, if the value of an expression is unchanged by some assignment to its variables, the value will be unchanged no matter how many times the assignment is applied. That is, if the assignment is applied zero times, the value of the expression is unchanged (obviously, because applying the assignment zero times means doing nothing). If the assignment is applied exactly once, the value of the expression is unchanged, by assumption. Applying the assignment twice means applying it once and then once again. Both times, the value of the expression remains unchanged, so the end result is also no change. And so on, for three times, four times, etc.

Note that the case of *zero* times is included here. It is very important not to forget zero. In the case of the chocolate-bar problem, it is vital to solving the problem in the case that the chocolate bar has exactly one square (in which case zero cuts are required).

Summary This completes our discussion of the chocolate-bar problem. A number of important problem-solving principles have been introduced — abstraction, invariants and induction. We will see these principles again and again throughout these lectures.

Exercise 2.1 A *knockout tournament* is a series of games. Two players compete in each game; the loser is knocked out (i.e. doesn't play anymore), the winner carries on. The winner of the tournament is the player that is left after all other players have been knocked out.

Suppose there are 1234 players in a tournament. How many games are played before the tournament winner is decided? (Hint: choose suitable variables, and seek an invariant.)

□

2.2 Empty Boxes

Try tackling the empty-box problem. Recall its statement.

> Eleven large empty boxes are placed on a table. An unknown number of the boxes is selected and into each eight medium boxes are placed. An

> unknown number of the medium boxes is selected and into each eight small boxes are placed.
>
> At the end of this process there are 102 empty boxes. How many boxes are there in total?

The following steps should help in determining the solution.

1. Introduce the variables e and f for the number of empty and the number of full boxes, respectively.

2. Identify the initial values of e and f. Identify the final value of e.

3. Model the process of putting eight boxes inside a box as an assignment to e and f.

4. Identify an invariant of the assignment.

5. Combine the previous steps to deduce the final value of f. Hence deduce the final value of $e+f$.

Note that this solution does not try to count the number of medium boxes, or the number of small boxes, or which are full and which are empty. All of these are irrelevant, and a solution that introduces variables representing these quantities is grossly over-complicated.

This is a key to effective problem-solving: keep it simple!

2.3 The Tumbler Problem

Let us now look at how to solve the tumbler problem. Recall the statement of the problem.

> Several tumblers are placed in a line on a table. Some tumblers are upside down, some are the right way up. It is required to turn all the tumblers the right way up. However, the tumblers may not be turned individually; an allowed move is to turn any *two* tumblers simultaneously. From which initial states of the tumblers is it possible to turn all the tumblers the right way up?

The problem suggests that we introduce just one variable that counts the number of tumblers that are upside down. Let us call it u.

There are three possible effects of turning two of the tumblers. Two tumblers that are both the right way up are turned upside down. This is modelled by the assignment

$$u := u+2 \ .$$

Turning two tumblers that are both upside down has the opposite effect — u decreases by two. This is modelled by the assignment

$$u := u-2 \ .$$

Finally, turning two tumblers that are the opposite way up (that is, one upside down, the other the right way up) has no effect on u. In programming terms, this is modelled by a so-called *skip* statement. "Skip" means "do nothing" or "having no effect". In this example, it is equivalent to the assignment

$$u := u \ ,$$

but it is better to have a name for the statement that does not depend on any variables. We use the name skip. So, the third possibility is to execute

$$\mathsf{skip} \ .$$

The choice of which of these three statements is executed is left unspecified. An invariant of the turning process must therefore be an invariant of each of the three.

Everything is an invariant of skip. So, we can discount skip. We therefore seek an invariant of the two assignments $u := u+2$ and $u := u-2$. What does not change if we add or subtract two from u?

The answer is: the so-called *parity* of u. The parity of u is a *boolean* value: it is either true or false. It is true if u is even (zero, two, four, eight etc.) and it is false if u is odd (one, three, five, seven, etc.). Let us write $even.u$ for this Boolean quantity. Then,

$$(even.u)[u := u+2] = even.(u+2) = even.u \ .$$

That is, $even.u$ is an invariant of the assignment $u := u+2$. Also,

$$(even.u)[u := u-2] = even.(u-2) = even.u \ .$$

That is, $even.u$ is also an invariant of the assignment $u := u-2$.

We conclude that, no matter how many times we turn two tumblers over, the parity of the number of upside-down tumblers will not change. If there is an even number at the outset, there will always be an even number; if there is an odd number at the outset, there will always be an odd number.

The goal is to repeat the turning process until there are zero upside-down tumblers. Zero is an even number, so the answer to the question is that there must be an even number of upside-down tumblers at the outset.

You should now be in a position to solve the problem of the black and white balls (problem 4 in the introductory section). Apply the method of introducing appropriate variables to describe the state of the balls in the urn. Then express the process of removing and/or replacing balls by a choice among a number of assignment statements. Identify an invariant, and draw the appropriate conclusion. The chessboard problem is a little harder, but can be solved in the same way. (Hint: use the colouring of the squares on the chessboard.) Problem 6(a) should be a bit easier. It's a preliminary to solving 6(b), which we do —together with 6(a)— in the next section. Have a peek if you want to.

2.4 Tetrominoes

In this section, we present the solution of problem 6(b). This gives us the opportunity to introduce a style of mathematical calculation that improves clarity.

Recall the problem.

> Suppose a rectangular board can be covered with T-tetrominoes. Show that the number of squares is a multiple of 8.

A brief analysis of this problem reveals an obvious invariant. Suppose c denotes the number of covered squares. Then, placing a tetromino on the board is modelled by

$$c \ := \ c+4 \ .$$

Thus, $c \bmod 4$ is invariant. ($c \bmod 4$ is the remainder after dividing c by 4. For example, $7 \bmod 4$ is 3, and $16 \bmod 4$ is 0.) Initially c is 0, so $c \bmod 4$ is $0 \bmod 4$, which is 0. So, $c \bmod 4$ is always 0. In words, we say "c is a multiple of 4 is an invariant property". More often, the words "is an invariant property" are omitted, and we say "c is a multiple of 4".

Now, suppose the tetrominoes cover an $m{\times}n$ board. (That is, the number of squares along one side is m and the number along the other side is n.) Then, $c = m{\times}n$ and, so, $m{\times}n$ is a multiple of 4. For the product $m{\times}n$ of two numbers m and n to be a multiple of 4, it must be the case that either m or n (or both) is a multiple of 2.

Note that, so far, the argument has been about tetrominoes in general, and not particularly about T-tetrominoes. What we have just shown is, in fact, the solution to problem 6(a): if a rectangular board is covered by tetrominoes, at least one of the sides of the rectangle must have even length.

The discovery of a solution to problem 6(a), in this way, illustrates a general phenomenon in solving problems . The process of solving more difficult problems typically involves formulating and solving simpler subproblems. In fact, one could say that a "difficult" problem is one that involves putting together the solution to several simple problems. Looked at this way, "difficult" problems become a lot more manageable. Just keep on solving simple problems until you have reached your goal!

At this point, we want to introduce a style for presenting calculations that is clearer than the normal mixture of text with interspersed mathematical expresssions. To introduce the style we repeat the argument just given. Here it is in the new style:

$$\begin{array}{ll} & \text{an } m\times n \text{ board is covered with tetrominoes} \\ \Rightarrow & \{ \quad \text{invariant: } c \text{ is a multiple of } 4, \\ & \quad\quad c = m\times n \quad \} \\ & m\times n \text{ is a multiple of } 4 \\ \Rightarrow & \{ \quad \text{property of multiples} \quad \} \\ & m \text{ is a multiple of } 2 \vee n \text{ is a multiple of } 2 \quad . \end{array}$$

This is a two-step calculation. The first step is a so-called "implication" step, as indicated by the "$\Rightarrow$" symbol. The step is read as

If an $m\times n$ board is covered with tetrominoes, $m\times n$ is a multiple of 4.

(Alternatively, "an $m\times n$ board is covered with tetrominoes *implies* $m\times n$ is a multiple of 4" or "an $m\times n$ board is covered with tetrominoes *only if* $m\times n$ is a multiple of 4.")

The text between curly brackets, following the "$\Rightarrow$" symbol is a hint why the statement is true. Here the hint is the combination of the fact, proved earlier, that the number of covered squares is always a multiple of 4 (whatever the shape of the area covered) together with the fact that, if an $m\times n$ board has been covered, the number of covered squares is $m\times n$.

The second step is read as:

If $m\times n$ is a multiple of 4, m is a multiple of 2 or n is a multiple of 2.

Again, the "$\Rightarrow$" symbol signifies an implication. The symbol "$\vee$" means "or". Note that by "or" we mean so-called "inclusive or" — the possibility that both m and n are multiples of 2 is included. A so-called "exclusive or" would mean that m is a multiple of 2 or n is a multiple of 2, but not both, i.e. it would exclude this possibility.

The hint, in this case, is less specific. The property that is being alluded to has to do with expressing numbers as multiples of prime numbers. You may or may not be familiar with the general theorem, but you should have sufficient knowledge of multiplying numbers by 4 to accept that the step is valid.

The conclusion of the calculation is also an "if" statement. It is:

If an $m{\times}n$ board is covered with tetrominoes, m is a multiple of 2 or n is a multiple of 2.

This style of presenting a mathematical calculation reverses the normal style: mathematical expressions are interspersed with text, rather than the other way around. Including hints within curly brackets between two expressions means that the hints may be as long as we like; they may even include other subcalculations. Including the symbol “$\Rightarrow$” makes clear the relation between the expressions it connects. More importantly, it allows us to use other relations. Later, we present calculations in which “$\Leftarrow$” is the connecting symbol. Such calculations work backwards from a goal to what has been given, which is often the most effective way to reason.

Let us now tackle problem 6(b) head on. Clearly, the solution must take account of the shape of a T-tetronomo. (It isn’t true for I-tetronimoes. A $4{\times}1$ board can be covered with 1 I-tetronimo, and 4 is not a multiple of 8.)

What distinguishes a T-tetronimo is that it has one square that is adjacent to the other three squares. Colouring this one square differently from the other three suggests colouring the squares of the rectangle in the way a chessboard is coloured.

Suppose we indeed colour the rectangle with black and white squares, as on a chessboard. The T-tetrominoes should be coloured in the same way. This gives us two types, one with three black squares and one white square, and one with three white squares and one black square. We call them *dark* and *light* T-tetrominoes. (See fig. 2.5.) Placing the tetrominoes on the board now involves choosing the appropriate type so that the colours of the covered squares match the colours of the tetrominoes.

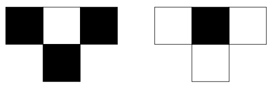


Figure 2.5: Dark and light T-tetrominoes

We introduce four variables to describe the state of the board. The variable b counts the number of covered black squares, whilst w counts the number of covered white squares. In addition, d counts the number of dark T-tetrominoes that have been used, and l counts the number of light tetrominoes.

Placing a dark tetromino on the board is modelled by the assignment

$$d\,,b\,,w\;:=\;d{+}1\,,b{+}3\,,w{+}1\quad .$$

Placing a light tetromino on the board is modelled by the assignment

$$l\,,b\,,w\;:=\;l{+}1\,,b{+}1\,,w{+}3\quad .$$

An invariant of both assignments is

$$b-3\times d-l \quad ,$$

since

$$
\begin{aligned}
& (b-3\times d-l)[d\,,b\,,w \;:=\; d{+}1\,,b{+}3\,,w{+}1] \\
= & \qquad \{ \qquad \text{definition of substitution} \quad \} \\
& (b{+}3)-3\times(d{+}1)-l \\
= & \qquad \{ \qquad \text{arithmetic} \quad \} \\
& b-3\times d-l
\end{aligned}
$$

and

$$
\begin{aligned}
& (b-3\times d-l)[l\,,b\,,w \;:=\; l{+}1\,,b{+}1\,,w{+}3] \\
= & \qquad \{ \qquad \text{definition of substitution} \quad \} \\
& (b{+}1)-3\times d-(l{+}1) \\
= & \qquad \{ \qquad \text{arithmetic} \quad \} \\
& b-3\times d-l \quad .
\end{aligned}
$$

Similarly, another invariant of both assignments is

$$w-3\times l-d \quad .$$

Now, the initial value of $b-3\times d-l$ is zero. So, it will always be zero, no matter how many T-tetrominoes are placed on the board. Similarly, the value of $w-3\times l-d$ will always be zero.

We can now solve the given problem.

$$
\begin{aligned}
& \text{a rectangular board is covered by T-tetrominoes} \\
\Rightarrow & \qquad \{ \qquad \text{from problem 6(a) we know that at least one} \\
& \qquad\qquad \text{side of the board has an even number of squares,} \\
& \qquad\qquad \text{which means that the number of black squares} \\
& \qquad\qquad \text{equals the number of white squares} \quad \} \\
& b=w \\
\Rightarrow & \qquad \{ \qquad b-3\times d-l=0 \\
& \qquad\qquad w-3\times l-d=0 \quad \} \\
& (b=w) \wedge (3\times d+l=3\times l+d)
\end{aligned}
$$

$\Rightarrow$ { arithmetic }

$(b = w) \wedge (l = d)$

$\Rightarrow$ { $b - 3\times d - l = 0$

$w - 3\times l - d = 0$ }

$b = w = 4\times d = 4\times l$

$\Rightarrow$ { arithmetic }

$b + w = 8\times d$

$\Rightarrow$ { $b + w$ is the number of covered squares }

the number of covered squares is a multiple of 8 .

We conclude that

> If a rectangular board is covered by T-tetrominoes, the number of covered squares is divisible by 8 .

You can now tackle 6(c). The problem looks very much like 6(b), which suggests that it can be solved in a similar way. Indeed, it can. Look at other ways of colouring the squares black and white. Having found a suitable way, you should be able to repeat the same argument as above. Be careful to check that all steps remain valid.

(How easily you can adapt the solution to one problem in order to solve another is a good measure of the effectiveness of your solution method. It shouldn't be too difficult to solve 6(c) because the solution to 6(b), above, takes care to clearly identify those steps where a property or properties of T-tetrominoes are used. Similarly, the solution also clearly identifies where the fact that the area covered is rectangular is exploited. Badly presented calculations do not make clear which properties are being used. As a result, they are difficult to adapt to new circumstances.)

Problem 6(d) is relatively easy, once 6(c) has been solved. Good luck!

2.5 Additional Exercises

Exercise 2.2 Given is a bag of three kinds of objects. The total number of objects is reduced by repeatedly removing two objects of different kind, and replacing them by an object of the third kind.

Identify exact conditions in which it is possible to remove all the objects except one.

□

2.6 Bibliographic Remarks

The empty-box problem was given to me by Wim Feijen. The problems of the black and white balls is from [Gri81]. The tetromino problems I found in the 1999 *Vierkant Voor Wiskunde* calendar. (See http://www.vierkantvoorwiskunde.nl/puzzels/.) Vierkant Voor Wiskunde —foursquare for mathematics— is a foundation that promotes mathematics in Dutch schools. Their publications contain many examples of mathematical puzzles, both new and old. I have made grateful use of them throughout this text. Thanks go to Jeremy Weissman for suggestions on how to improve the presentation of the tetronimo problems, some of which I have made use of. The domino and tumbler problems are old chestnuts. I do not know their origin.

Exercise 2.2 was posed to me by Dmitri Chubarov. It was posed (in a slightly different form) in the Russian national Mathematics Olympiad in 1975 and appears in a book by Vasiliev entitled "Zadachi Vsesoyuzynykh Matemticheskikh Olympiad" published in Moscow, 1988. The author of the problem is apparently not stated.

Chapter 3

Crossing a River

The examples in this chapter all involve getting a number of people or things across a river under certain constraints. We use them as simple illustrations of "brute-force" search and problem decomposition.

Brute-force search means systematically trying all possibilities. It's a technique that doesn't require any skill, but does require a lot of careful and accurate work. Using brute force is not something human beings are good at; lots of careful, accurate work is something more suited to computers. But, brute force isn't even practical for implementation on a computer. The amount of work involved explodes as the problem size gets bigger, making it impractical for all but toy problems. Nevertheless, it is useful to know what brute force entails, because it helps to understand the nature of problem-solving.

Problem decomposition is something we humans are much better at. Problem decomposition involves exploiting the structure of a problem to break it down into smaller, more manageable problems. Once a problem has been broken down in this way, brute force can be applied. Indeed, it is often the case that, ultimately, brute force is the only solution method, so we can't dispense with it. However, it is much better to spend more effort in decomposing a problem, postponing the use of a brute-force search for as long as possible.

All river-crossing problems have an obvious structural property, namely the symmetry between the two banks of the river. The exploitation of symmetry is a very important problem-solving technique, but is often overlooked, particularly when using brute force. You may already have seen the problems, or similar ones, elsewhere. As illustrations of brute-force search —which is how their solutions are often presented— they are extremely uninteresting! However, as illustrations of the use of symmetry, combined with problem decomposition, they have startling, hidden beauty.

An important issue that emerges in this chapter is *naming* the elements of a problem. Deciding on what and how names should be introduced can be crucial to success. We shall see how inappropriate or unnecessary naming can increase the complexity of a problem, making it impossible to solve even with the aid of a very powerful computer.

3.1 Problems

1. **Goat, Cabbage and Wolf.**

   A farmer wishes to ferry a goat, a cabbage and a wolf across a river. However, his boat is only large enough to take one of them at a time, making several trips across the river necessary. Also, the goat should not be left alone with the cabbage (otherwise, the goat would eat the cabbage), and the wolf should not be left alone with the goat (otherwise, the wolf would eat the goat).

   How can the farmer achieve the task?

2. **The Jealous Couples.**

   Three couples (husband and wife) wish to cross a river. They have one boat that can carry at most two people, making several trips across the river necessary. The husbands are so jealous of each other that none is willing to allow their wife to be with another man, if they are not themselves present.

   How can all three couples get across the river?

3. **Adults and Children.**

   A group of adults and children are on one side of a river. They have one boat that is only big enough to accommodate one adult or two children.

   How can all the adults and all the children cross the river? Make clear any assumptions you are obliged to make.

4. **Overweight**

   Ann, Bob, Col and Dee are on one side of a river. They have one rowing boat that can carry at most 100 kilos. Ann is 46 kilos, Bob is 49 kilos, Col is 52 kilos and Dee is 100 kilos. Bob can't row.

   How can they all get to the other side?

3.2 Brute Force

3.2.1 Goat, Cabbage and Wolf

The goat-, cabbage- and wolf-problem is often used to illustrate brute-force search. Our main purpose in showing the brute-force solution is to illustrate the pitfalls of *poor* problem-solving skills. Additionally, we introduce some terminology that is useful when discussing the efficiency of a particular solution to a problem.

> A farmer wishes to ferry a goat, a cabbage and a wolf across a river. However, his boat is only large enough to take one of them at a time, making several trips across the river necessary. Also, the goat should not be left alone with the cabbage (otherwise, the goat would eat the cabbage), and the wolf should not be left alone with the goat (otherwise, the wolf would eat the goat).
>
> How can the farmer achieve the task?

The problem involves four individuals, and each is at one of the two river banks. This means that we can represent a state by four variables, each of which has one of two values. We call the variables f (for farmer), g (for goat), c (for cabbage) and w (for wolf), and we call their possible values L (for left) and R (for right). A value of R means "at the right bank". A value of L means at the left bank. Note that the boat is always where the farmer is, so we do not need to introduce a variable to represent its position.

A *brute-force search* involves constructing a *state-transition graph* that models all possible *states*, and ways of changing from one state to another — the *state transitions.* In the goat-, cabbage-, wolf-problem, a state describes on which bank each of the four individuals can be found. A state transition is a change of state that is allowed by the problem specification. For example, two states between which there is a valid state transition are:

1. All four are at the left bank.
2. The farmer and goat are at the right bank, whilst the cabbage and wolf are at the left bank.

For the very simplest problems, a diagram can be drawn depicting a state-transition graph. The states are drawn as circles, and the state transitions are drawn as lines connecting the circles. The lines have arrows on them if some state transitions are not reversible; if so, the diagram is called a *directed graph.* If all state transitions are reversible, the arrows are not necessary and the diagram is called an *undirected graph.* We are going to draw a state-transition graph to demonstrate the brute-force solution to this problem.

If four variables can each have one of two values, there are 2^4 (i.e. sixteen) different combinations of values. However, in this problem some of these combinations are excluded. The requirement that the goat cannot be left alone with the cabbage is expressed by the *system invariant*

$$f = g = c \ \vee \ g \neq c \ \ .$$

That is, either the farmer, the goat and the cabbage are all on the same bank ($f = g = c$), or the goat and cabbage are on different banks ($g \neq c$). This excludes cases where g and c are equal, but different from f. Similarly, the requirement that the goat cannot be left alone with the wolf is expressed by the system invariant

$$f = g = w \ \vee \ g \neq w \ \ .$$

If we list all states, eliminating the ones that are not allowed, the total reduces to ten. The table below shows the ten different combinations. (Notice that when f and g are equal all combinations of c and w are allowed; when f and g are different, c and w are required to be equal.)

f	g	c	w
L	L	L	L
L	L	L	R
L	L	R	L
L	L	R	R
L	R	L	L
R	L	R	R
R	R	L	L
R	R	L	R
R	R	R	L
R	R	R	R

Now, we enumerate all the possible transitions between these states. The graph in fig. 3.1 does just this. The *nodes* of the graph —the boxes— represent states, and the *edges* of the graph —the lines connecting the boxes— represent transitions. There are no arrows on the edges because each transition can be reversed.

At the very left, the box labelled "LLLL" represents the state where all four are on the left bank. The only allowed transition from this state is to the state where the farmer and goat are at the right bank, and the cabbage and wolf are at the left bank. This is represented by the line connecting the "LLLL" box to the "RRLL" box.

From the graph, it is clear that there are two solutions to the problem. Each solution is given by a *path* through the graph from "LLLL" box to the "RRRR" box. The upper path gives the following solution:

1. The farmer takes the goat to the right bank, and returns alone. This is the path from LLLL to LRLL.

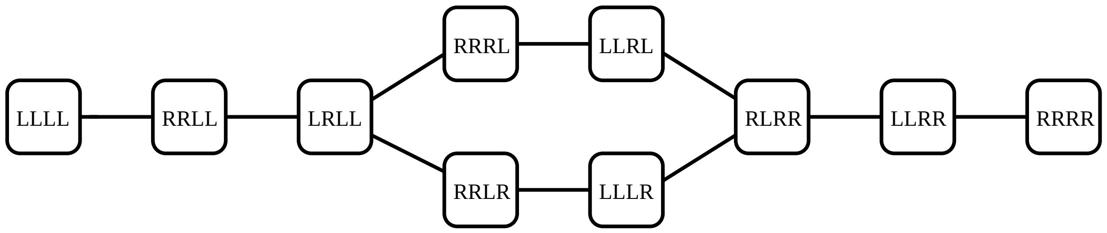


Figure 3.1: Goat-, Cabbage-, Wolf-Problem

2. The farmer takes the cabbage to the right bank, and returns with the goat. This is the path from LRLL to LLRL.

3. The farmer takes the wolf to the right bank, and returns alone. This is the path from LLRL to LLRR.

4. The farmer takes the goat to the right bank. This is the path from LLRR to RRRR.

The alternative solution, given by the lower path, interchanges "cabbage" and"wolf" in the second and third steps.

3.2.2 State-Space Explosion

There is often a tendency to apply brute force without thinking when faced with a new problem. However, it should only be used where it is unavoidable. Brute force is only useful for very simple problems. For other problems, the search space quickly becomes much too large. In the jargon used by computing scientists, brute force does not "scale up" to larger problems. The goat-, cabbage- and wolf-problem is not representative; the above —thoughtless!— solution has a manageable number of states, and a manageable number of transitions.

We can see how quickly the search space can grow by analysing what is involved in using brute force to solve the remaining problems in section 3.1.

In the "overweight" problem, there are four named individuals and no restrictions on their being together on the same side of the bank. So, there are 16 possible states; unlike in the goat-, cabbage- and wolf-problem, no restriction on the size of the state space is possible. Also, from the initial state there are four different transitions; from most other states, there are at least two transitions. So, the total number of transitions is large, too large even for the most diligent problem-solvers.

The situation in an unskilled solution of the "jealous-couples" problem is even worse. Here, there are six individuals involved, each of whom can be on one side or other of the river bank. If we give each individual a distinct name, the number of states is 2^6, i.e. 64! That's an impossible number for any human being to cope with, and we haven't even begun to count the number of transitions. In another variation on the jealous-couples problem, there are five couples, and the boat can take three people at a time. That means, if all are named, there are are 2^{10}, i.e. 1024, different states, and a yet larger number of transitions. Take note: these are "toy" problems, not real problems.

The "adults-and-children" problem illustrates another failing of brute force, namely that it can only be applied in specific cases, and not in the general case. The number of adults and children is not specified in this problem. Yet, it is in fact the easiest of all to solve.

The use of a computer to perform a brute-force search shifts the meaning of what is a "small" problem and what is a "large" problem, but not as much as one might expect. The so-called "state-space explosion problem" gets in the way. The river-crossing problems illustrate "state-space explosion" very well. If there are n individuals in such a problem, there are, in principle, 2^n different states to be considered. But, even for quite small n, 2^n is a very large number. We speak of an "exponential" growth in the number of states (n is the exponent in 2^n). Whenever the state space of a class of problems grows exponentially, it means that even the largest and fastest supercomputers can only tackle quite small instances.

Drawing state-transition diagrams is equally ineffective. A diagram can occasionally be used to illustrate the solution of a simple, well-chosen problem. But constructing a diagram is rarely helpful in problem-solving. Instead, diagrams quickly become a problem in themselves — apart from the size of paper needed, how are the nodes to be placed on the paper so that the diagram becomes readable?

3.2.3 Abstraction

The state-space explosion is often caused by a failure to properly analyse a problem; a particularly frequent cause is unnecessary or inappropriate *naming*. The goat-cabbage-and-wolf problem is a good example.

In the goat-cabbage-and-wolf problem, distinct names are given to the "farmer", the "goat", the "cabbage" and the "wolf". But, do we really need to distinguish between all four? In the discussion of the state space, we remarked on a "similarity" between the wolf and the cabbage. Specifically, the goat cannot be left with either the wolf or the cabbage. This "similarity" also emerged in the solution: two solutions were obtained, symmetrical in the interchange of "wolf" and "cabbage". Why, then, are the "wolf" and the "cabbage" distinguished by giving them different names?

Let us restate the problem, this time with a naming convention that omits the unnecessary distinction between the wolf and the cabbage. In the restated problem, we call the goat an “alpha” and the cabbage and the wolf “betas”.

> A farmer wishes to ferry an alpha and two betas across a river. However, his boat is only large enough to take one of them at a time, making several trips across the river necessary. Also, an alpha should not be left alone with a beta.
>
> How can the farmer achieve the task?

Now the problem becomes much easier to solve. Indeed, there is only one solution: Take the alpha across, and then one beta across, returning with the alpha. Then take the second beta across, followed by the alpha. Because there is only one solution, it is easy to discover, and it is unnecessary to construct a state-transition diagram for the problem.

The problem-solving principle that we learn from this example is very important.

> ***Avoid unnecessary or inappropriate naming.***

When elements of a problem are given individual names, it distinguishes them from other elements of the problem, and adds to the size of the state space. The process of omitting unnecessary detail, and reducing a problem to its essentials is called *abstraction*. Poor solutions to problems are ones that fail to “abstract” adequately, making the problem more complicated than it really is. We encounter the importance of appropriate naming time and again in the coming chapters. Bear it in mind as you read.

3.3 Jealous Couples

Very often, a problem has an inherent structure that facilitates decomposing the problem into smaller problems. The smaller problems can then be further decomposed until they become sufficiently manageable to be solvable by other means, perhaps even by brute force. Their solutions are then put together to form a solution to the original problem.

The jealous-couples problem is an excellent example. It can be solved by brute force, making it decidedly boring. But, it can be solved much more effectively, making use of general problem-solving principles.

Recall its statement:

> Three couples (husband and wife) wish to cross a river. They have one boat that can carry at most two people, making several trips across the river necessary. The husbands are so jealous of each other that none is willing to allow their wife to be with another man if they are not themselves present.
>
> How can all three couples get across the river?

3.3.1 What's The Problem?

Before we tackle this particular problem, let us try to determine what the essence of the problem is.

Suppose there is one boat that can carry two "things", and there are no other restrictions. Then, clearly, it is possible to get any number of "things" across the river: repeat the process of letting two cross from left to right, followed by one returning from right to left, until at most two remain on the left bank.

Now, by replacing "thing" by "couple", we infer that a boat that can carry two couples at one crossing can be used to ferry an arbitrary number of couples across the river. (After all, couples are not jealous of each other!) Since a couple is two people, this means that a boat that can carry four people is sufficient to ferry an arbitrary number of couples across the river.

This simple analysis gives us a different slant on the problem. Rather than tackle the problem as stated, we can tackle a related problem, namely, what is the minimum capacity needed to ferry three couples across the river? More generally, what is the minimum capacity needed to ferry n couples across the river? Obviously, the minimum capacity is at least two (since it is not possible to ferry more than one person across a river in a boat that can only carry one person at a time), and we have just shown that it is at most four.

Alternatively, we can specify the capacity of the boat and ask what is the maximum number of couples that can be ferried across with that capacity. If the capacity is one (or less) the maximum number of couples is zero, and if the capacity is four, there is no maximum. So, the question is how many couples can be ferried with a boat of capacity two, and how many couples can be ferried with a boat of capacity three.

The new problems look more difficult than the original. In the original problem, we are given the answer —in the case of three couples, a boat with capacity two is needed— and we are required to give a constructive proof that this is the case. But, there is often an advantage in not knowing the answer — because we can sometimes gain insight by generalising, and then first solving simpler instances of the general problem.

3.3.2 Problem Structure

The structure of this problem suggests several ways in which it might be decomposed. First, there are three couples. This suggests seeking a solution that gets each couple across in turn. That is, we decompose the problem into three subproblems: get the first couple across, get the second couple across, and get the third couple across.

Another decomposition is into husbands and wives. According to the maxim "ladies before gentlemen", we could try first getting all the wives across, followed by all the husbands. Alternatively, letting “age go before beauty”, we could try first getting all the husbands across, followed by all the wives.

Getting all the wives across, whilst their husbands remain at the left bank turns out to be easy. The reason is that, if the husbands all stay in one place, there is no difficulty in transferring the wives *away* from them. Getting all the husbands across first, whilst their wives stay at the left bank, seems much harder. On the other hand, getting the husbands to join their wives may prove to be harder than getting the wives to join their husbands. Ladies before gentleman, or age before beauty; there doesn't seem much to choose between them.

There is, however, one key structural property of the problem that we have not yet considered. It is the *symmetry* between the left and right banks. The process of getting a group of people from left to right can always be reversed; the result is a process for getting the same group of people from right to left. Perhaps a symmetric solution is possible! If that is the case, we only need to do half the work, and that is a major saving. This is indeed what we do.

(The state-transition diagram for the goat-, cabbage-, wolf-problem exhibits the left-right symmetry very well. The diagram also illustrates the symmetry between the cabbage and wolf. Both symmetries were to be expected from the problem statement; by ignoring them and using brute-force, we lost the opportunity of a reduction in effort.)

3.3.3 Denoting States and Transitions

We begin by introducing some local notation to make the solution strategy precise. The introduction of notation involves *naming* the elements of the problem that we want to distinguish. As discussed earlier, this is a crucial step in finding a solution.

Here, we use letters H, W and C to mean husband, wife and couple, respectively. These are preceded by a number; for example, $2\mathsf{H}$ means two husbands, $3\mathsf{C}$ means three couples and $1\mathsf{C}{,}2\mathsf{H}$ means one couple and two husbands. We exploit the notation to distinguish between couples and individuals; for example, $1\mathsf{H}{,}1W$ means a husband and wife who do not form a couple, whilst $1\mathsf{C}$ means a husband and wife who do form a couple.

Note that we do *not* name the individual people as in, for example, Ann, Bob, Clare etc. It is only the *number* of husbands, wives and couples that is relevant to the problem's solution. Number is an extremely important mathematical abstraction.

We distinguish between *states* and *actions*.

A *state* describes a situation when each individual (husband or wife) is at one of the banks. A state is denoted by two sequences separated by bars. An example is $3\mathsf{H} \,||\, 3W$, which denotes the state in which all three husbands are at the left bank, and all three wives are at the right bank. A second example of a state is $1\mathsf{C}{,}2\mathsf{H} \,||\, 2W$, which denotes the state in which one couple and two husbands are at the left bank and two wives are at the right bank. The starting state is thus $3C \,||$ and the required finishing state is $||\, 3C$.

An *action* is when some individuals are being transported across the river. An example is $3H\ |2W|\ 1W$; this denotes the action of transporting two wives across the river, leaving three husbands at the left bank and one wife at the right bank.

Note that the notation for states and actions does not specify the position or direction of the boat, and, taken out of context, could be ambiguous. Since the position of the boat must alternate between the left bank and the right bank, this ambiguity is easily resolved.

The notation allows valid and invalid states/actions to be easily identified. For example, $1C{,}1W \,||\, 1C{,}1H$ is invalid (because there is a wife who is on the same side of the river as a man other than her husband, who is on the other side of the river). Also, $3H\ |3W|$ is invalid because the boat can only carry at most two people.

In general, a complete, detailed solution to the problem is a sequence, beginning with the state $3C \,||$ and ending with the state $||\, 3C$, that alternates between states and actions.

An action results in a change of state. (In the terminology of state-transition diagrams, an action effects a transition between states.) Additional notation helps to express the result of actions. If p and q denote states, and S denotes a sequence of actions,

$$\begin{array}{l} \{\ \ p\ \ \} \\ S \\ \{\ \ q\ \ \} \end{array}$$

is the property that, if the sequence of actions S is performed beginning in state p , it will result in state q . So, for example,

$$\begin{array}{l} \{\ \ 2C{,}1H \,||\, 1W\ \ \} \\ 3H\ |2W|\ 1W \\ \{\ \ 3H \,||\, 3W\ \ \} \end{array}$$

is the property that, beginning in the state where two couples and one husband are at the left bank, letting two wives cross will result in a state in which all three husbands are at the left bank, whilst all three wives are at the right bank.

Of course, we should always check the validity of such properties. It is easy to make a mistake and make an invalid claim. Care is needed, but the checks are straightforward.

3.3.4 Problem Decomposition

Using this notation we can express our strategy for decomposing the problem. The goal is to construct a sequence of actions S_0 satisfying

$$\{\ 3C \parallel\ \}\quad S_0\quad \{\ \parallel 3C\ \}\ .$$

Our strategy can be summarised as exploiting two properties of the problem.

- The left-right symmetry.
- The fact that it is easy to get the wives from one side to the other whilst their husbands remain on one bank.

This strategy is realised by decomposing S_0 into three sequences S_1, S_2 and S_3 such that

$$\{\ 3C \parallel\ \}\quad S_1\quad \{\ 3H \parallel 3W\ \}\ ,$$

$$\{\ 3H \parallel 3W\ \}\quad S_2\quad \{\ 3W \parallel 3H\ \}\ ,$$

$$\{\ 3W \parallel 3H\ \}\quad S_3\quad \{\ \parallel 3C\ \}\ .$$

The sequence S_1 changes the state from the start state to the state where all the wives are at the right bank and all the husbands are at the left bank. The sequence S_2 changes the end state of S_1 to the state where the positions of the wives and husbands are reversed. Finally, the sequence S_3 changes the end state of S_2 to the state where everyone is at the right bank. So, doing S_1 followed by S_2 followed by S_3, which we denote by $S_1\ ;\ S_2\ ;\ S_3$, will achieve the objective of changing the state from the start state (everyone is at the left bank) to the final state (everyone is at the right bank).

The decomposition is into *three* components because we want to exploit symmetry, but, clearly, an *odd* number of crossings will be necessary. Symmetry is captured by making the function of S_3 entirely symmetrical to the function of S_1. If we consider the reverse of S_3, its task is to transfer all the wives from the right bank to the left bank. So, if we construct S_1, it is a simple task to construct S_3 directly from it.

We now have to tackle the problem of constructing S_1 and S_2.

As mentioned earlier, getting all the wives across the river, leaving their husbands at the left bank is easy. (It is a problem that can be solved by brute force, if necessary.) Here is how it is achieved.

$$\begin{array}{ll}
 & \{\ 3C \parallel\ \} \\
 & 1C{,}2H\ |2W| \\
; & \{\ 1C{,}2H \parallel 2W\ \} \\
 & 1C{,}2H\ |1W|\ 1W \\
; & \{\ 2C{,}1H \parallel 1W\ \} \\
 & 3H\ |2W|\ 1W \\
 & \{\ 3H \parallel 3W\ \}\ .
\end{array}$$

That is,

$$\{\ 3C \parallel\ \}\quad 1C{,}2H\ |2W|\ ;\ 1C{,}2H\ |1W|\ 1W\ ;\ 3H\ |2W|\ 1W\quad \{\ 3H \parallel 3W\ \}\ .$$

As discussed above, the sequence S_3 is the reverse of S_1:

$$\begin{array}{ll}
 & \{\ 3W \parallel 3H\ \} \\
 & 1W\ |2W|\ 3H \\
; & \{\ 1W \parallel 2C{,}1H\ \} \\
 & 1W\ |1W|\ 1C{,}2H \\
; & \{\ 2W \parallel 1C{,}2H\ \} \\
 & |2W|\ 1C{,}2H \\
 & \{\ \parallel 3C\ \}\ .
\end{array}$$

We are now faced with the harder task of constructing S_2. We seek a solution that is symmetrical about the middle.

Note that, for S_2, the starting position of the boat is the right bank, and its finishing position is the left bank. This is a requirement for S_2 to follow S_1 and be followed by S_3. The length of S_2 must also be odd.

Again, we look for a decomposition into three subsequences. If the solution is to remain symmetric, it must surely take the following form:

$$\begin{array}{ll}
 & \{\ 3H \parallel 3W\ \} \\
 & T_1 \\
; & 1C\ |1C|\ 1C \\
; & T_2 \\
 & \{\ 3W \parallel 3H\ \}\ .
\end{array}$$

Note particularly the middle action — $1C\ |1C|\ 1C$ — . This may be a left-to-right crossing, or a right-to-left crossing; which is not immediately clear. The task is now to construct the symmetric sequences of actions T_1 and T_2.

If the middle action is from right to left, the action must be preceded by the state $1C \parallel 2C$ and results in the state $2C \parallel 1C$. Vice-versa, if the middle action is from left to right, the action must be preceded by the state $2C \parallel 1C$ and results in the state $1C \parallel 2C$. There is little alternative but to use brute-force search to try to determine which can be achieved.

Fortunately, T_1 is soon discovered. It consists of just two actions:

$$\begin{array}{ll}
 & \{\ \ 3H \parallel 3W\ \ \} \\
 & 3H\ |1W|\ 2W \\
; & \{\ \ 1C{,}2H \parallel 2W\ \ \} \\
 & 1C\ |2H|\ 2W \\
 & \{\ \ 1C \parallel 2C\ \ \}\ \ .
\end{array}$$

Symmetrically, for T_2 we have:

$$\begin{array}{ll}
 & \{\ \ 2C \parallel 1C\ \ \} \\
 & 2W\ |2H|\ 1C \\
; & \{\ \ 2W \parallel 1C{,}2H\ \ \} \\
 & 2W\ |1W|\ 3H \\
 & \{\ \ 3W \parallel 3H\ \ \}\ \ .
\end{array}$$

Finally, putting everything together, we have the complete solution to the jealous-couples problem:

$$\begin{array}{ll}
 & \{\ \ 3C \parallel\ \ \} \\
 & 1C{,}2H\ |2W|\ \ ;\ \ 1C{,}2H\ |1W|\ 1W\ \ ;\ \ 3H\ |2W|\ 1W \\
; & \{\ \ 3H \parallel 3W\ \ \} \\
 & 3H\ |1W|\ 2W\ \ ;\ \ 1C\ |2H|\ 2W \\
; & \{\ \ 1C \parallel 2C\ \ \} \\
 & 1C\ |1C|\ 1C \\
; & \{\ \ 2C \parallel 1C\ \ \} \\
 & 2W\ |2H|\ 1C\ \ ;\ \ 2W\ |1W|\ 3H
\end{array}$$

; { 3W || 3H }

1W |2W| 3H ; 1W |1W| 1C,2H ; |2W| 1C,2H

{ || 3C } .

(In this solution, not all intermediate states are shown. This helps to document the solution, by recording the main steps, but not every step. Too much detail in program documentation can be a hindrance.)

3.3.5 A Review

Pause awhile to review the method used to solve the jealous-couples problem, so that you can fully appreciate how much more effective it is than brute-force search.

The construction seeks at each stage to exploit the symmetry between the left and right banks. Since the number of crossings will inevitably be odd, each decomposition is into *three* subsequences, where the first and last are "mirror images" in some sense. Naming the unknown sequences, and formally specifying their function using the { p } S { q } notation helps to clarify what has to be achieved, and to avoid error.

The final solution involves eleven crossings. That's too many for anyone to commit to memory. But, because the solution *method* is well structured, it is easy to remember, making a reconstruction of the solution very simple. Moreover, the solution to the problem *cannot* be used in other contexts, but the solution method *can*. For the proof of the pudding, solve the following related problem:

Exercise 3.1 (Five-couple Problem) There are five jealous couples, and their boat can carry a maximum of three individuals. Determine how to transport all the couples across the river.

□

Exercise 3.2 (Four-couple Problem) Unfortunately, the symmetry between the left and right banks does not guarantee that every river-crossing problem has a symmetric solution. The case that there are four jealous couples, and their boat can carry a maximum of three, has a solution, but it is not symmetric. Determine a solution to this problem.

The following hint may be helpful. Four is less than five, and, by now, you will have solved the problem of transporting five couples across the river (exercise 3.1). So, try to modify the solution for five couples to obtain a solution for four. You should be able to find two solutions in this way, one being obtained from the other by reversing left and right.

(In general, individual solutions need not be symmetric, but the set of solutions is symmetric. That is, there is a transformation from solutions to solutions based on reversing left and right. A solution is symmetric if this transformation maps the solution to itself.)

□

Exercise 3.3 Show that, if the boat can hold a maximum of two people, it is impossible to transport four or more couples across the river.

Show that, if the boat can hold a maximum of three people, it is impossible to transport six or more couples across the river.

Hint: Both problems can be handled together. The crucial properties are:

- At most half of the husbands can cross together.
- The boat can only hold one couple.

□

3.4 Rule of Sequential Composition

The $\{ p \} S \{ q \}$ notation we used for solving the jealous-couples problem is the notation used for specifying and constructing computer programs. It is called a *Hoare triple*. (Sir Tony Hoare is a British computing scientist who pioneered techniques for formally verifying the correctness of computer programs; he was one of the first to use the notation.)

A computer program is specified by a relation between the input values and the output values. The allowed input values are specified by a so-called *precondition*, p, and the output values are specified by a *postcondition*, q. Preconditions and postconditions are properties of the program variables.

If S is a program, and p and q are properties of the program variables,

$$\{ p \} S \{ q \}$$

means that, if the program variables satisfy property p before execution of statement S, execution of S is guaranteed to terminate and, afterwards, the program variables will satisfy property q. For example, a program to compute the remainder r and dividend d after dividing number M by number N would have precondition

$$N \neq 0 \quad ,$$

(since dividing by 0 is not allowed) and postcondition

$$M = N{\times}d + r \ \wedge \ 0 \le r < N \ .$$

If the program is S, the specification of the program is thus

$$\{\, N \neq 0 \,\} \ \ S \ \{\, M = N{\times}d + r \ \wedge \ 0 \le r < N \,\} \ .$$

Programs are often composed by sequencing; the individual components are executed one after the other. A semicolon is usually used to denote sequencing. Thus, if S_1, S_2 and S_3 are programs, $S_1 ; S_2 ; S_3$ denotes the program that is executed by first executing S_1, then executing S_2, and then executing S_3. This is called the *sequential composition* of S_1, S_2 and S_3.

A sequential composition is introduced into a program when the problem it solves is decomposed into subproblems. In the case of a decomposition into two components, given a precondition p and a postcondition q, an intermediate condition, r say, is invented. The problem of constructing a program S satisfying the specification

$$\{\, p \,\}\ S \ \{\, q \,\}$$

is then resolved by letting S be $S_1 ; S_2$ and constructing S_1 and S_2 to satisfy the specifications

$$\{\, p \,\}\ S_1 \ \{\, r \,\}$$

and

$$\{\, r \,\}\ S_2 \ \{\, q \,\} \ .$$

The intermediate condition r thus acts as postcondition for S_1 and precondition for S_2.

If the problem is decomposed into three subproblems, two intermediate conditions are needed. This is what we did in solving the jealous-couples problem. The initial problem statement has precondition $3C \,||$ and postcondition $||\, 3C$. The intermediate conditions $3H \,||\, 3W$ and $3W \,||\, 3H$ were then introduced in order to make the first decomposition.

There are different ways of using the rule of sequential composition. The structure of the given problem may suggest an appropriate intermediate condition. Alternatively, the problem may suggest an appropriate initial computation S_1; the task is then to identify the intermediate condition and the final computation S_2. Conversely, the problem may suggest an appropriate final computation S_2; then the task becomes one of identifying the intermediate condition r and the initial computation S_1.

A concrete illustration is the bridge problem posed in chapter 1. Recall its statement:

> Four people wish to cross a bridge. It is dark, and it is necessary to use a torch when crossing the bridge, but they only have one torch between them. The bridge is narrow and only two people can be on it at any one time. The four people take different amounts of time to cross the bridge; when two cross together they must proceed at the speed of the slowest. The first person takes 1 minute to cross, the second 2 minutes, the third 5 minutes and the fourth 10 minutes. The torch must be ferried back and forth across the bridge, so that it is always carried when the bridge is crossed.
>
> Show that all four can cross the bridge within 17 minutes.

It is easy to see that five trips are needed to get all four people across the bridge. The five trips comprise three trips where two people cross together interleaved with two trips where one person returns with the torch. Since there are only two return trips, and there are four people, at least two people never make a return trip. Clearly, to achieve the shortest time, the two slowest should not make a return journey. The question is whether they should cross together or seperately.

The times have been chosen in this example so that the shortest time is achieved when the two slowest cross together. (It isn't always the case that the best strategy is to let the two slowest cross together. See exercise 3.4.) Using the times as identifiers for the individuals, the solution will therefore include, for some p and q, a crossing of the form

$$p\ |5{,}10|\ q$$

as one step. Seeking to exploit symmetry, the task becomes one of determining sequences of crossings S_1 and S_2, and p and q such that

$$\{\ 1{,}2{,}5{,}10\ ||\ \}\ \ S_1\ \ \{\ p{,}5{,}10\ ||\ q\ \}$$

$$\{\ p{,}5{,}10\ ||\ q\ \}\ \ p\ |5{,}10|\ q\ \ \{\ p\ ||\ q{,}5{,}10\ \}$$

and

$$\{\ p\ ||\ q{,}5{,}10\ \}\ \ S_2\ \ \{\ ||\ 1{,}2{,}5{,}10\ \}\ .$$

We leave you to complete the rest.

Exercise 3.4 (The Torch Problem) Consider the torch problem in the case that the crossing times of the four individuals form the input parameters. Suppose the first person takes t_1 minutes to cross, the second t_2 minutes, the third t_3 minutes and the fourth t_4 minutes. Assume that $t_1 \leq t_2 \leq t_3 \leq t_4$. Find a method of getting all four across that mimimises the total crossing time. You may assume that, in an optimal solution, every forward trip involves two people and every return trip involves one person.

Apply your solution to the following two cases:

(a) The times taken are 1 minute, 1 minute, 3 minutes and 3 minutes.

(b) The times taken are 1 minute, 4 minutes, 4 minutes and 5 minutes.

Hint: In the specific case discussed above (where t_1, t_2, t_3 and t_4 are 1, 2, 5 and 10, respectively), the shortest time is achieved by letting the two slowest cross together. However, this isn't always the best strategy. An alternative strategy is to let the fastest person accompany each of the others across in turn. You will need to evaluate the time taken for both strategies and choose between them on the basis of the times. In order to derive the solution methodically, we suggest the following steps. (Note that steps (a) and (b) were already discussed above.)

(a) How many times must the torch be carried across the bridge in order to get all four people across? (Include crossings in both directions in the count.) How many of these are return trips?

(b) Comparing the number of times a return journey must be made with the number of people, what can you say about the number of people who do not make a return trip? Which of the four people would you choose not to make a return trip? (Give a convincing argument to support your choice.)

(c) Now focus on how to get the people who do not make a return trip across the bridge. What are the different strategies? Evaluate the time taken for each. Hence, construct a formula for the minimum time needed to get all four people across in the general case.

(d) Give a solution to the general problem. Use a conditional statement to decide which strategy to use.

Note that this exercise is much harder than the specific case discussed above. This is not just because the input times are parameters but primarily because you must establish without doubt that your solution cannot be bettered. The original problem was carefully worded so that this was not required.

□

Exercise 3.5 Suppose a brute-force search is used to solve the torch problem (exercise 3.4 above). This would mean enumerating all the different ways of getting four people across the bridge. How many ways are there?

□

3.5 Summary

In this chapter, we have contrasted brute-force search with problem decomposition. Brute-force search should only be used as a last resort. Modern computer technology means that some problems that are too large for human beings to solve do become solvable, but the state-space explosion makes the method impractical for realistic problems, even with the most powerful computers. Complexity theory, which you will study in a course on algorithm design, lends greater force to this argument; no matter how much bigger and faster computers become, they can never compete with the increase in size of the state space caused by modest increases in problem size.

Problem decomposition seeks to exploit the inherent structure of the problem domain. In all river-crossing, or similar, problems, there is a symmetry between the left and right banks. This suggests tackling the problems by decomposing them into three components, the first and last being symmetrical in some way. Unfortunately, this strategy has no guarantee of success, but, if the problems are tackled in this way, they become more manageable, and often have clear, easily remembered and easily reproduced solutions. Most importantly, the solution method can be applied repeatedly, in contrast to the solutions, which are only relevant to one particular problem.

Along the way, the issue of deciding what to name (and what not to name) has emerged as an important problem-solving skill that can have significant impact on the complexity of the problem. The process is called *abstraction* — from the myriad of details that surround any real-world description of a problem, we abstract the few that are relevant, introducing appropriate, clearly defined mathematical notation to assist in the problem's solution.

Chapter 4

Games

This chapter is about how to win some simple two-person games. Games provide very good examples of algorithmic problem solving because playing games is all about winning. The goal is to have some method (i.e. "algorithm") for deciding what to do so that the eventual outcome is a win.

The key to winning is the recognition of invariants. So, in essence, this chapter is a continuation of chapter 2. The chapter is also about trying to identify and exploit structure in problems. In this sense, it introduces the importance of algebra in problem solving.

The next section introduces a number of games with matchsticks, in order to give the flavour of the games that we consider. Following it, we develop a method of systematically identifying winning and losing positions in a game (assuming a number of simplifying constraints on the rules of the game). A *winning strategy* is then what we call "maintaining an invariant". "Maintaining an invariant" is an important technique in algorithm development. Here, it will mean ensuring that the opponent is always placed in a position from which losing is inevitable.

4.1 Matchstick Games

A *matchstick game* is played with one or more piles of matches. Two players take it in turn to make a move. Moves involve removing one or more matches from one of the piles, according to a given rule. The game *ends* when it is no longer possible to make a move. The player whose turn it is to move is the *loser*, and the other player is the *winner*.

A matchstick game is an example of an *impartial*, *two-person* game with *complete information*. "Impartial" means that rules for moving apply equally to both players. (Chess, for example, is not impartial, because white can only move white pieces, and black can only move black pieces.) "Complete information" means that both players know the complete state of the game. In contrast, in card games like poker, it is usual that each player does not know the cards held by the other player; the players have incomplete information about the state of the game.

A *winning position* is one from which a perfect player is always assured of a win. A *losing position* is one from which a player can never win, when playing against a perfect player. A *winning strategy* is an algorithm for choosing moves from winning positions, that guarantees a win.

As an example, suppose there is one pile of matches, and an allowed move is to remove 1 or 2 matches. The losing positions are the positions where the number of matches is a multiple of 3 (that is, the number of matches is 0, 3, 6, 9 etc.). The remaining positions are the winning positions. If m is the number of matches in such a position (so, m is not a multiple of 3), the strategy is to remove $m \bmod 3$ matches[1]. This is either 1 or 2, and so the move is valid. The opponent is then put in a position where the number of matches is a multiple of 3. This means that there are either 0 matches left, in which case the opponent loses, or any move they make will result in there again being a number of matches remaining that is not a multiple of 3.

In an impartial game that is guaranteed to terminate no matter how the players choose their moves (i.e. the possibility of stalemate is excluded), it is always possible to characterise the positions as either winning or losing positions. The following exercises ask you to do this in specific cases.

1. There is one pile of matches. Each player is allowed to remove 1 match. What are the winning positions?

2. There is one pile of matches. Each player is allowed to remove 0 matches. What are the winning positions?

3. Can you see a pattern in the last two problems and the example discussed above (in which a player is allowed to remove 1 or 2 matches)? In other words, can you see how to win a game in which an allowed move is to remove at least one and at most N matches, where N is some number fixed in advance?

4. There is one pile of matches. Each player is allowed to remove 1, 3 or 4 matches. What are the winning positions and what is the winning strategy?

5. There is one pile of matches. Each player is allowed to remove 1, 3 or 4 matches, except that it is not allowed to repeat the opponent's last move. (So, if, say, your opponent removes 1 match, your next move must be to remove 3 or 4 matches.) What are the winning positions and what is the winning strategy?

[1]Recall that $m \bmod 3$ denotes the remainder after dividing m by 3.

6. There are two piles of matches. A move is to choose one pile and, from that pile, remove 1, 2 or 3 matches. What are the winning positions and what is the winning strategy?

7. There are two piles of matches. A move is to choose one pile; from the left pile 1, 2 or 3 matches may be removed, and from the right pile 1 thru[2] 7 matches may be removed. What are the winning positions and what is the winning strategy?

8. There are two piles of matches. A move is to choose one pile; from the left pile, 1, 3 or 4 matches may be removed, and, from the right pile, 1 or 2 matches may be removed. What are the winning positions and what is the winning strategy?

4.2 Winning Strategies

In this section, we formulate what is required of a winning strategy. We begin with the simple matchstick game where a move is to remove one or two matches from a single pile of matches; we show how to search systematically through all the positions of the game, labelling each as either a winning or a losing position. Although a brute-force search, and thus not practical for more complicated games, the algorithm does give a better understanding of what is involved, and can be used as a basis for developing more efficient solutions in particular cases.

4.2.1 Assumptions

We make a number of assumptions about the game, in order that the search will work.

- We assume that the number of positions is finite.
- We assume that the game is guaranteed to terminate no matter how the players choose their moves.

The first assumption is necessary because a one-by-one search of the positions can never be complete if the number of positions is infinite. The second assumption is necessary because the algorithm relies on being able to characterise all positions as either losing or winning; we exclude the possibility that there are stalemate positions. Stalemate positions are ones from which the players can continue the game indefinitely, so that neither wins or loses.

[2]We use "thru" when we want to specify an inclusive range of numbers. For example, "1 thru 4" means the numbers 1, 2, 3 and 4. The English expression "1 to 4" is ambiguous about whether the number 4 is included or not.

4.2.2 Labelling Positions

The first step is to draw a *directed graph* depicting all positions, and all moves in the game. Fig. 4.1 is a graph of the matchstick game described at the beginning of section 4.1.

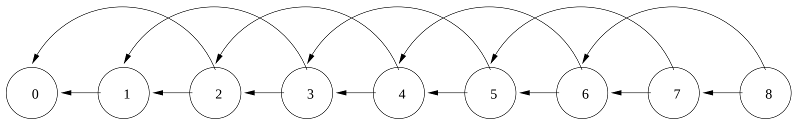


Figure 4.1: Matchstick Game. Players may take one or two matches at each turn.

A directed graph has a set of *nodes* and a set of *edges*. Each edge is *from* one node *to* another node. When graphs are drawn, nodes are depicted by circles, and edges are depicted by arrows pointing from the *from* node to the *to* node.

The nodes in fig. 4.1 are labelled by a number, the number of matches remaining in the pile. From the node labelled 0, there are no edges. It is impossible to move from the position in which no matches remain. From the node labelled 1, there is exactly one edge, to the node labelled 0. From the position in which one match remains, there is only one move that can be made, namely to remove the remaining match. From all other nodes, there are two edges. From the node labelled n, where n is at least 2, there is an edge to the node labelled $n-1$ and an edge to the node labelled $n-2$. That is, from a position in which the number of remaining matches is at least 2, one is allowed to remove one or two matches.

Having drawn the graph, we can begin labelling the nodes as either losing positions or winning positions. A player who finds themself in a losing position will inevitably lose, if playing against a perfect opponent. A player who finds themself in a winning position is guaranteed to win, provided the right choice of move is made at each turn.

The labelling rule has two parts, one for losing positions, the other for winning positions:

- A node is labelled *losing* if *every* edge from the node is to a winning position.
- A node is labelled *winning* if *there is* an edge from the node to a losing position.

At first sight, it may seem that it is impossible to begin to apply these rules; after all, the first rule defines losing positions in terms of winning positions, whilst the second rule does the reverse. It seems like a vicious circle! However, we can begin by labelling as losing positions all the nodes with no outgoing edges. This is because, if there are no edges from a node, the statement "every edge from the node is to a winning position" is true. It is indeed the case that all of the (non-existent) edges is to a winning position.

This is an instance of a general rule of logic. A statement of the form "every x has property p" is what is called a *for-all quantification*, or a *universal quantification*. Such a statement is said to be *vacuously* true when there are no instances of the "x" in the quantification. In a sense, the statement is "vacuous" (i.e. empty) because it is a statement about nothing.

Returning to fig. 4.1, the node 0 is labelled "losing", because there are no edges from it. It is indeed a losing position, because the rules of the game specify that a player who cannot make a move loses.

Next, nodes 1 and 2 are labelled "winning", because, from each, there is an edge to 0, which we know to be a losing position. Note that the edges we have identified dictate the move that should be made from these positions if the game is to be won.

Now, node 3 is labelled "losing", because both edges from node 3 are to nodes (1 and 2) that we have already labelled "winning". From a position in which there are 3 matches remaining, every move is to a position starting from which a win is guaranteed. A player that finds themself in this position will eventually lose.

The process we have described repeats itself until all nodes have been labelled. Nodes 4 and 5 are labelled "winning", then node 6 is labelled "losing", then nodes 7 and 8 are labelled "winning", and so on.

Fig. 4.2 shows the state of the labelling process at the point that node 7 has been labelled but not node 8. The circles depicting losing positions are drawn with thick lines; the circles depicting winning positions are the ones from which there is an edge drawn with a thick line. These edges depict the winning move from that position.

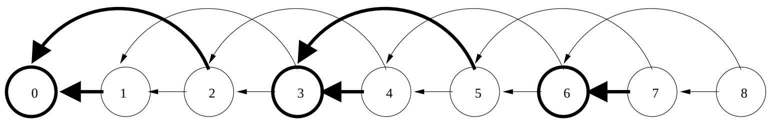


Figure 4.2: Labelling Positions. Winning edges are indicated by thick edges.

Clearly, a pattern is emerging from this process. The pattern is that the losing positions are the ones where the number of matches is a multiple of 3. The winning positions are the remaining positions; the winning strategy is to remove one or two matches so as to leave the opponent in a position where the number of matches is once again a multiple of 3.

4.2.3 Formulating Requirements

The terminology we use to describe the winning strategy is to "maintain invariant" the property that the number of matches is a multiple of 3. In programming terms, we express this property using Hoare triples. Let n denote the number of matches in the pile. Then, the correctness of the winning strategy is expressed by the following annotated program segment:

```
    { $n$ is a multiple of $3$, and $n \neq 0$ }
    if $1 \leq n \rightarrow n := n-1 \;\square\; 2 \leq n \rightarrow n := n-2$ fi
;   { $n$ is not a multiple of $3$ }
    $n := n - (n \bmod 3)$
    { $n$ is a multiple of $3$ }
```

There are five components of this program segment, each on a separate line. The first line is the precondition. This expresses the assumption that we begin from a position in which the number of matches is a multiple of 3, and non-zero.

The second line is a so-called *conditional statement*. Conditional statements are recognised by "if - fi" brackets. Within these brackets is a non-deterministic choice — indicated by the "$\square$" symbol— among a number of so-called *guarded commands*. A guarded command has the form $b \rightarrow S$, where b is a boolean-valued expression called the *guard*, and S is a statement called the *body*. Starting in a given state, a conditional statement is executed by choosing a guarded command whose guard evaluates to true, and then executing its body. If several guards evaluate to true, an arbitrary choice of command is made. If none of the guards evaluates to true, execution is aborted[3].

In this way, the if - fi statement in the second line models an arbitrary move. Removing one match is only allowed if $1 \leq n$; hence, the statement $n := n-1$ is "guarded" by this condition. Similarly, removing two matches —modelled by the assignment $n := n-2$— is "guarded" by the condition $2 \leq n$. At least one of these guards, and possibly both, is true because of the assumption that $n \neq 0$.

The postcondition of the guarded command is the assertion "n is not a multiple of 3". The triple, comprising the first three lines, thus asserts that, if the number of matches is a multiple of 3, and a valid move is made that reduces the number of matches by one or two, then, on completion of the move, the number of matches will not be a multiple of 3.

The fourth line of the sequence is the implementation of the winning strategy; specifically, remove $n \bmod 3$ matches. The fifth line is the final postcondition, which asserts that, after execution of the winning strategy, the number of matches will again be a multiple of 3.

[3]If you are already familiar with a conventional programming language, you will be familiar with *deterministic* conditional statements — so-called if-then-else statements. In such statements, the choice of which of the optional statements should be executed is completely determined by the state of the program variables. In a non-deterministic choice, as used here, the choice is not completely determined.

In summary, beginning from a state in which n *is* a multiple of 3, and making an arbitrary move, results in a state in which n is *not* a multiple of 3. Subsequently, removing $n \bmod 3$ matches results in a state in which n is again a multiple of 3.

In general, a winning strategy is obtained by characterising the losing positions by some property, $losing$ say. The end positions (the positions where the game is over) must satisfy the property $losing$. The winning positions are then the positions that do not satisfy $losing$. For each winning position, one has to identify a way of calculating a losing position to which to move; the algorithm that is used is the *winning strategy.* More formally, the losing and winning positions, and the winning strategy must satisfy the following specification.

```
      { losing position, and not an end position }
      make an arbitrary (legal) move
   ;  { winning position, i.e. not a losing position }
      apply winning strategy
      { losing position }
```

In summary, a winning strategy is a way of choosing moves that divides the positions into two types, the losing positions and the winning positions, in such a way that the following three properties hold:

- End positions are losing positions.
- From a losing position that is not an end position, every move is to a winning position.
- From a winning position, it is always possible to apply the winning strategy, resulting in a losing position.

If both players are perfect, the winner is decided by the starting position. If the starting position is a losing position, the second player is guaranteed to win. Vice-versa, if the starting position is a winning position, the first player is guaranteed to win. Starting from a losing position, one can only hope that one's opponent is not perfect, and will make a mistake.

Formally, a winning strategy maintains invariant the boolean quantity

> (the number of moves remaining before the game ends is even) equals (the position is a losing position).

Since end positions are losing positions, by assumption, and positions where the number of moves remaining is zero, which is an even number, this quantity will always be true in a game played by perfect players.

We recommend that you now try to solve the matchstick-game problem when the rule is that any number of matches from 1 thru M may be removed at each turn. The number M is a natural number, fixed in advance. We recommend that you try to solve this general problem by first considering the case that M is 0. This case has a very easy solution, although it is a case that is very often neglected. Next, consider the case that M is 1. This case also has an easy solution, but slightly more complicated. Now, combine these two cases with the case that M is 2, which is the case we have just considered. Do you see a pattern in the solutions? If you don't see a pattern immediately, try a bit harder. As a last resort, try working out the case that M is 3. (Don't draw a diagram. Construct a table instead. A diagram is much too complicated.) Then, return to the cases that M is 0, 1 and 2 (in particular, the extreme cases 0 and 1) in order to check the pattern you have identified. Finally, formulate the correctness of the strategy by a sequence of assertions and statements, as we did above for the case that M is 2.

Exercise 4.1 (31st December Game) Two players alternately name dates. The winner is the player who names 31st December, and the starting date is 1st January.

Each part of this exercise uses a different rule for the dates that a player is allowed to name. For each, devise a winning strategy, stating which player should win. State also if it depends on whether the year is a leap year or not.

Hint: in principle, you have to determine for each of 365 days (or 366 in the case of a leap year) whether naming the day results in losing against a perfect player. In practice, a pattern soon becomes evident and the days in each month can be grouped together into winning and losing days. Begin by identifying the days in December that one should avoid naming.

a) (Easy) A player can name the 1st of the next month, or increase the day of the month by an arbitrary amount. (For example, the first player begins by naming 1st February, or a date in January other than the 1st.)

b) (Harder) A player can increase the day by one, leaving the month unchanged, or name the 1st of the next month.

□

4.3 Subtraction-Set Games

A class of matchstick games is based on a single pile of matches and a (finite) set of numbers; a move is to remove m matches, where m is an element of the given set. A game in this class is called a *subtraction-set game*, and the set of numbers is called the *subtraction set*.

The games we have just discussed are examples of matchstick games; if the rule is that 1 thru M matches may be removed at each turn, the subtraction set is $\{1..M\}$. More interesting examples are obtained by choosing a subtraction set with less regular structure.

For any given subtraction set, the winning and losing positions can always be computed. We exemplify the process in this section by calculating the winning and losing positions when the allowed moves are:

- remove one match,
- remove three matches,
- remove four matches.

In other words, the subtraction set is $\{1,3,4\}$.

Positions in the game are given by the number of matches in the pile. We refer to the positions using this number. So, "position 0" means the position in which there are no matches remaining in the pile, "position 1" means the position in which there is just one match in the pile, and so on.

Beginning with position 0, and working one-by-one through the positions, we identify whether each position is a winning position using the rules that

- a position is a losing position if every move from it is to a winning position, and
- a position is a winning position if there is a move from it to a losing position.

The results are entered in a table. Table 4.1 shows the entries when the size of the pile is at most 6. The top row is the position, and the middle row shows whether or not it is a winning (W) or losing position (L). In the case that the position is a winning position, the bottom row shows the number of matches that should be removed in order to move from the position to a losing position. For example, 2 is a losing position because the only move from 2 is to 1; positions 3 and 4 are winning positions because from both a move can be made to 0. Note that there may be a choice of winning move. For example, from position 3 there are two winning moves — remove 3 matches to move to position 0, or remove 1 match to move to position 2. It suffices to enter just one move in the bottom row of the table.

Continuing this process, we get the next seven entries in the table: see table 4.2.

Comparing tables 4.1 and 4.2, we notice that the pattern of winning and losing positions repeats itself. Once the pattern begins repeating in this way, it will continue to do so forever. We may therefore conclude that, for the subtraction set $\{1,3,4\}$, whether or not the position is a winning position can be determined by computing the remainder, r say, after dividing the number of matches by 7. If r is 0 or 2, the position is a losing

Position	0	1	2	3	4	5	6
Type	L	W	L	W	W	W	W
Move		1		3	4	3	4

Table 4.1: Winning (W) and Losing (L) Positions for subtraction set $\{1,3,4\}$

Position	7	8	9	10	11	12	13
Type	L	W	L	W	W	W	W
Move		1		3	4	3	4

Table 4.2: Winning (W) and Losing (L) Positions for subtraction set $\{1,3,4\}$

position. Otherwise, it is a winning position. The winning strategy is to remove 1 match if r is 1, remove 3 matches if r is 3 or 5, and remove 4 matches if r is 4 or 6.

The repetition in the pattern of winning and losing positions that is evident in this example is a general property of subtraction-set games, with the consequence that, for a given subtraction set, it is always possible to determine for an arbitrary position whether or not it is a winning position (and, for the winning positions, a winning move). The following argument gives the reason why.

Suppose a subtraction set is given. Since the set is assumed to be finite, it must have a largest element. Let this be M. Then, from each position, there are at most M moves. For each position k, let $W.k$ be true if k is a winning position, and false otherwise. When k is at least M, $W.k$ is completely determined by the sequence $W.(k-1)$, $W.(k-2)$, $\ldots$, $W.(k-M)$. Call this sequence $s.k$. Now, there are only 2^M different sequences of booleans of length M. As a consequence, the sequence $s.(M+1)$, $s.(M+2)$, $s.(M+3)$, $\ldots$ must eventually repeat, and it must do so within at most 2^M steps. That is, for some j and k, with $M \leq j < k < M+2^M$, we must have $s.j = s.k$. It follows that $W.j = W.k$ and the sequence W repeats from the kth position onwards.

For the example above, this analysis predicts that the W-L pattern will repeat from the 20th position onwards. In fact, it begins repeating much earlier. Generally, we can say that the pattern of win-lose positions will repeat at position 2^M+M, *or before.* To determine whether an arbitrary position is a winning or losing position involves computing the status of each position k, for successive values of k, until a repetition in $s.k$ is observed. If the repetition occurs at position R, then, for an arbitrary position k, $W.k$ equals $W.(k \bmod R)$.

Exercise 4.2 Suppose there is one pile of matches. In each move, 2, 5 or 6 matches may be removed. (That is, the subtraction set is $\{2,5,6\}$.)

(a) For each n, $0 \leq n < 22$, determine whether a pile of n matches is a winning or losing position.

(b) Identify a pattern in the winning and losing positions. Specify the pattern by giving precise details of a boolean function of n that determines whether a pile of n matches is a winning position or not.

Verify the pattern by constructing a table showing how the function's value changes when a move is made.

□

Exercise 4.3 This exercise is challenging; its solution involves thinking beyond the material presented in the rest of the chapter.

Figure 4.3 shows a variant of snakes and ladders. In this game, there is just one counter. The two players take it in turn to move the counter at most four spaces forward. The start is square 1 and the finish is square 25; the winner is the first to reach the finish. As in the usual game of snakes and ladders, if the counter lands on the head of a snake, it falls down to the tail of the snake; if the counter lands at the foot of a ladder, it climbs to the top of the ladder.

(a) List the positions in this game. (These are not the same as the squares. Think carefully about squares linked by a snake or a ladder.)

(b) Identify the winning and losing positions. Use the rule that a losing position is one from which every move is to a winning position, and a winning position is one from which there is a move to a losing position.

(c) Some of the positions cannot be identified as winning or losing in this way. Explain why.

□

4.4 Sums of Games

In this section, we look at how to exploit the structure of a game in order to compute a winning strategy more effectively.

The later examples of matchstick games in section 4.1 have more than one pile of matches. When a move is made, one of the piles must first be chosen; then, matches may be removed from the chosen pile according to some prescribed rule, which may differ from pile to pile. The game is thus a combination of two games; this particular way of combining games is called *summing* the games.

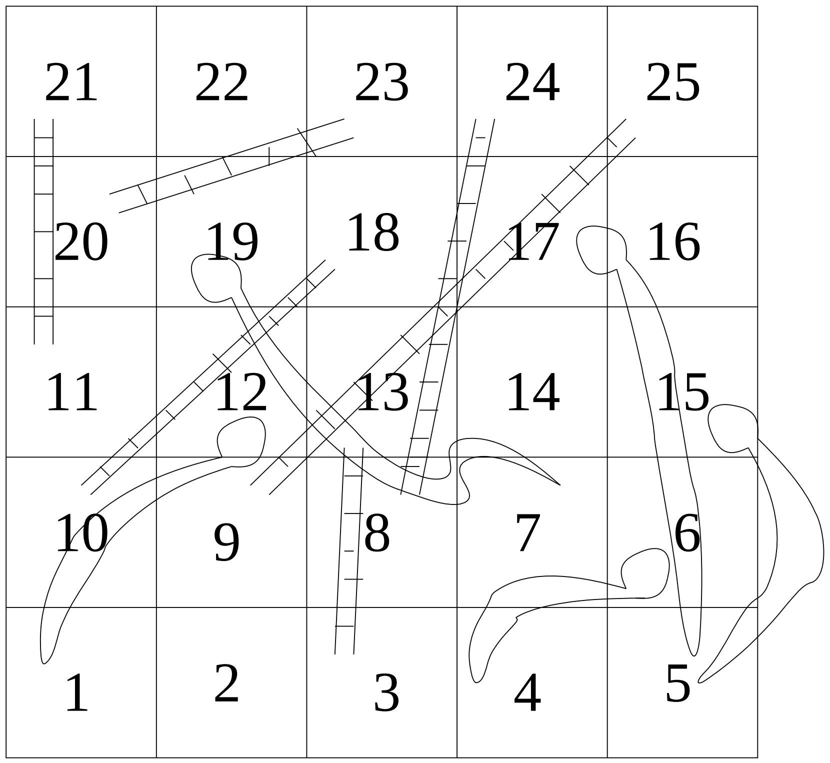


Figure 4.3: Snakes and Ladders. Players take it in turn to move the counter at most four spaces forward.

In general, given two games each with its own rules for making a move, the *sum* of the games is the game described as follows. For clarity, we call the two games the *left* and the *right* game. A position in the sum game is the combination of a position in the left game and a position in the right game. A move in the sum game is a move in one of the games.

Figure 4.4 is an example of the sum of two games. Each graph represents a game, where the positions are represented by the nodes, and the moves are represented by the edges. Imagine a coin placed on a node. A move is then to displace the coin along one of the edges to another node. The nodes in the left graph and right graphs are named by capital letters and small letters, respectively, so that we can refer to them later.

In the “sum” of the games, two coins are used, one coin being placed over a node in each of the two graphs. A move is then to choose one of the coins, and displace it along an edge to another node. Thus, a position in the “sum” of the games is given by a pair Xx where “X” names a node in the left graph, and “x” names a node in the right graph; a move has the effect of changing exactly one of “X” or “x”.

Both the left and right games in fig. 4.4 are unstructured; consequently, the brute-

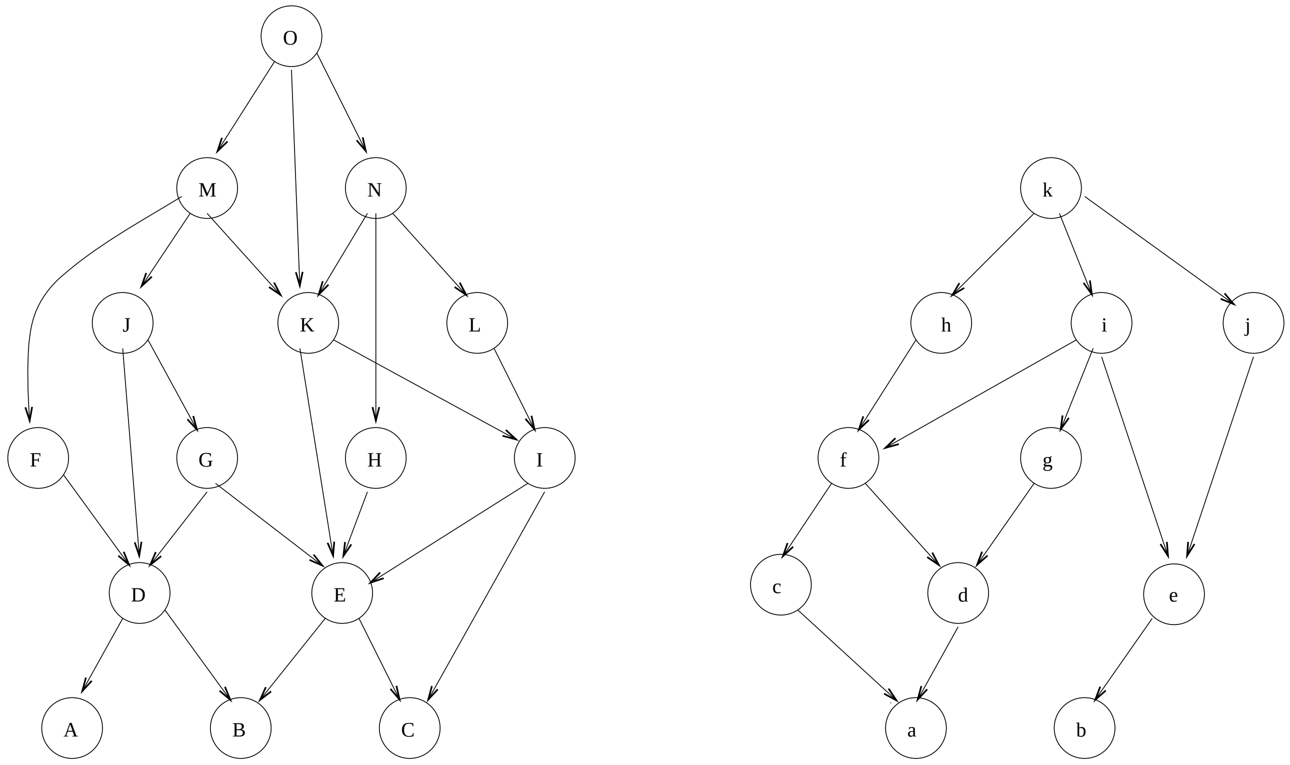


Figure 4.4: A Sum Game. The left and right games are represented by the two graphs. A position is a pair Xx where "X" is the name of a node in the left graph, and "x" is the name of a node in the right graph. A move changes exactly one of X or x.

force search procedure described in section 4.2.2 is unavoidable when determining their winning and losing positions. However, the left game in fig. 4.4 has 15 different postions, and the right game has 11 ; thus, the sum of the two games has $15{\times}11$ different positions. For this game, and for sums of games in general, a brute-force search is highly undesirable. In this section, we study how to compute a winning strategy for the sum of two games. We find that the computational effort is the sum (in the usual sense of addition of numbers) of the effort required to compute winning and losing positions for the component games, rather than the product. We find, however, that it is not sufficient to know just the winning strategy for the individual games. Deriving a suitable generalisation forms the core of the analysis.

4.4.1 Symmetry

A Simple Sum Game

We begin with a very simple example of the sum of two games. Suppose there are two piles of matches. An allowed move is to choose any one of the piles and remove at least one match from the chosen pile. Otherwise, there is no restriction on the number of matches that may be removed. As always, the game is lost when a player cannot make a move.

This game is the "sum" of two instances of the same, very, very simple, game: given a (single) pile of matches, a move is to remove at least one match from the pile. In this simple game, the winning positions are, obviously, the positions in which the pile has at least one match, and the winning strategy is to remove all the matches. The position in which there are no matches remaining is the only losing position.

It quickly becomes clear that knowing the winning strategy for the individual games is insufficient to win the sum of the games. If a player removes all the matches from one pile —that is, applies the winning strategy for the individual game— , the opponent wins by removing the remaining matches in the other pile.

The symmetry between left and right allows us to easily spot a winning strategy. Suppose we let m and n denote the number of matches in the two piles. In the end position, there is an equal number of matches in both piles, namely 0. That is, in the end position, $m=n=0$. This suggests that the losing positions are given by $m=n$. This is indeed the case. From a position in which $m=n$, and a move is possible (that is, either $1\leq m$ or $1\leq n$), any move will be to a position where $m\neq n$. Subsequently, choosing the pile with the larger number of matches, and removing the excess matches from this pile, will restore the property that $m=n$.

Formally, the correctness of the winning strategy is expressed by the following sequence of assertions and program statements.

```
    {  $m=n \wedge (m\neq 0 \vee n\neq 0)$  }
    if  $1\leq m \rightarrow$ reduce $m$
    □   $1\leq n \rightarrow$ reduce $n$
    fi
;   {  $m\neq n$  }
    if  $m<n \rightarrow n := n-(n-m)$
    □   $n<m \rightarrow m := m-(m-n)$
    fi
```

$\{\ m=n\ \}$

The non-deterministic choice between reducing m, in the case that $1 \leq m$, and reducing n, in the case that $1 \leq n$, models an arbitrary choice of move in the sum game. The fact that either m changes in value, or n changes in value, but not both, guarantees $m \neq n$ after completion of the move.

The property $m \neq n$ is the precondition for the winning strategy to be applied. Equivalently, $m < n$ or $n < m$. In the case that $m < n$, we infer that $1 \leq n-m \leq n$, so that $n-m$ matches can be removed from the pile with n matches. Since, $n-(n-m)$ simplifies to m, it is clear that, after the assignment $n := n-(n-m)$, the property $m = n$ will hold. The case $n < m$ is symmetric.

The following sequence of assertions and program statements summarises the argument just given for the validity of the winning strategy. Note how the two assignments have been annotated with a precondition and a postcondition. The precondition expresses the legitimacy of the move; the postcondition is the *losing* property that the strategy is required to establish.

```
{ $m \neq n$ }
{ $m < n \vee n < m$ }
if  $m < n \rightarrow \{ 1 \leq n-m \leq n \}\ n := n-(n-m)\ \{ m=n \}$
□  $n < m \rightarrow \{ 1 \leq m-n \leq m \}\ m := m-(m-n)\ \{ m=n \}$
fi
{ $m = n$ }
```

4.4.2 Maintain Symmetry!

The game in section 4.4.1 is another example of the importance of symmetry; the winning strategy is to ensure that the opponent is always left in a position of symmetry between the two individual components of the sum-game. We see shortly that this is how to win all sum-games, no matter what the individual components are.

There are many examples of games where symmetry is the key to winning. Here is a couple. The solutions can be found at the end of the book.

The Daisy Problem Suppose a daisy has 16 petals arranged symmetrically around its centre. There are two players. A move involves removing one petal or two adjacent petals. The winner is the one who removes the last petal. Who should win and what is the winning strategy? Generalise your solution to the case that there are initially n petals and a move consists of removing between 1 and M adjacent petals (where M is fixed in advance of the game).

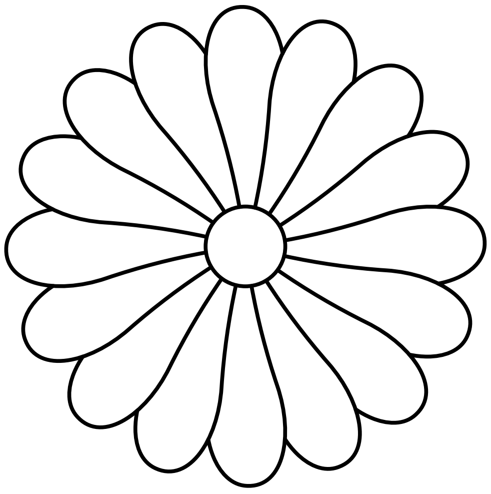


Figure 4.5: A 16-petal daisy

The Coin Problem Two players are seated at a rectangular table which initially is bare. They each have an unlimited supply of circular coins of varying diameter. The players take it in turns to place a coin on the table, such that it does not overlap any coin already on the table. The winner is the one who puts the last coin on the table. Who should win and what is the winning strategy? (*Harder*) What, if anything, do you assume about the coins in order to justify your answer?

4.4.3 More Simple Sums

Let us return to our matchstick games. A variation on the sum game in section 4.4.1 is to restrict the number of matches that can be removed. Suppose the restriction is that at most K matches can be removed from either pile (where K is fixed, in advance).

The effect of the restriction is to disallow some winning moves. If, as before, m and n denote the number of matches in the two piles, it is not allowed to remove $m-n$ matches when $K<m-n$. Consequently, the property $m=n$ no longer characterises the losing positions. For example, if K is fixed at 1, the position in which one pile has two matches whilst the second pile has no matches is a losing position: in this position a player is forced to move to a position in which one match remains; the opponent can then remove the match to win the game.

A more significant effect of the restriction seems to be that the strategy of establishing symmetry is no longer applicable. Worse is if we break symmetry further by imposing different restrictions on the two piles: suppose, for example, we impose the limit M on the number of matches that may be removed from the left pile, and N on the number of matches that may be removed from the right pile, where $M\neq N$. Alternatively, suppose the left and right games are completely different, for example, if one is a matchstick game and the other is the daisy game. If this is the case, how is it possible to maintain symmetry? Nevertheless, a form of "symmetry" is a key to the winning strategy: symmetry is too important to abandon so easily!

We saw, in section 4.2, that the way to win the one-pile game, with the restriction that at most M matches can be removed, is to continually establish the property that the remainder after dividing the number of matches by $M+1$ is 0. Thus, for a pile of m matches, the number $m \bmod (M+1)$ determines whether the position is a winning position or not. This suggests that, in the two-pile game, "symmetry" between the piles is formulated as the property that

$$m \bmod (M+1) = n \bmod (N+1) \quad .$$

(M is the maximum number of matches that can be removed from the left pile, and N is the maximum number that can be removed from the right pile.)

This, indeed, is the correct solution. In the end position, where both piles have 0 matches, the property is satisfied. Also, the property can always be maintained following an arbitrary move by the opponent, as given by the following annotated program segment.

```
    { $m \bmod (M+1) = n \bmod (N+1) \;\wedge\; (m \neq 0 \vee n \neq 0)$ }
    if  $1 \leq m \rightarrow$ reduce $m$ by at most $M$
    □  $1 \leq n \rightarrow$ reduce $n$ by at most $N$
    fi
;   { $m \bmod (M+1) \neq n \bmod (N+1)$ }
    if  $m \bmod (M+1) < n \bmod (N+1) \rightarrow n := n - (n \bmod (N+1) - m \bmod (M+1))$
    □  $n \bmod (N+1) < m \bmod (M+1) \rightarrow m := m - (m \bmod (M+1) - n \bmod (N+1))$
    fi
    { $m \bmod (M+1) = n \bmod (N+1)$ }
```

(Note: we discuss later the full details of how to check the assertions made in this program segment.)

4.4.4 The MEX Function

The idea of defining "symmetric" to be "the respective remainders are equal" can be generalised to an arbitrary sum game.

Consider a game that is the sum of two games. A position in the sum game is a pair (l,r) where l is a position in the left game, and r is a position in the right game. A move affects just one component; so, a move is modelled by either a (guarded) assignment $l := l'$ (for some l') to the left component or a (guarded) assignment $r := r'$ (for some r') to the right component.

The idea is to define two functions L and R, say, on left and right positions, respectively, in such a way that a position (l,r) is a losing position exactly when $L.l = R.r$. The question is: what properties should these functions satisfy? In other words, how do we specify the functions L and R?

The analysis given earlier of a winning strategy allows us to distill the specification.

First, since (l,r) is an end position of the sum game exactly when l is an end position of the left game and r is an end position of the right game, it must be the case that L and R have equal values on end positions.

Second, every allowed move from a losing position —a position (l,r) satisfying $L.l = R.r$— , that is not an end position, should result in a winning position —a position (l,r) satisfying $L.l \neq R.r$— . That is,

```
{  $L.l = R.r \wedge (l$ is not an end position $\vee\ r$ is not an end position$)$  }
if   $l$ is not an end position $\rightarrow$ change $l$
□    $r$ is not an end position $\rightarrow$ change $r$
fi
{  $L.l \neq R.r$  }  .
```

Third, applying the winning strategy, from a winning position —a position (l,r) satisfying $L.l \neq R.r$— should result in a losing position —a position (l,r) satisfying $L.l = R.r$—. That is,

```
{  $L.l \neq R.r$  }
apply winning strategy
{  $L.l = R.r$  }  .
```

We can satisfy the first and second requirements if we define L and R to be functions with range the set of natural numbers, and require that:

- For end positions l and r of the respective games, $L.l = 0 = R.r$.
- For every l' such that there is a move from l to l' in the left game, $L.l \neq L.l'$. Similarly, for every r' such that there is a move from r to r' in the right game, $R.r \neq R.r'$.

Note that the choice of the natural numbers as range of the functions, and the choice of 0 as the functions' value at end positions is quite arbitrary. The advantage of this choice arises from the third requirement. If $L.l$ and $R.r$ are different natural numbers, either $L.l < R.r$ or $R.r < L.l$. This allows us to refine the process of applying the winning strategy, by choosing to move in the right game when $L.l < R.r$ and choosing to move in the left game when $R.r < L.l$. (See below.)

```
{ $L.l \neq R.r$ }
if $L.l < R.r \rightarrow$ change $r$
□ $R.r < L.l \rightarrow$ change $l$
fi
{ $L.l = R.r$ } .
```

For this to work, we require that:

- For any number m less than $R.r$, it is possible to move from r to a position r' such that $R.r' = m$. Similarly, for any number n less than $L.l$, it must be possible to move from l to a position l' such that $L.l' = n$.

The bulleted requirements are satisfied if we define the functions L and R to be the so-called "mex" function. The precise definition of this function is as follows.

Let p be a position in a game G. The mex value of p, denoted $mex_G.p$, is defined to be the smallest natural number, n, such that

- There is no legal move in the game G from p to a position q satisfying $mex_G.q = n$.
- For every natural number m less than n, there is a legal move in the game G from p to a position q satisfying $mex_G.q = m$.

"Mex" is short for "minimal excludant". A brief, informal description of the mex number of a position p is the minimum number that is excluded from the mex numbers of positions q to which a move can be made from p.

4.4.5 Using the MEX Function

We use the game depicted in fig. 4.4 to illustrate the calculation of mex numbers. Figure 4.6 shows the mex numbers of each of the nodes in their respective games.

The graphs do not have any systematic structure; consequently, the only way to compute the mex numbers is by a brute-force search of all positions. This is easily done

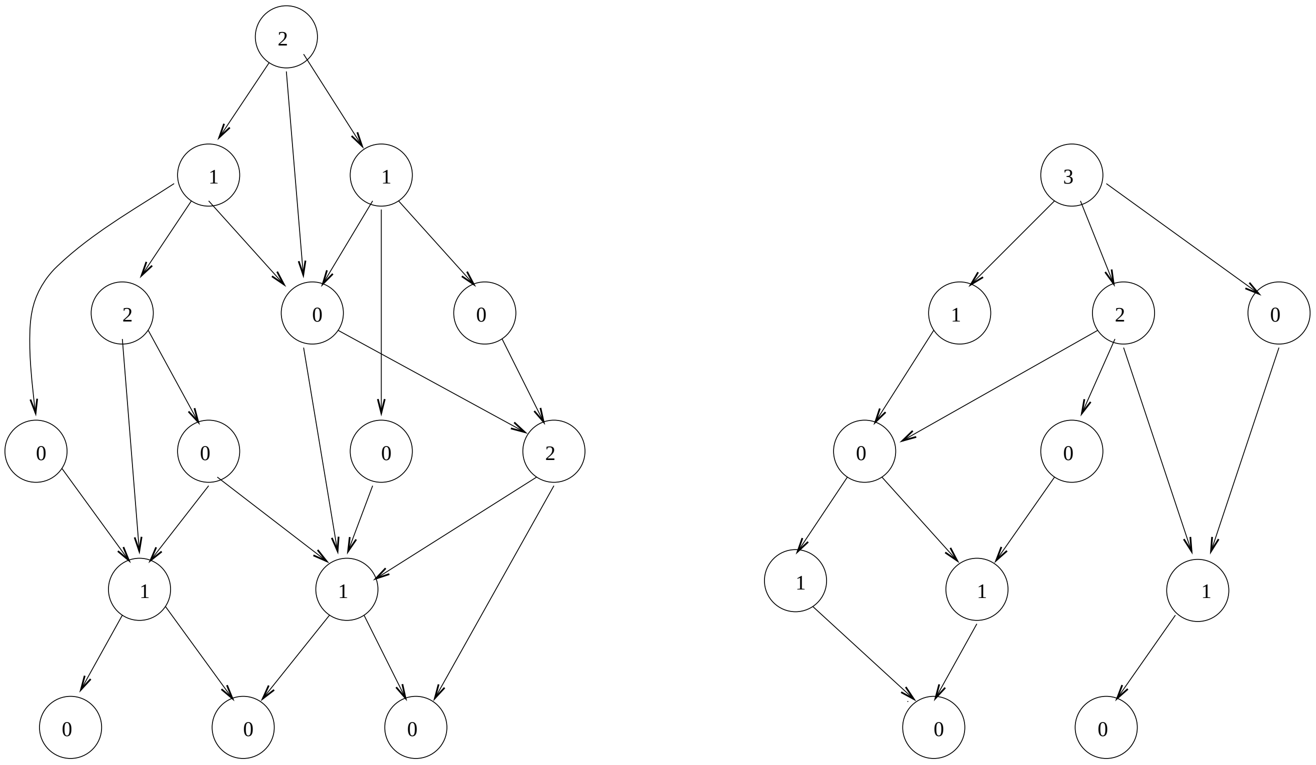


Figure 4.6: Mex Numbers. The mex number of a node is the smallest natural number not included among the mex numbers of its successors.

by hand. The end positions are each given mex number 0. Subsequently, a mex number can be given to a node when all its successors have already been given a mex number. (A successor of a node p is a node q such that there is an edge from p to q.) The number is, by definition, the smallest number that is not included in the mex numbers of its successors. Fig. 4.7 shows a typical situation. The node at the top of the figure is given a mex number when all its successors have been given mex numbers. In the situation shown, the mex number given to it is 2 because none of its successors have been given this number, but there are successors with the mex numbers 0 and 1.

Now, suppose we play this game. Let us suppose the starting position is “Ok”. This is a winning position because the mex number of “O” is different from the mex number of “k”. The latter is larger (3 against 2). So, the winning strategy is to move in the right graph to the node “i”, which has the same mex number as “O”. The opponent is then obliged to move, in either the left or right graph, to a node with mex number different from 2. The first player then repeats the strategy of ensuring that the mex numbers are equal, until eventually the opponent can move no further.

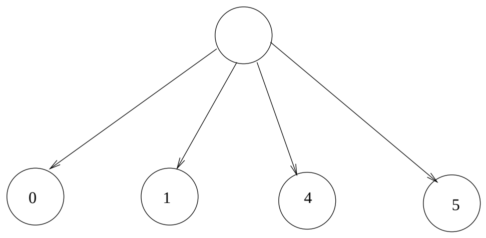


Figure 4.7: Computing mex numbers. The unlabelled node is given the mex number 2.

Note that, because of the lack of structure of the individual games, we have to search through all 15 positions of the left game and all 11 positions of the right game, in order to calculate the mex numbers of each position. In total, therefore, we have to search through 26 different positions. But, this is just the sum (in the usual sense of the word) of 15 and 11, and is much less than their product, 165. This is a substantial saving in computational effort. Moreover, the saving grows as the size of the component games increases.

Exercise 4.4 a) Consider the subtraction-set game where there is one pile of matches from which at most 2, 5 or 6 matches may be removed. Calculate the mex number for each position until you spot a pattern.
b) Consider a game which is the sum of two games. In the left game, 1 or 2 matches may be removed at each turn. In the right game, 2, 5 or 6 matches may be removed. In the sum game, a move is made by choosing to play in the left game, or choosing to play in the right game.

The table below shows a number of different positions in this game. A position is given by a pair of numbers: the number of matches in the left pile, and the number of matches in the right pile.

Left Game	Right Game	"losing" or winning move
10	20	?
20	20	?
15	5	?
6	9	?
37	43	?

Table 4.3: Fill in entries marked "?"

For each position, state whether it is a winning or a losing position. For winning positions, give the winning move in the form $X\,m$ where "X" is one of "L" (for "left game") or "R" (for right game), and m is the number of matches to be removed.

□

Exercise 4.5 A rectangular board is divided into m horizontal rows and n vertical columns, where m and n are both strictly positive integers, making a total of $m{\times}n$ squares. The number of squares is called the *area* of the board.

A game is played on the board as follows. Each of the two players takes it in turn to cut the board either horizontally or vertically along one of the dividing lines. A cut divides the board into two parts; when a cut has been made a part whose area is at most the area of the other part is discarded. (This means that the part with the smaller area is discarded if the two parts have different areas, and one of the two parts is chosen arbitrarily if the two areas are equal.) For example, if the board has $4{\times}5$ squares, a single move reduces it to $2{\times}5$, $3{\times}5$, $4{\times}3$, or $4{\times}4$ squares. Also, if the board has $4{\times}4$ squares, a single move reduces it to either $2{\times}4$ or $3{\times}4$ squares. (Boards with $3{\times}4$ and $4{\times}3$ squares are effectively the same; the orientation of the board is not significant.) The game ends when the board has been reduced to a $1{\times}1$ board. At this point, the player whose turn it is to play loses.

This game is a sum game because, at each move, a choice is made between cutting horizontally or vertically. The component games are copies of the same game. This game is as follows. A position in the game is given by a strictly positive integer m. A move in the game is to replace m by a number n such that $n<m\leq 2n$; the game ends when m has been reduced to 1, at which point the player whose turn it is to play loses. For example, if the board has 5 columns, the number of columns can be reduced to 3 or 4 because $3<5\leq 6$ and $4<5\leq 8$. No other moves are possible because, for n less than 3, $2n<5$, and for n greater than 4, $5\leq n$.

The game is easy to win if it is possible to make the board square. This question is about calculating the mex numbers of the component games in order to determine a winning move even when the board cannot be made square.

(a) For the component game, calculate which positions are winning and which positions are losing for the first 15 positions. Make a general conjecture about the winning and losing positions in the component game and prove your conjecture.

Base your proof on the following facts. The end position, position 1, is a losing position. A winning position is a position from which there is a move to a losing position. A losing position is a position from which every move is to a winning position.

(b) For the component game, calculate the mex number of each of the first 15 positions.

Give the results of your calculation in the form of a table with two rows. The first row is a number m and the second row is the mex number of position m. Split the table into four parts. Part i gives the mex numbers of positions 2^i thru $2^{i+1}-1$ (where i begins at 0) as shown below. (The first three entries have been completed as illustration.)

Position:	1
Mex Number:	0

Position:	2	3
Mex Number:	1	0

Position:	4	5	6	7
Mex Number:	?	?	?	?

Position:	8	9	10	11	12	13	14	15
Mex Number:	?	?	?	?	?	?	?	?

(You should find that the mex number of each of the losing positions (identified in part (a)) is 0. You should also be able to observe a pattern in the way entries are filled in for part $i+1$ knowing the entries for part i. The pattern is based on whether the position is an even number or an odd number.)

(c) Table 4.3 shows a position in the board game; the first column shows the number of columns and the second column the number of rows. Using your table of mex numbers, or otherwise, fill in "losing" if the position is a losing position. If the position is not a losing position, fill in a winning move either in the form "Cn" or "Rn", where n is an integer; "C" or "R" indicates whether the move is to reduce the number of "C"olumns or the number of "R"ows, and n is the number which it should become.

□

No. of Columns	No. of Rows	"losing" or winning move
2	15	?
4	11	?
4	14	?
13	6	?
21	19	?

Table 4.4: Fill in entries marked "?"

4.5 Summary

This chapter has been about determining winning strategies in simple two-person games. The underlying theme of the chapter has been problem *specification*. We have seen how winning and losing positions are specified. A precise, formal specification enabled us to formulate a brute-force search procedure to determine which positions are which.

Brute-force search is only advisable for small, unstructured problems. The analysis of the "sum" of two games exemplifies the way structure is exploited in problem solving. Again, the focus was on problem specification. By formulating a notion of "symmetry" between the left and right games, we were able to determine a specification of the "mex" function on game positions. The use of mex functions substantially reduces the effort needed to determine winning and losing positions in the "sum" of two games, compared to a brute-force search.

Game theory is a rich, well-explored area of Mathematics, which we have only touched upon in this chapter. It is a theory that is becoming increasingly important in Computing Science. One reason for this is that problems that beset software design, such as the security of a system, are often modelled as a game, with the user of the software as the adversary. Another reason is that games often provide excellent examples of "computational complexity"; it is easy to formulate games having very simple rules but for which no efficient algorithm implementing the winning strategy is known.

Mex numbers were introduced by Sprague and Grundy to solve the "Nim" problem, and mex numbers are sometimes called "Sprague-Grundy" numbers, after their originators. Nim is a well-known matchstick game involving three piles of matches. We have not developed the theory sufficiently, in this chapter, to show how Nim, and sums of more than two games, are solved using mex numbers. (What is missing is how to compute the mex number of the sum of two games.)

4.6 Bibliographic Remarks

The two-volume book “Winning Ways” [BCG82] by Berlekamp, Conway and Guy is the bible of game theory.

The 31st December game (exercise 4.1) is adapted from [DW00]. Exercise 4.5 was suggested by Atheer Aheer.

Chapter 5

Knights and Knaves

The island of knights and knaves is a fictional island that is often used to test students' ability to reason logically. The island has two types of natives, "knights" who always tell the truth, and "knaves" who always lie. Logic puzzles involve deducing facts about the island from statements made by its natives without knowing whether or not the statements are made by a knight or a knave.

The temptation is to solve such problems by case analysis —in a problem involving n natives, consider the 2^n different cases obtained by assuming that the individual natives are knights or knaves— . Case analysis is a clumsy way of tackling the problems. In contrast, these, and similar logic puzzles, are easy exercises in the use of *calculational logic*, which we introduce in this chapter.

5.1 Logic Puzzles

Here is a typical collection of knights-and-knaves puzzles.

1. It is rumoured that there is gold buried on the island. You ask one of the natives whether there is gold on the island. The native replies: "There is gold on this island is the same as I am a knight." The problem is

   (a) Can it be determined whether the native is a knight or a knave?

   (b) Can it be determined whether there is gold on the island?

2. Suppose you come across two of the natives. You ask both of them whether the other one is a knight. Will you get the same answer in each case?

3. There are three natives A, B and C. Suppose A says "B and C are the same type." What can be inferred about the number of knights?

4. Suppose C says “A and B are as like as two peas in a pod”. What question should you pose to A to determine whether or not C is telling the truth?

5. Devise a question that allows you to determine whether a native is a knight.

6. What question should you ask A to determine whether B is a knight?

7. What question should you ask A to determine whether A and B are the same type (i.e. both knights or both knaves)?

8. You would like to determine whether an odd number of A, B and C is a knight. You may ask one yes/no question to any one of them. What is the question you should ask?

9. A tourist comes to a fork in the road, where one branch leads to a restaurant and one doesn’t. A native of the island is standing at the fork. Formulate a single yes/no question that the tourist can ask such that the answer will be yes if the left fork leads to the restaurant, and otherwise the answer will be no.

5.2 Calculational Logic

5.2.1 Propositions

The algebra we learn at school is about calculating with expressions whose values are numbers. We learn, for example, how to manipulate an expression like m^2-n^2 in order to show that its value is the same as the value of $(m+n)\times(m-n)$, independently of the values of m and n. We say that m^2-n^2 and $(m+n)\times(m-n)$ are *equal*, and write

$$m^2-n^2 = (m+n)\times(m-n) \quad .$$

The basis for these calculations is a set of *laws*. Laws are typically primitive, but general, equalities between expressions. They are “primitive” in the sense that they cannot be broken down into simpler laws, and they are “general” in the sense that they hold independently of the values of any variables in the constituent expressions. We call them *axioms*. Two examples of axioms, both involving zero are:

$$n+0 = n \quad ,$$

and

$$n-n = 0 \quad ,$$

both of which are true whatever the value of the variable n. We say they are true "for all n". The laws are often given names so that we can remember them more easily. For example, "associativity of addition" is the name given to the equality:

$$(m+n)+p \;=\; m+(n+p) \quad ,$$

which is true for all m, n and p.

(Calculational) logic is about calculating with expressions whose values are so-called "booleans" — that is, either true or false. Examples of such expressions are "it is sunny" (which is either true or false depending on to when and where "it" refers) , $n=0$ (which is either true or false depending on the value of n), and $n<n+1$ (which is true for all numbers n). Boolean-valued expressions are called *propositions*. *Atomic* propositions are propositions that cannot be broken down into simpler propositions. The three examples above are all atomic. A non-atomic proposition would be, for example, $m<n<p$, which can be broken down into the so-called *conjunction* of $m<n$ *and* $n<p$.

Logic is not concerned with the truth or otherwise of atomic propositions; that is the concern of the problem domain being discussed. Logic is about rules for manipulating the *logical connectives* — the operators like "and", "or", and "if" that are used to combine atomic propositions.

Calculational logic places emphasis on *equality* of propositions, in contrast to other axiomatisations of logic, which emphasise logical *implication* (if ... then ...). Equality is the most basic concept of logic —a fact first recognised by Gottfried Wilhelm Leibniz, who lived from 1646 to 1716 and who was the first to try to formulate logical reasoning— and equality of propositions is no exception. We see shortly that equality of propositions is particularly special, recognition of which considerably enhances the beauty and power of reasoning with propositions.

5.2.2 Knights and Knaves

Equality of propositions is central to solving puzzles about knights and knaves. Recall that a knight always tells the truth, and a knave always lies. If A is a native of the island, the statement "A is a knight" is either true or false, and so is a proposition. Also, the statements made by the natives are propositions. A statement like "the restaurant is to the left" is either true or false. Suppose A denotes the proposition "A is a knight", and suppose native A makes a statement S. Then, the crucial observation is that the values of these two propositions are the same. That is,

$$A=S \quad .$$

For example, if A says “the restaurant is to the left”, then

$$A = L \quad ,$$

where L denotes the truth value of the statement “the restaurant is to the left”. In words, A is a knight and the restaurant is to the left, or A is not a knight and the restaurant is not to the left.

Using this rule, if A says “I am a knight”, we deduce

$$A = A \quad .$$

This doesn’t tell us anything! A moment’s thought confirms that this is what one would expect. Both knights and knaves would claim that they are knights.

If native A is asked a yes/no question Q, the response to the question is the truth value of $A = Q$. That is, the response will be “yes” if A is a knight and the answer is really yes, or A is a knave and the answer is really no. Otherwise the response will be “no”. For example, asked the question “are you a knight” all natives will answer “yes”, as $A = A$. Asked the question “is B a knight?” A will respond “yes” if they are both the same type (i.e, $A = B$), otherwise “no”. That is, A’s response is “yes” or “no” depending on the truth or falsity of $A = B$.

Because these rules are equalities, the algebraic properties of equality play a central role in the solution of logic puzzles formulated about the island. A simple, first example is if A is asked whether B is a knight, and B is asked whether A is a knight. As discussed above, A’s response is $A = B$. Reversing the roles of A and B, B’s response is $B = A$. But, equality is symmetric; therefore, the two responses will always be the same. Note that this argument does not involve any case analysis on the four different values of A and B.

The calculational properties of equality of booleans are discussed in the next section before we return again to the knights and knaves.

5.2.3 Boolean Equality

Equality —on any domain of values— has a number of characteristic properties. First, it is *reflexive*. That is $x = x$ whatever the value (or type) of x. Second, it is *symmetric*. That is, $x = y$ is the same as $y = x$. Third, it is *transitive*. That is, if $x = y$ and $y = z$ then $x = z$. Finally, if $x = y$ and f is any function then $f.x = f.y$ (where the infix dot denotes function application). This last rule is called *substitution of equals for equals* or *Leibniz’s rule*.

Equality is a binary relation. When studying relations, reflexivity, symmetry and transitivity are properties that we look out for. Equality is also a function. It is a function with range the boolean values true and false. When we study functions, the sort of properties we look out for are associativity and symmetry. For example, addition and multiplication are both associative: for all x, y and z,

$$x+(y+z) \;=\; (x+y)+z$$

and

$$x\times(y\times z) \;=\; (x\times y)\times z \quad .$$

They are also both symmetric: for all x and y,

$$x+y \;=\; y+x$$

and

$$x\times y \;=\; y\times x \quad .$$

Symmetry of equality, viewed as a function, is just the same as symmetry of equality, viewed as a relation. But, what about associativity of equality? Is equality an associative operator?

The answer is that, in all but one case, the question doesn't make sense. Associativity of a binary function only makes sense if the domains of its two arguments and the range of its result are all the same. The expression $(p=q)=r$ just doesn't make sense when p, q and r are numbers, or characters, or sequences, etc. The one exception is equality of boolean values. When p, q and r are all booleans it makes sense to compare the boolean $p=q$ with r for equality. That is, $(p=q)=r$ is a meaningful boolean value. Similarly, so too is $p=(q=r)$. It also makes sense to compare these two values for equality. In other words, it makes sense to ask whether equality of boolean values is associative — and, perhaps surprisingly, *it is*. That is, for all booleans p, q and r,

(5.1) [**Associativity**] $\quad ((p=q)=r) \;=\; (p=(q=r)) \quad .$

You should check this property by constructing truth tables for $(p=q)=r$ and for $p=(q=r)$ and comparing the entries. You should observe that the entries for which $(p=q)=r$ is true are those for which an odd number of p, q and r is true. If two of the three are true, and the third is false, $(p=q)=r$ is also false.

The associativity of equality is a very powerful property, for one because it enhances economy of expression. We will see several examples; an elementary example is the following.

The reflexivity of equality is expressed by the rule

$$(p=p)=\mathsf{true} \quad .$$

This holds for all p, whatever its type (number, boolean, string, etc.). But, for boolean p, we can apply the associativity of equality to get:

$$p = (p = \mathsf{true}) \quad .$$

This rule is most commonly used to simplify expressions by eliminating "true" from an expression of the form $p = \mathsf{true}$. We use it several times below.

5.2.4 Hidden Treasures

We can now return to the island of knights and knaves, and discover the hidden treasures. Let us consider the first problem posed in section 5.1. What can we deduce if a native says "I am a knight equals there is gold on the island"? Let A stand for "the native is a knight" and G stand for "there is gold on the island". Then the native's statement is $A = G$, and we deduce that

$$A = (A = G)$$

is true. So,

$$\begin{array}{ll}
 & \mathsf{true} \\
= & \{ \quad \text{A's statement} \quad \} \\
 & A = (A = G) \\
= & \{ \quad \text{equality of booleans is associative} \quad \} \\
 & (A = A) = G \\
= & \{ \quad (A = A) = \mathsf{true}\,, \\
 & \qquad \text{substitution of equals for equals} \quad \} \\
 & \mathsf{true} = G \\
= & \{ \quad \text{equality is symmetric} \quad \} \\
 & G = \mathsf{true} \\
= & \{ \quad G = (G = \mathsf{true}) \quad \} \\
 & G \quad .
\end{array}$$

We conclude that there is gold on the island, but it is not possible to determine whether the native is a knight or a knave.

Suppose, now, that the native is at a fork in the road, and you want to determine whether the gold can be found by following the left or right fork. You want to formulate a question such that the reply will be “yes” if the left fork should be followed, and “no” if the right fork should be followed.

As usual, we give the unknown a name. Let Q be the question to be posed. Then, as we saw earlier, the response to the question will be $A=Q$. Let L denote “the gold can be found by following the left fork.” The requirement is that L is the same as the response to the question. That is, we require that $L=(A=Q)$. But,

$$\begin{array}{ll} & L=(A=Q) \\ = & \{ \quad \text{equality is associative} \quad \} \\ & (L=A)=Q \quad . \end{array}$$

So, the question Q to be posed is $L=A$. That is, ask the question “Is the truth value of ‘the gold can be found by following the left fork’ equal to the truth value of ‘you are a knight’ ”.

Note that this analysis is valid independently of what L denotes. It might be that you want to determine whether there is a restaurant on the island, or whether there are any knaves on the island, or whatever. In general, if it is required to determine whether some proposition P is true or false, the question to be posed is $P=A$. In the case of more complex propositions P, the question may be simplified.

5.2.5 Equals for Equals

Equality is distinguished from other logical connectives by Leibniz’s rule: if two expressions are equal, one expression can be substituted for the other. Here, we consider one simple example of the use of Leibniz’s rule.

> Suppose there are three natives of the island, A, B and C, and C says “A and B are both the same type”. Formulate a question that, when posed to A, determines whether C is telling the truth.

To solve this problem, we let A, B and C denote the propositions A (respectively, B and C) is a knight. We also let Q be the unknown question.

The response we want is C. So, by the analysis in section 5.2.4, $Q=(A=C)$. But, C’s statement is $A=B$. So we know that $C=(A=B)$. Substituting equals for equals, $Q=(A=(A=B))$. But, $A=(A=B)$ simplifies to B. So, the question to be posed is “is B a knight?”. Here is this argument again, but set out as a calculation of Q, with hints showing the steps taken at each stage.

$$\begin{array}{ll} & Q \\ = & \{ \quad \text{rule for formulating questions} \quad \} \end{array}$$

$$\begin{array}{ll}
 & A = C \\
= & \{ \quad \text{from C's statement, } C = (A = B), \\
 & \quad\;\; \text{substitution of equals for equals.} \quad \} \\
 & A = (A = B) \\
= & \{ \quad \text{associativity of equality} \quad \} \\
 & (A = A) = B \\
= & \{ \quad (A = A) = \mathsf{true} \quad \} \\
 & \mathsf{true} = B \\
= & \{ \quad (\mathsf{true} = B) = B \quad \} \\
 & B \;\; .
\end{array}$$

5.3 Equivalence and Continued Equalities

Associative functions are usually denoted by infix operators[1]. The benefit in calculations is immense. If a binary operator $\oplus$ is associative (that is, $(x \oplus y) \oplus z = x \oplus (y \oplus z)$ for all x, y and z), we can write $x \oplus y \oplus z$ without fear of ambiguity. The expression becomes more compact because of the omission of parentheses. More importantly, the expression is unbiased; we may choose to simplify $x \oplus y$ or $y \oplus z$ depending on which is the most convenient. If the operator is also symmetric (that is, $x \oplus y = y \oplus x$ for all x and y) the gain is even bigger, because then, if the operator is used to combine several subexpressions, we can choose to simplify $u \oplus w$ for any pair of subexpressions u and w.

Infix notation is also often used for binary relations. We write, for example, $0 \leq m \leq n$. Here, the operators are being used *conjunctionally*: the meaning is $0 \leq m$ *and* $m \leq n$. In this way, the formula is more compact (since m is not written twice). More importantly, we are guided to the inference that $0 \leq n$. The algebraic property that is being hidden here is the transitivity of the at-most relation. If the relation between m and n is $m < n$ rather than $m \leq n$ and we write $0 \leq m < n$, we may infer that $0 < n$. Here, the inference is more complex since there are two relations involved. But, it is an inference that is so fundamental that the notation is designed to facilitate its recognition.

In the case of equality of boolean values, we have a dilemma. Do we understand

[1]An *infix operator* is a symbol used to denote a function of two arguments that is written between the two arguments. The symbols " $+$ " and " $\times$ " are both infix operators, denoting addition and multiplication, respectively.

equality as a relation and read a continued expression of the form

$$x = y = z$$

as asserting the equality of all of x, y and z? Or do we read it "associatively" as

$$(x = y) = z \quad ,$$

or, equally, as

$$x = (y = z) \quad ,$$

in just the same way as we would read $x+y+z$? The two readings are, unfortunately, not the same (for example $\mathsf{true} = \mathsf{false} = \mathsf{false}$ is false according to the first reading but true according to the second and third readings). There are advantages in both readings, and it is a major drawback to have to choose one in favour of the other.

It would be very confusing and, indeed, dangerous to read $x = y = z$ in any other way than $x = y$ and $y = z$; otherwise, the meaning of a sequence of expressions separated by equality symbols would depend on the type of the expressions. Also, the conjunctional reading (for other types) is so universally accepted —for good reasons— that it would be quite unacceptable to try to impose a different convention.

The solution to this dilemma is to use two different symbols to denote equality of boolean values — the symbol "$=$" when the transitivity of the equality relation is to be emphasised and the symbol "$\equiv$" when its associativity is to be exploited. Accordingly, we write both $p = q$ and $p \equiv q$. When p and q are expressions denoting boolean values, these both mean the same. But a continued expression

$$p \equiv q \equiv r \quad ,$$

comprising more than two boolean expressions connected by the "$\equiv$" symbol, is to be evaluated *associatively* —i.e. as $(p \equiv q) \equiv r$ or $p \equiv (q \equiv r)$, whichever is the most convenient— whereas a continued expression

$$p = q = r$$

is to be evaluated *conjunctionally* —i.e as $p = q$ *and* $q = r$— . More generally, a continued *equality* of the form

$$p_1 = p_2 = \ldots = p_n$$

means that all of p_1, p_2 , ... , p_n are equal, whilst a continued *equivalence* of the form

$$p_1 \equiv p_2 \equiv \ldots \equiv p_n$$

has the meaning given by fully parenthesising the expression (in any way whatsover, since the outcome is not affected) and then evaluating the expression as indicated by the chosen parenthesisation.

Moreover, we recommend that the "$\equiv$" symbol is pronounced as "equivales"; being an unfamiliar word, its use will help to avoid misunderstanding.

Shortly, we introduce a number of laws governing boolean equality. They invariably involve a contninued equivalence. A first example is its reflexivity.

(5.2) **[Reflexivity]** $\mathsf{true} \equiv p \equiv p$.

5.3.1 Examples of the Associativity of Equivalence

This section contains a couple of beautiful examples illustrating the effectiveness of the associativity of equivalence.

Even and Odd Numbers The first example is the following property of the predicate even on numbers. (A number is even exactly when it is a multiple of two.)

$$m{+}n \text{ is even} \;\equiv\; m \text{ is even} \;\equiv\; n \text{ is even} \;\; .$$

It will help if we refer to whether or not a number is even or odd as the *parity* of the number. Then, if we parenthesise the statement as

$$m{+}n \text{ is even} \;\equiv\; (m \text{ is even} \;\equiv\; n \text{ is even}) \;\; ,$$

it states that the number $m{+}n$ is even exactly when the parities of m and n are the same. Parenthesising it as

$$(m{+}n \text{ is even} \;\equiv\; m \text{ is even}) \;\equiv\; n \text{ is even} \;\; ,$$

it states that the operation of adding a number n to a number m does not change the parity of m exactly when n is even.

Another way of reading the statement is to use the fact that, in general, the equivalence $p \equiv q \equiv r$ is true exactly when an odd number of p , q and r is true. So the property captures four different cases:

$$\begin{array}{ll}
 & ((m{+}n \text{ is even}) \;\text{ and }\; (m \text{ is even}) \;\text{ and }\; (n \text{ is even})) \\
\text{or} & ((m{+}n \text{ is odd}) \;\text{ and }\; (m \text{ is odd}) \;\text{ and }\; (n \text{ is even})) \\
\text{or} & ((m{+}n \text{ is odd}) \;\text{ and }\; (m \text{ is even}) \;\text{ and }\; (n \text{ is odd})) \\
\text{or} & ((m{+}n \text{ is even}) \;\text{ and }\; (m \text{ is odd}) \;\text{ and }\; (n \text{ is odd})) \;\; .
\end{array}$$

The beauty of this example lies in the avoidance of case analysis. There are four distinct combinations of the two booleans "m is even" and "n is even". Using the associativity of equivalence the value of "$m+n$ is even" is expressed in one simple formula, without any repetition of the component expressions, rather than as a list of different cases. Avoidance of case analysis is vital to effective reasoning.

Sign of Non-Zero Numbers The *sign* of a number says whether or not the number is positive. For non-zero numbers x and y, the product $x{\times}y$ is positive if the signs of x and y are equal. If the signs of x and y are different, the product $x{\times}y$ is negative.

Assuming that x and y are non-zero, this rule is expressed as

$$x{\times}y \text{ is positive } \equiv x \text{ is positive } \equiv y \text{ is positive } .$$

Just as for the predicate even, this one statement neatly captures a number of different cases, even though no case analysis is involved. Indeed, our justification of the rule is the statement

$$x{\times}y \text{ is positive } \equiv (x \text{ is positive } \equiv y \text{ is positive}) .$$

The other parenthesisation —which states that the sign of x is unchanged when it is multiplied by y exactly when y is positive— is obtained "for free" from the associativity of boolean equality.

5.3.2 On Natural Language

Many mathematicians and logicians are not aware that equality of booleans is associative; those that do are often unaware or dismissive of how effective its use can be. At the present time, there is considerable resistance to a shift in focus from implication to equality. Most courses on logic introduce boolean equality as "if and only if". It is akin to introducing equality of numbers by first introducing the at-most ($\leq$) and at-least ($\geq$) relations, and then defining an "at-most and at-least" operator.

The most probable explanation lies in the fact that many logicians view the purpose of logic as formalising "natural" or "intuitive" reasoning, and our "natural" tendency is not to reason in terms of equalities, but in causal terms. ("If it is raining, I will take my umbrella.") The equality symbol was first introduced into mathematics by Robert Recorde in 1557, which, in the history of mathematics, is quite recent; were equality "natural" it would have been introduced much earlier. Natural language has no counterpart to a continued equivalence. To take a concrete example, the continued equivalence "a blind man can see through two eyes equivales a blind man can see through one eye equivales a blind man can see through no eyes" may seem very odd, if not nonsensical, even though it is actually true!

This fact should not be a deterrent to the use of continued equivalence. At one time (admittedly, a very long time ago) there was probably similar resistance to the introduction of continued additions and multiplications. The evidence is still present in the language we use today. For example, the most common way to express time is in words: like "quarter to ten" or "ten past eleven". Calculational requirements (eg. wanting to determine how long is it before the train is due to arrive) have influenced natural language so that, nowadays, people sometimes say, for example, 9:45 or 11:10 in everyday speech. But, we still don't find it acceptable to say 10:70! Yet, this is what we actually use when we want to calculate the time difference between 9.45 and 11:10. In fact, several laws of arithmetic, including associativity of addition, are fundamental to the calculation. Changes in natural language have occurred, and will continue to occur, as a result of progress in mathematics, but will always lag a long way behind. The language of mathematics has developed in order to overcome the limitations of natural language. The goal is not to mimic "natural" reasoning, but to provide a more effective alternative.

5.4 Negation

Consider the following knights-and-knaves problem. There are two natives, A and B. Native A says, "B is a knight equals I am not a knight". What can you determine about A and B?

This problem involves a so-called *negation*: the use of "not". Negation is a unary operator (meaning that it is a function with exactly one argument) mapping a boolean to a boolean, and is denoted by the symbol "$\neg$", written as a prefix to its argument. If p is a boolean expression, "$\neg p$" is pronounced "*not* p".

Using the general rule that, if A makes a statement S, we know that $A \equiv S$, we get, for this problem:

$$A \;\equiv\; B \;\equiv\; \neg A \quad .$$

(We switch from "$=$" to "$\equiv$" here in order to exploit associativity.) The goal is to simplify this expression.

In order to tackle this problem, it is necessary to begin by formulating calculational rules for negation. For arbitrary proposition p, the law governing $\neg p$ is:

(5.3) **[Negation]** $\neg p \;\equiv\; p \;\equiv\; \mathsf{false}$.

Reading this as

$$\neg p \;=\; (p \equiv \mathsf{false}) \quad ,$$

it functions as a definition of negation. Reading it the other way:

$$(\neg p \equiv p) \;=\; \mathsf{false}$$

it provides a way of simplifying propositional expressions. In addition, the symmetry of equivalence means that we can rearrange the terms in a continued equivalence in any order we like. So, we also get the property:

$$p \;=\; (\neg p \equiv \mathsf{false}) \quad .$$

Returning to the knights-and-knaves problem, we are given that:

$$A \equiv B \equiv \neg A \quad .$$

This simplifies to $\neg B$ as follows:

$$\begin{array}{ll} & A \equiv B \equiv \neg A \\ = & \{ \quad \text{rearranging terms} \quad \} \\ & \neg A \equiv A \equiv B \\ = & \{ \quad \text{law (5.3) with } p := A \quad \} \\ & \mathsf{false} \equiv B \\ = & \{ \quad \text{law (5.3) with } p := B \text{ and rearranging} \quad \} \\ & \neg B \quad . \end{array}$$

So, B is a knave, but A could be a knight or a knave. Note how (5.3) is used in two different ways.

The law (5.3), in conjunction with the symmetry and associativity of equivalence, provides a way of simplifying continued equivalences in which one or more terms are repeated and/or negated. Suppose, for example, we want to simplify

$$\neg p \equiv p \equiv q \equiv \neg p \equiv r \equiv \neg q \quad .$$

We begin by rearranging all the terms so that repeated occurrences of " p " and " q " are grouped together. Thus we get

$$\neg p \equiv \neg p \equiv p \equiv q \equiv \neg q \equiv r \quad .$$

Now we can use (5.2) and (5.3) to reduce the number of occurrences of " p " and " q " to at most one (possibly negated). In this particular example we obtain

$$\mathsf{true} \equiv p \equiv \mathsf{false} \equiv r \quad .$$

Finally, we use (5.2) and (5.3) again. The result is that the original formula is simplified to

$$\neg p \equiv r \quad .$$

Just as before, this process can be compared with the simplification of an arithmetic expression involving continued addition, where now negative terms may also appear. The expression

$$p+(-p)+q+(-p)+r+q+(-q)+r+p$$

is simplified to

$$q+2r$$

by counting all the occurrences of p, q and r, an occurrence of $-p$ cancelling out an occurrence of p. Again, the details are different but the process is essentially identical.

The two laws (5.2) and (5.3) are all that is needed to define the way that negation interacts with equivalence; using these two laws we can derive several other laws. A simple example of how these two laws are combined is a proof that $\neg\mathsf{false} = \mathsf{true}$:

$$\begin{array}{ll} & \neg\mathsf{false} \\ = & \{ \quad \text{law } \neg p \equiv p \equiv \mathsf{false} \text{ with } p := \mathsf{false} \quad \} \\ & \mathsf{false} \equiv \mathsf{false} \\ = & \{ \quad \text{law } \mathsf{true} \equiv p \equiv p \text{ with } p := \mathsf{false} \quad \} \\ & \mathsf{true} \quad . \end{array}$$

5.5 Contraposition

A rule that should now be obvious, but which is surprisingly useful, is the rule we call *contraposition*[2].

(5.4) **[Contraposition]** $p \equiv q \equiv \neg p \equiv \neg q \quad .$

The name refers to the use of the rule in the form $(p \equiv q) = (\neg p \equiv \neg q)$.

We used the rule of contraposition implicitly in the river-crossing problems. (See chapter 3.) Recall that each problem involves getting a group of people from one side of a river to another, using one boat. If we let n denote the number of crossings, and l

[2]Other authors use the name "contraposition" for a less general rule combining negation with implication.

denote the boolean "the boat is on the left side of the river", a crossing of the river is modelled by the assignment:

$$n\,,l \;:=\; n{+}1\,,\neg l \;\;.$$

In words, the number of crossings increases by one, and the boat changes side. The rule of contraposition tells us that

$$even.n \equiv l$$

is invariant under this assignment. This is because

$$\begin{array}{ll}
 & (even.n \equiv l)[n\,,l \;:=\; n{+}1\,,\neg l] \\
= & \{ \quad \text{rule of substitution} \quad \} \\
 & even.(n{+}1) \equiv \neg l \\
= & \{ \quad even.(n{+}1) \equiv \neg(even.n) \quad \} \\
 & \neg(even.n) \equiv \neg l \\
= & \{ \quad \text{contraposition} \quad \} \\
 & even.n \equiv l \;\;.
\end{array}$$

We are given that, initially, the boat is on the left side. Since zero is an even number, we conclude that $even.n \equiv l$ is invariantly true. In words, the boat is on the left side equivales the number of crossings is even.

Another example is the following. Suppose it is required to move a square armchair sideways by a distance equal to its own width. (See figure 5.1.) However, the chair is so heavy that it can only be moved by rotating it through $90°$ around one of its four corners. Is it possible to move the chair as desired? If so, how? If not, why not?

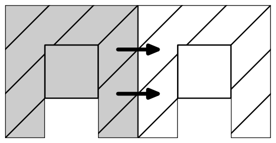


Figure 5.1: Moving a heavy armchair.

The answer is that it is impossible. Suppose the armchair is initially positioned along a north-south axis. Suppose, also, that the floor is painted alternately with black and white squares, like a chess board, with each of the squares being the same size as the armchair. (See Figure 5.2.) Suppose the armchair is initially on a black square. The requirement is to move the armchair from a north-south position on a black square to a north-south position on a white square.

Now, let boolean col represent the colour of the square that the armchair is on (say, $true$ for black and $false$ for white), and dir represent the direction that the armchair is facing (say, $true$ for north-south and $false$ for east-west). Then, rotating the armchair about any corner is represented by the assignment:

$$col, dir \;:=\; \neg col, \neg dir$$

The rule of contraposition states that an invariant of this assignment is

$$col \equiv dir \quad .$$

So, the value of this expression will remain equal to its initial value, no matter how many times the armchair is rotated. But, moving the armchair sideways one square changes the colour but does not change the direction. That is, it changes the value of $col \equiv dir$, and is impossible to achieve by continually rotating the armchair as prescribed.

In words, an invariant of rotating the armchair through $90°$ around a corner point is

$$\text{the chair is on a black square} \;\equiv\; \text{the chair is facing north-south}$$

which is false when the chair is on a white square and facing north-south.

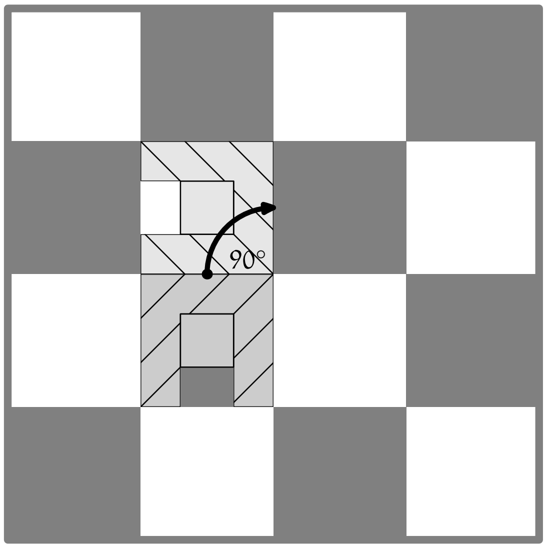


Figure 5.2: Invariant when moving a heavy armchair.

Exercise 5.5 (Knight's Move) In the game of chess, a knight's move is two places up or down and one place left or right, or, vice versa, two places left or right and one place up or down. A chessboard is an 8×8 grid of squares.

Show that it is impossible to move a knight from the bottom-left corner of a chessboard to the top-right corner in such a way that every square on the board is visited exactly once.

Hint: How many moves have to be made? Model a move in terms of the effect on the number of moves and the colour of the square on which the knight is standing; identify a relation between the two that is invariant under a move.

□

5.6 Handshake Problems

Logical properties of negation are fundamental to solving so-called *handshake problems*.

The simplest example of a handshake problem is this: Suppose that at a party, some people shake hands and some don't. Suppose each person counts the number of times they shake hands. Show that at least two people have the same count.

Crucial to how we solve this problem are the properties of shaking hands. These are that it is a *binary relation*; it is *symmetric*, and it is *anti-reflexive*. It being a "binary relation" on people, means that, for any two people —Jack and Jill, say— Jack shakes hands with Jill is either true or false. (In general, a *relation* is any boolean-valued function. Binary means that it is a function of two arguments.) It being a "symmetric" relation means that, for any two people —Jack and Jill, say— Jack shakes hands with Jill equivales Jill shakes hands with Jack. Finally, it being "anti-reflexive" means that no-one shakes hands with themselves.

We are required to show that (at least) two people shake hands the same number of times. Let us explore the consequences of the properties of the shake-hands relation with respect to the number of times that each person shakes hands.

Suppose there are n people. Then, everyone shakes hands with at most n people. However, the anti-reflexivity property is that noone shakes hands with themselves. We conclude that everyone shakes hands with between 0 and $n-1$ people.

There are n numbers in the range 0 to $n-1$. The negation of "two people shake hands the same number of times" is "everyone shake hands a distinct number of times". In particular, someone shakes hands 0 times and someone shakes hands $n-1$ times. The symmetry of the shake-hands relation makes this impossible.

Suppose we abbreviate "shake hands" to S, and suppose we use x and y to refer to people. In this way, xSy is read as "person x shakes hands with person y", or just x shakes hands with y. Then the symmetry of "shakes hands" gives us the rule, for all x and y,

$$xSy \equiv ySx \quad .$$

The contrapositive of this rule is that, for all x and y,

$$\neg(xSy) \equiv \neg(ySx) \quad .$$

In words, x doesn't shake hands with y equivales y doesn't shake hands with x. Now, suppose person a shakes hands with noone and person b shakes hands with everyone. Then, in particular, a does not shake hands with b, i.e. $\neg(aSb)$, and b shakes hands with a, i.e. bSa. But then, substituting equals for equals, we have both $\neg(aSb)$ and aSb, which is false.

The assumption that everyone shakes hands with a distinct number of people has led to a contradiction, and so we conclude that two people must shake hands the same number of times.

Note carefully how the symmetry and anti-reflexivity of the shakes-hands relation are crucial. Were we to consider a similar problem involving a different relation, the outcome might be different. For example, if we replace "shake hands" by some other form of greeting like "bows or curtsies", which is not symmetric, the property need not hold[3]. (Suppose there are two people, and one bows to the other, but the greeting is not returned.) However, if "shake hands" is replaced by "rub noses", the property does hold. Like "shake hands", "rub noses" is a symmetric and anti-reflexive relation.

Exercise 5.6 Here is another handshaking problem. It's a bit more difficult to solve, but the essence of the problem remains the same: "shake hands" is a symmetric relation, as is "don't shake hands".

Suppose a number of couples (husband and wife) attend a party. Some people shake hands, others do not. Husband and wife never shake hands. One person, the "host", asks everyone else how many times they have shaken hands, and gets a different answer every time. How many times did the host and the host's partner shake hands?

□

5.7 Inequivalence

In the knights-and-knaves problem mentioned at the beginning of section 5.4, A might have said "B is different from myself". This statement is formulated as $B \neq A$, or $\neg(B = A)$ This is, in fact, the same as saying "B is a knight equals I am not a knight", as the following calculation shows. Note that we switch from "$=$" to "$\equiv$" once again, in order to exploit associativity.

[3]At the time of writing, anyone meeting the British Queen is required to bow or curtsey, whereas the Queen never does. The relation is not symmetric.

$$
\begin{array}{ll}
 & \neg(B \equiv A) \\
= & \{ \quad \text{the law } \neg p \equiv p \equiv \mathsf{false} \text{ with } p := (B \equiv A) \quad \} \\
 & B \equiv A \equiv \mathsf{false} \\
= & \{ \quad \text{the law } \neg p \equiv p \equiv \mathsf{false} \text{ with } p := A \quad \} \\
 & B \equiv \neg A \quad .
\end{array}
$$

We have thus proved, for all propositions p and q,

(5.7) **[Inequivalence]** $\neg(p \equiv q) \equiv p \equiv \neg q$.

Note how associativity of equivalence has been used silently in this calculation. Note also how associativity of equivalence in the summary of the calculation gives us two properties for the price of one. The first is the one proved directly:

$$\neg(p \equiv q) = (p \equiv \neg q) \quad ,$$

the second comes free with associativity:

$$(\neg(p \equiv q) \equiv p) = \neg q \quad .$$

The proposition $\neg(p \equiv q)$ is usually written $p \not\equiv q$. The operator is called *inequivalence* (or *exclusive-or*, abbreviated *xor*). Inequivalence is also associative:

$$
\begin{array}{ll}
 & (p \not\equiv q) \not\equiv r \\
= & \{ \quad \text{definition of “}\not\equiv\text{”, applied twice} \quad \} \\
 & \neg(\neg(p \equiv q) \equiv r) \\
= & \{ \quad \text{(5.7), with } p,q := \neg(p \equiv q)\,,r \quad \} \\
 & \neg(p \equiv q) \equiv \neg r \\
= & \{ \quad \text{contraposition (5.4), with } p,q := p \equiv q\,,r \quad \} \\
 & p \equiv q \equiv r \\
= & \{ \quad \text{contraposition (5.4), with } p,q := p\,, q \equiv r \quad \} \\
 & \neg p \equiv \neg(q \equiv r) \\
= & \{ \quad \text{(5.7), with } p,q := p\,, q \equiv r \quad \} \\
 & \neg(p \equiv \neg(q \equiv r)) \\
= & \{ \quad \text{definition of “}\not\equiv\text{”, applied twice} \quad \} \\
 & p \not\equiv (q \not\equiv r) \quad .
\end{array}
$$

As a result, we can write the *continued inequivalence* $p \not\equiv q \not\equiv r$ without fear of ambiguity[4]. Note that, as a byproduct, we have shown that $p \not\equiv q \not\equiv r$ and $p \equiv q \equiv r$ are equal.

As a final worked example, we show that inequivalence associates with equivalence:

$$
\begin{array}{ll}
 & (p \not\equiv q) \equiv r \\
= & \quad \{ \quad \text{expanding the definition of } p \not\equiv q \quad \} \\
 & \neg(p \equiv q) \equiv r \\
= & \quad \{ \quad \neg(p \equiv q) \equiv p \equiv \neg q \quad \} \\
 & p \equiv \neg q \equiv r \\
= & \quad \{ \quad \text{using symmetry of equivalence, the law (5.7)} \\
 & \qquad\quad \text{is applied in the form } \neg(p \equiv q) \equiv \neg q \equiv p \\
 & \qquad\quad \text{with } p,q := q,r \quad \} \\
 & p \equiv \neg(q \equiv r) \\
= & \quad \{ \quad \text{definition of } q \not\equiv r \quad \} \\
 & p \equiv (q \not\equiv r) \ .
\end{array}
$$

Exercise 5.8 Simplify the following. (Note that in each case it does not matter in which order you evaluate the subexpressions. Also, rearranging the variables and/or constants doesn't make any difference.)

(a) $\mathsf{false} \not\equiv \mathsf{false} \not\equiv \mathsf{false}$

(b) $\mathsf{true} \not\equiv \mathsf{true} \not\equiv \mathsf{true} \not\equiv \mathsf{true}$

(c) $\mathsf{false} \not\equiv \mathsf{true} \not\equiv \mathsf{false} \not\equiv \mathsf{true}$

(d) $p \equiv p \equiv \neg p \equiv p \equiv \neg p$

(e) $p \not\equiv q \equiv q \equiv p$

(f) $p \not\equiv q \equiv r \equiv p$

(g) $p \equiv p \not\equiv \neg p \not\equiv p \equiv \neg p$

(h) $p \equiv p \not\equiv \neg p \not\equiv p \equiv \neg p \not\equiv \neg p$

[4]This is to be read associatively, and should not be confused with $p \neq q \neq r$, which some authors occasionally write. Inequality is not transitive, so such expressions should be avoided.

□

Exercise 5.9 Prove that $\neg\mathsf{true} = \mathsf{false}$

□

Exercise 5.10 (Double Negation) Prove the rule of double negation

$$\neg\neg p = p \quad .$$

□

Exercise 5.11 (Encryption) The fact that inequivalence is associative, that is

$$(p \not\equiv (q \not\equiv r)) \equiv ((p \not\equiv q) \not\equiv r) \quad ,$$

is used to encrypt data. To encrypt a single bit b of data, a key a is chosen and the encrypted form of b that is transmitted is $a \not\equiv b$. The receiver decrypts the received bit, c, using the same operation[5]. That is, the receiver uses the same key a to compute $a \not\equiv c$. Show that, if bit b is encrypted and then decrypted in this way, the result is b independently of the key a.

□

Exercise 5.12 On the island of knights and knaves, you encounter two natives, A and B. What question should you ask A to determine whether A and B are different types?

□

5.8 Summary

In this chapter, we have used simple logic puzzles to introduce logical equivalence — the equality of boolean values— , the most fundamental logical operator. Equivalence has the remarkable property of being associative, in addition to the standard properties of equality. Exploitation of the associativity of equivalence eliminates the tedious and ineffective case analysis that is often seen in solutions to logic puzzles.

[5]This operation is usually called "exclusive-or" in texts on data encryption; it is not commonly known that exclusive-or and inequivalence are the same. Inequivalence can be replaced by equivalence in the encryption and decryption process. But, very few scientists and engineers are aware of the algebraic properties of equivalence, and this possibility is never exploited!

The associativity of equivalence can be difficult to get used to, particularly if one tries to express its properties in natural language. However, this should not be used as an excuse for ignoring it. The painful, centuries-long process of accepting zero as a number, and introducing the symbol "0" to denote it, provides ample evidence that the adherence to "natural" modes of reasoning is a major impediment to effective reasoning. The purpose of a calculus is not to mimic "natural" or "intuitive" reasoning, but to provide a more powerful alternative.

The fact that equality of boolean values is associative has been known since at least the 1920's , having been mentioned by Alfred Tarski in his PhD thesis, where its discovery is attributed to J. Lukasiewicz. (See the paper "On the primitive term of logistic" [Tar56]; Tarski is a famous logician.) Nevertheless, its usefulness was never recognised until brought to the fore by E.W. Dijkstra in his work on program semantics and mathematical method. (See e.g. [DS90].)

The origin of the logic puzzles is Raymond Smullyan's book "What Is The Name Of This Book?" [Smu78]. This is an entertaining book which leads on from simple logic puzzles to a discussion of the logical paradoxes and Gödel's undecidability theorem. But Smullyan's proofs invariably involve detailed case analyses. The exploitation of the associativity of equivalence in the solution of such puzzles is due to Gerard Wiltink [Wil87]. For a complete account of calculational logic, which includes discussion of conjunction ("and"), disjunction ("or"), follows-from ("if") and implication ("only if"), see [Bac03] or [GS93].

Chapter 6

Induction

"Induction" is the name given to a problem-solving technique based on using the solution to small instances of a problem to solve larger instances of the problem.

The idea is that we somehow measure the "size" of instances of a problem. For example, the problem might involve a pile of matchsticks, where the number of matches is a parameter; an instance of the problem is then a particular pile of matches, and its size is the number of matches in the pile. A requirement is that the size is a non-negative, whole number — thus 0, 1, 2, 3, etc. We use the term *natural number* for a non-negative, whole number[1]. Usually, how the size of an instance of a problem is measured is quite obvious from the problem description.

Having decided how to measure size, we then solve the problem in two steps. First, we consider problems of size 0. This is called the *basis* of the induction. Almost invariably, such problems are very easy to solve. (They are often dismissed as "trivial".) Second, we show how to solve, for an arbitrary natural number n, a problem of size $n+1$, given a solution to a problem of size n. This is called the *induction step*.

By this process, we can solve problems of size 0. We also know how to solve problems of size 1; we apply the induction step to construct a solution to problems of size 1 from the known solution to problems of size 0. Then, we know how to solve problems of size 2; we apply the induction step again to construct a solution to problems of size 2 from the known solution to problems of size 1. And so it goes on. We can now solve problems of size 3, then problems of size 4, etc.

6.1 Example Problems

All the following problems can be solved by induction. In the first, the size is the number of lines. In the second and third problems, it is explictly given by the parameter n, and

[1] ***Warning***: Mathematicians often exclude the number 0 from the natural numbers. There are, however, very good reasons why 0 should always be included, making a break with tradition imperative.

in the fourth, it is the number of disks. In each case, the basis should be easy. You then have to solve the induction step. We discuss each problem in turn in coming sections.

1. **Cutting the Plane**

   A number of straight lines are drawn across a sheet of paper, each line extending all the way from from one border to another. See fig. 6.1. In this way, the paper is divided into a number of regions. Show that it is possible to colour each region black or white so that no two adjacent regions have the same colour (that is, so that the two regions on opposite sides of any line segment have different colours).

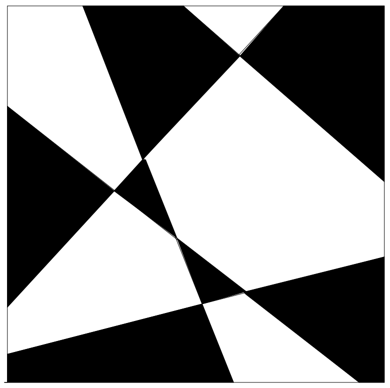


Figure 6.1: Black and White Colouring.

2. **Triominoes**

   A square piece of paper is divided into a grid of size $2^n \times 2^n$, where n is a natural number[2]. The individual squares are called *grid squares*. One grid square is covered, and the others are left uncovered. A triomino is an L-shape made of three grid squares. Figure 6.2 shows, on the left, an 8×8 grid with one square covered. On the right is a triomino.

   Show that it is possible to cover the remaining squares with (non-overlapping) triominoes. (Fig. 6.3 shows a solution in one case.)

   NB: The case $n=0$ should be included in your solution.

3. **Trapeziums**

   An equilateral triangle, with side of length 2^n for some natural number n, is made of smaller equilateral triangles. The topmost equilateral triangle is covered. A bucket-shaped trapezium is made from three equilateral triangles. See fig. 6.4.

[2]Recall that the natural numbers are the numbers 0, 1, 2, etc.

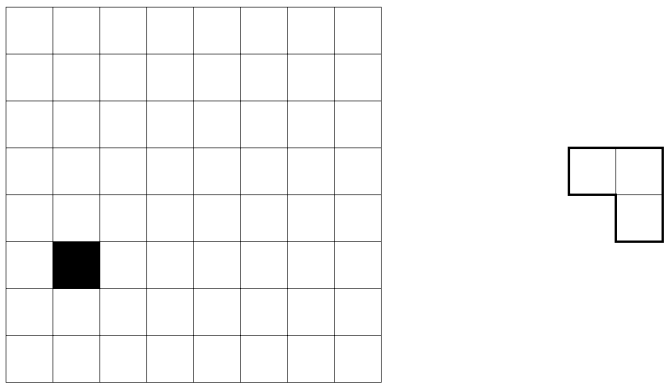


Figure 6.2: Triomino Problem.

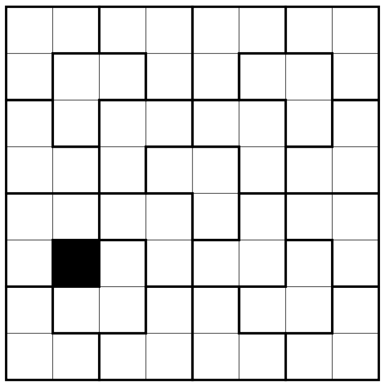


Figure 6.3: Triomino Problem. Solution to fig. 6.2.

Show that it is possible to cover the remaining triangles with (non-overlapping) trapeziums. See fig. 6.5 for the solution in the case that n is 2.

NB. Include the case $n=0$ in your solution.

4. **Towers of Hanoi**

   The Towers of Hanoi problem comes from a puzzle marketed in 1883 by the French mathematician Édouard Lucas, under the pseudonym M. Claus.

   The puzzle is based on a legend according to which there is a temple in Bramah where there are three giant poles fixed in the ground. On the first of these poles, at the time of the world's creation, God placed sixty four golden disks, each of different size, in decreasing order of size. (See fig. 6.6.) The Brahmin monks were given the task of moving the disks, one per day, from one pole to another according to the rule that no disk may ever be above a smaller disk. The monks' task will be complete when they have succeeded in moving all the disks from the first of the

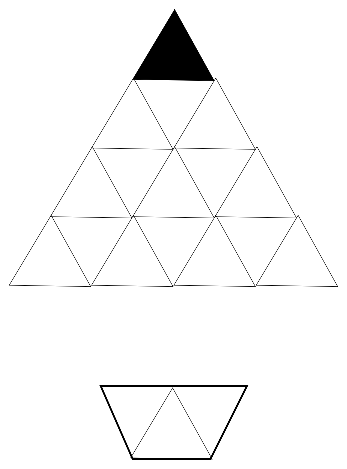


Figure 6.4: A Pyramid of Equilateral Triangles.

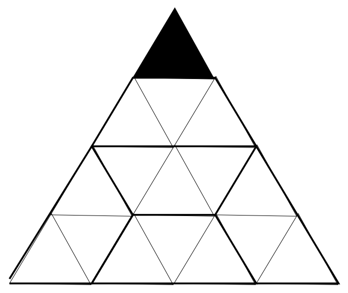


Figure 6.5: Solution to fig. 6.4.

poles to the second and, on the day that they complete their task, the world will come to an end!

Construct an inductive solution to this problem. The base case is when there are no disks to be moved.

(We see later that the inductive solution is certainly not the one that the Brahmin monks use. However, it does provide the basis for constructing a so-called *iterative* solution to the problem.)

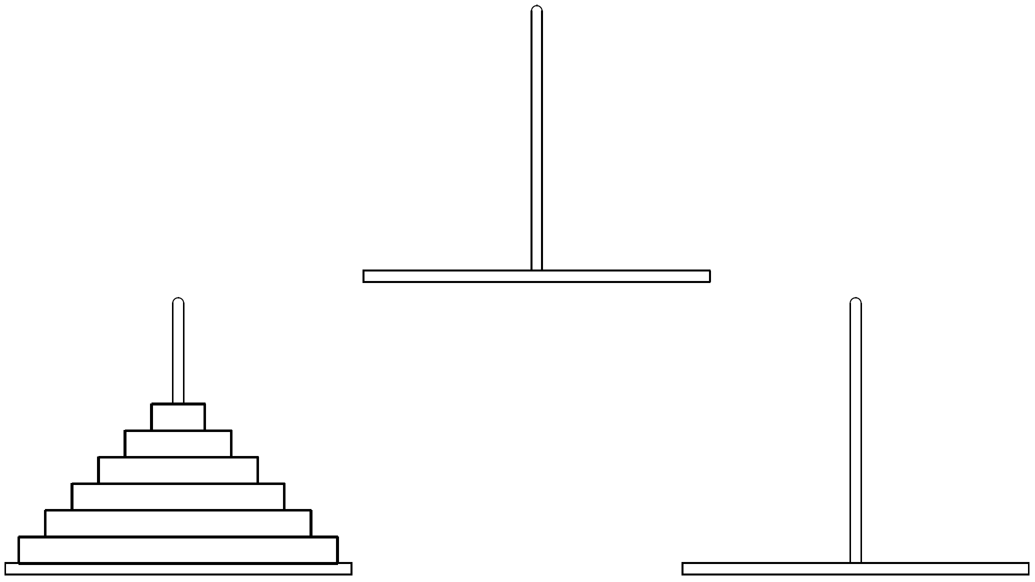


Figure 6.6: Towers of Hanoi Problem

6.2 Cutting The Plane

Recall the statement of the problem:

> A number of straight lines are drawn across a sheet of paper, each line extending all the way from from one border to another. See fig. 6.1. In this way, the paper is divided into a number of regions. Show that it is possible to colour each region black or white so that no two adjacent regions have the same colour (that is, so that the two regions on opposite sides of any line segment have different colours).

For this problem, the number of lines is an obvious measure of the "size" of the problem. The goal is, thus, to solve the problem "by induction on the number of lines". This means that we have to show how to solve the problem when there are zero lines —this is the "basis" of the induction— and we have to show how to solve the problem when there are $n+1$ lines, where n is an arbitrary number, assuming that we can solve the problem when there are n lines —this is the induction step— .

For brevity, we call a colouring of the regions with the property that no two adjacent regions have the same colour a *satisfactory colouring*.

The case where there are zero lines is easy. The sheet of paper is divided into one region, and this can be coloured black or white, either colouring meeting the conditions of a solution (because there is no pair of adjacent regions).

For the induction step, we assume that a number of lines have been drawn on the sheet of paper, and the different regions have been coloured black or white so that no two adjacent regions have the same colour. This assumption is called the *induction hypothesis*. We now suppose that an additional line is drawn on the paper. This will divide some of the existing regions into two; such pairs of regions will have the same colour, and so the existing colouring is not satisfactory. Fig. 6.7 is an example. The plane has been divided into twelve regions by five lines, and the regions coloured black and white, as required. An additional line, shown in red for clarity, has been added. This has had the effect of dividing four of the regions into two, thus increasing the number of regions by four. On either side of the red line, the regions have the same colour. Elsewhere, adjacent regions have different colours. The task is to show how to modify the colouring so that it does indeed become a satisfactory solution.

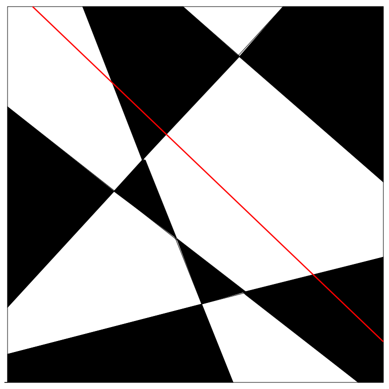


Figure 6.7: Cutting the Plane. Additional line shown in red.

The key to a solution is to note that inverting the colours of any satisfactory colouring (that is, changing a black region to white, and vice-versa) also gives a satisfactory colouring. Now, the additional line divides the sheet of paper into two regions. Let us call these regions the *left* and *right* regions. (By this choice of names, we do not imply that the additional line must be from top to bottom of the page. It is just a convenient, easily remembered, way of naming the regions.) Note that the assumed colouring is a satisfactory colouring of the left region and of the right region. In order to guarantee that, either side of the additional line, all regions have opposite colour, choose, say, the left region, and invert all the colours in that region. This gives a satisfactory colouring of the left region (because inverting the colours of a satisfactory colouring gives a satisfactory colouring). It also gives a satisfactory colouring of the right region (because the colouring hasn't changed, and was satisfactory already). Also, the colouring of adjacent regions at the boundary of the left and right regions is satisfactory, because they have changed from being the same to being different.

Fig. 6.8 shows the effect on our example. Blue has been used instead of black in order to make the inversion of the colours more evident.

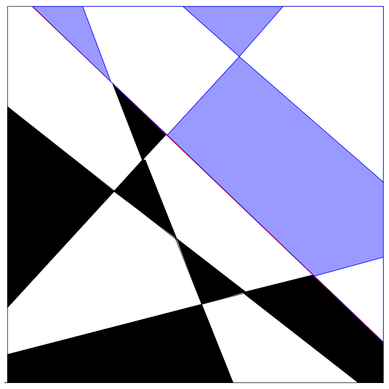


Figure 6.8: Cutting the Plane. The colours are inverted to one side of the additional line (black is shown as blue to make clear which colours have changed).

This completes the induction step. In order to apply the construction to an instance of the problem with, say, seven lines, we begin by colouring the whole sheet of paper. Then the lines are added one-by-one. Each time a line is added, the existing colouring is modified as prescribed in the induction step, until all seven lines have been added.

The algorithm is non-deterministic in several ways. The initial colouring of the sheet of paper (black or white) is unspecified. The ordering of the lines (which to add first, which to add next, etc.) is also unspecified. Finally, which region is chosen as the "left" region, and which the "right" region is unspecified. This means that the final colouring may be achieved in lots of different ways. But that doesn't matter. The final colouring is guaranteed to be "satisfactory", as required in the problem specification.

Exercise 6.1 Check your understanding by considering variations on the problem.

Why is it required that the lines are straight? How might this assumption be relaxed without invalidating the solution.

The problem assumes the lines are drawn on a piece of paper. Is the solution still valid if the lines are drawn on the surface of a ball, or on the surface of a doughnut?

We remarked that the algorithm for colouring the plane is non-deterministic. How many different colourings does it construct?

□

6.3 Triominoes

As a second example of an inductive construction, let us consider the grid problem posed in section 6.1. Recall the statement of the problem.

> A square piece of paper is divided into a grid of size $2^n \times 2^n$, where n is a natural number. The individual squares are called *grid squares*. One grid square is covered, and the others are left uncovered. A triomino is an L-shape made of three grid squares. Show that it is possible to cover the remaining squares with (non-overlapping) triominoes.

The obvious measure of the "size" of instances of the problem, in this case, is the number n. We solve the problem by induction on n.

The base case is when n equals 0. The grid then has size $2^0 \times 2^0$, i.e. 1×1. That is, there is exactly one square. This one square is, inevitably, the one that is covered, leaving no squares uncovered. It takes 0 triominoes to cover no squares! This, then, is how the base case is solved.

Now, suppose we consider a grid of size $2^{n+1} \times 2^{n+1}$. We make the *induction hypothesis* that it is possible to cover any grid of size $2^n \times 2^n$ with triominoes if, first, an arbitrary grid square has been covered. We have to show how to exploit this hypothesis in order to cover a grid of size $2^{n+1} \times 2^{n+1}$ of which one square has been covered.

A grid of size $2^{n+1} \times 2^{n+1}$ can be subdivided into 4 grids each of size $2^n \times 2^n$, simply by drawing horizontal and vertical dividing lines through the middle of each side. Let us call the four grids the *bottom-left*, *bottom-right*, *top-left*, and *top-right* grids. One grid square is already covered. This square will be in one of the four sub-grids. We may assume that it is in the bottom-left grid. (If not, the entire grid can be rotated about the centre so that it does become the case.)

The bottom-left grid is thus a grid of size $2^n \times 2^n$ of which one square has been covered. By the induction hypothesis, the remaining squares in the bottom-left grid can be covered with triominoes. This leaves us with the task of covering the bottom-right, top-left and top-right grids with triominoes.

None of the squares in these three grids is covered, as yet. We can apply the induction hypothesis to them if just one square in each of the three is covered. This is done by placing a triomino at the junction of the three grids, as shown in fig. 6.9.

Now, the inductive hypothesis is applied to cover the remaining squares of the bottom-right, top-left and top-right grids with triominoes. On completion of this process, the entire $2^{n+1} \times 2^{n+1}$ grid has been covered with triominoes.

Exercise 6.2 Solve the trapezium problem given in section 6.1.

□

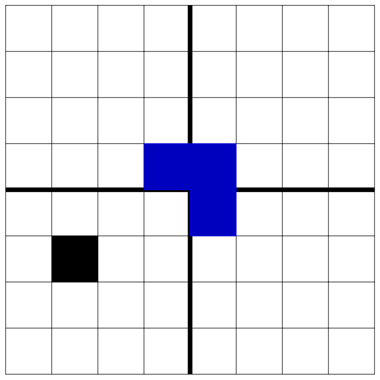


Figure 6.9: Triomino Problem. Inductive Step. The grid is divided into four sub-grids. The covered square, shown in black, identifies one sub-grid. A triomino, shown in blue, is placed at the junction of the other three grids. The induction hypothesis is then used to cover all four sub-grids with triominoes.

6.4 Looking For Patterns

In sections 6.2 and 6.3, we have seen how induction is used to solve problems of a given “size”. Technically, the process we described is called “mathematical induction”; “induction”, as it is normally understood, is more general.

“Induction”, as used in, for example, the experimental sciences, refers to a process of reasoning whereby general laws are inferred from a collection of observations. A famous example of induction is the process that led Charles Darwin to formulate his theory of evolution by natural selection, based on his observations of plant and animal life in remote parts of the world. In simple terms, induction is about *looking for patterns*.

Laws formulated by a process of induction go beyond the knowledge on which they are based, thus introducing inherently new knowledge. In the experimental sciences, however, such laws are only *probably* true; they are tested by the predictions they make, and may have to be discarded if the predictions turn out to be false. In contrast, *deduction* is the process of inferring laws from existing laws, whereby the deductions made are guaranteed to be true provided that the laws on which they are based are true. In a sense, laws deduced from existing laws add nothing to our stock of knowledge since they are, at best, simply reformulations of existing knowledge.

Mathematical induction is a combination of induction and deduction. It’s a process of looking for patterns in a set of observations, formulating the patterns as *conjectures*, and then testing whether the conjectures can be *deduced* from existing knowledge. Guess-and-verify is a brief way of summarising mathematical induction. (Guessing is the formulation of a conjecture; verification is the process of deducing whether the guess is correct.)

Several of the matchstick games studied in chapter 4 provide good examples of mathematical induction. Recall, for example, the game discussed in section 4.2.2: there is one pile of matches from which it is allowed to remove one or two matches. Exploring this game, we discovered that a pile with 0, 3 or 6 matches is a losing position, and piles with 1, 2, 4, 5, 7 and 8 matches are winning positions. There seems to be a pattern in these numbers: losing positions are the positions in which the number of matches are a multiple of 3, and winning positions are the remaining positions. This is a conjecture about *all* positions made from observations on just nine positions. However, we can verify that the conjecture is true by using mathematical induction to construct a winning strategy.

In order to use induction, we measure the "size" of a pile of matches not by the number of matches but by the number of matches divided by 3, rounded down to the nearest natural number. So, the "size" of a pile of 0, 1 or 2 matches is 0, the "size" of a pile of 3, 4 or 5 matches is 1, and so on. The induction hypothesis is that a pile of $3n$ matches is a losing position, and a pile of $3n+1$ or $3n+2$ matches is a winning position.

The basis for the induction is when n equals 0. A pile of 0 matches is, indeed, a losing position because, by definition, the game is lost when it is no longer possible to move. A pile of 1 or 2 matches is a winning position because the player can remove the matches, leaving the opponent in a losing position.

Now, for the induction step, we assume that a pile of $3n$ matches is a losing position, and a pile of $3n+1$ or $3n+2$ matches is a winning position. We have to show that a pile of $3(n+1)$ matches is a losing position, and a pile of $3(n+1)+1$ or $3(n+1)+2$ matches is a winning position.

Suppose there are $3(n+1)$ matches. The player, whose turn it is, must remove 1 or 2 matches, leaving either $3(n+1)-1$ or $3(n+1)-2$ behind. That is, the opponent is left with either $3n+2$ or $3n+1$ matches. But, by the induction hypothesis, this leaves the opponent in a winning position. Hence, the position in which there are $3(n+1)$ is a losing position.

Now, suppose there are $3(n+1)+1$ or $3(n+1)+2$ matches. By taking 1 match in the first case, and 2 matches in the second case, the player leaves the opponent in a position where there are $3(n+1)$ matches. This we now know to be a losing position. Hence, the positions in which there are $3(n+1)+1$ or $3(n+1)+2$ are both winning positions.

This completes the inductive construction of the winning moves, and thus verifies the conjecture that a position is a losing position exactly when the number of matches is a multiple of 3.

6.5 The Need For Proof

When using induction, it is vital that any conjecture is properly verified. It is too easy to extrapolate from a few cases to a more general claim that is *not* true. Many conjectures turn out to be false; only by subjecting them to the rigours of proof can we be sure of their validity. This section is about a non-trivial example of a false conjecture.

Suppose n points are marked on the circumference of a circular cake and then the cake is cut along the chords joining them. The points are chosen in such a way that all intersection points of pairs of chords are distinct. The question is, in how many portions does this cut the cake?

Figure 6.10 shows the case when n is 1, 2, 3 or 4.

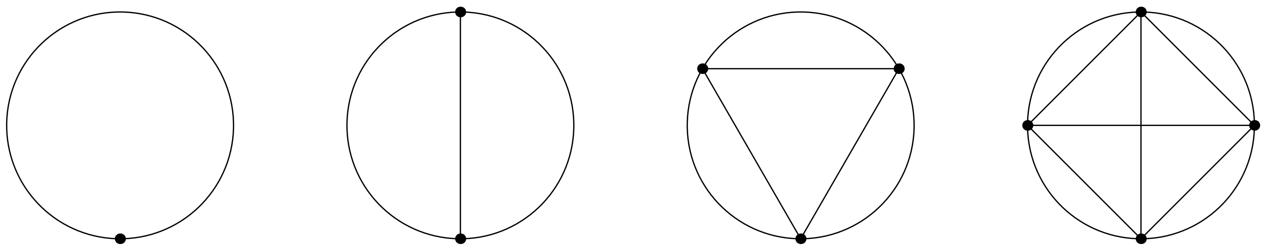


Figure 6.10: Cutting the cake

The number of portions is successively 1, 2, 4 and 8. This suggests that the number of portions, for arbitrary n, is 2^{n-1}. Indeed, this conjecture is supported by the case that $n=5$. (We leave the reader to draw the figure.) In this case, the number of portions is 16, which is 2^{5-1}. However, for $n=6$, the number of portions is 31! (See fig. 6.11.) Note that $n=6$ is the first case in which the points are not allowed to be placed at equal distances around the perimeter.

Had we begun by considering the case that $n=0$, suspicions about the conjecture would already have been raised — it doesn't make sense to say that there are 2^{0-1} portions, even though cutting the cake as stated does make sense! The easy, but inadequate way out, is to dismiss this case, and impose the requirement that n is different from 0. The derivation of the correct formula for the number of portions is too complicated to discuss here, but it does include the case that n equals 0!

6.6 From Verification to Construction

In mathematical texts, induction is often used to *verify* known formulae. Verification is important but has a major drawback — it seems that a substantial amount of clairvoyance

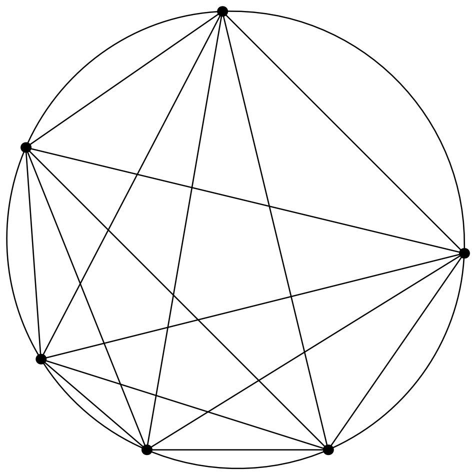


Figure 6.11: Cutting the cake. The case $n=6$. The number of portions is 31, not 2^{6-1}

is needed to come up with the formula that is to be verified. And, if one's conjecture is wrong, verification gives little help in determining the correct formula.

Induction is *not* important in computing science as a verification principle but because it is a fundamental principle in the *construction* of computer programs. This section introduces the use of induction in the *construction* of mathematical formulae.

The problem we consider is how to determine a closed formula for the sum of the kth powers of the first n natural numbers.

A well-known formula gives the sum of the natural numbers from 1 thru n:

$$1+2+\ldots+n = \frac{1}{2}n(n+1) \quad .$$

Two other exercises, often given in mathematical texts, are to verify that

$$1^2+2^2+\ldots+n^2 \;=\; \frac{1}{6}n(n+1)(2n+1)$$

and

$$1^3+2^3+\ldots+n^3 \;=\; \frac{1}{4}n^2(n+1)^2 \quad .$$

As well as being good examples of the strength of the principle of mathematical induction, the examples also illustrate the weakness of verification: the technique works if the answer is known, but what happens if the answer is not already known! Suppose, for example, that you are now asked to determine a closed formula for the sum of the 4th powers of the first n numbers

$$1^4+2^4+\ldots+n^4 \;=\; ? \quad .$$

How would you proceed? Verification, using the principle of mathematical induction, does not seem to be applicable unless we already know the right side of the equation. Can you guess what the right side would be in this case? Can you guess what the right side would be in the case that the term being summed is, say, k^{27}? Almost certainly, not!

Constructing solutions to non-trivial problems involves a creative process. This means that a certain amount of guesswork is necessary, and trial-and-error cannot be completely eliminated. Reducing the guesswork to a minimum, replacing it by mathematical calculation is the key to success.

Induction can be used to construct closed formulae for such summations. The general idea is to seek a pattern, formulate the pattern in precise mathematical terms and then verify the pattern. The key to success is simplicity. Don't be over-ambitious. Leave the work to mathematical calculation.

A simple pattern in the formulae displayed above is that, for m equal to 1, 2 and 3, the sum of the mth powers of the first n numbers is a polynomial in n of degree $m+1$. (The sum of the first n numbers is a quadratic function of n, the sum of the first n squares is a cubic function of n, and the sum of the first n cubes is a quartic function of n.) This pattern is also confirmed in the (oft-forgotten) case that m is 0:

$$1^0+2^0+\ldots+n^0 \;=\; n \quad .$$

A strategy for determining a closed formula for, say, the sum of the fourth powers is thus to guess that it is a fifth degree polynomial in n and then *use induction to calculate the coefficients*. The calculation in this case is quite long, so let us illustrate the process by showing how to construct a closed formula for $1+2+\ldots+n$. (Some readers will already know a simpler way of deriving the formula in this particular case. If this is the case, please bear with us. The method described here is more general.)

We conjecture that the required formula is a second degree polynomial in n, say $a+bn+cn^2$ and then calculate the coefficients a, b and c. Here is how the calculation goes.

For brevity, let us use $S.n$ to denote

$$1+2+\ldots+n \quad .$$

We also use $P.n$ to denote the proposition

$$S.n \;=\; a+bn+cn^2 \quad .$$

Then,

$$P.0$$

$$= \qquad \{ \quad \text{definition of } P \quad \}$$

$$
\begin{array}{ll}
 & S.0 \; = \; a+b\times 0+c\times 0^2 \\
= & \{ \quad S.0=0 \text{ (the sum of an empty set of numbers} \\
 & \quad \text{is zero) and arithmetic} \quad \} \\
 & 0=a \quad .
\end{array}
$$

So the basis of the induction has allowed us to deduce that a, the coefficient of n^0, is 0. Now, we calculate b and c. To do so, we make the induction hypothesis that $0\leq n$ and $P.n$ is true. Then

$$
\begin{array}{ll}
 & P.(n{+}1) \\
= & \{ \quad \text{definition of } P\,,\; a=0 \quad \} \\
 & S.(n{+}1) \; = \; b(n{+}1)+c(n{+}1)^2 \\
= & \{ \quad \text{heading for use of the induction hypothesis,} \\
 & \quad S.(n{+}1) \; = \; S.n+n+1 \quad \} \\
 & S.n+n+1 \; = \; b(n{+}1)+c(n{+}1)^2 \\
= & \{ \quad \text{assumption: } P.n\,. \text{ Also, } a=0\,. \\
 & \quad \text{That is, } S.n \; = \; bn+cn^2 \quad \} \\
 & bn+cn^2+n+1 \; = \; b(n{+}1)+c(n{+}1)^2 \\
= & \{ \quad \text{arithmetic} \quad \} \\
 & cn^2+(b{+}1)n+1 \; = \; cn^2+(b+2c)n+b+c \\
\Leftarrow & \{ \quad \text{comparing coefficients of powers of } n \quad \} \\
 & c=c \;\wedge\; b{+}1 = b+2c \;\wedge\; 1=b+c \\
= & \{ \quad \text{arithmetic} \quad \} \\
 & \frac{1}{2}=c \wedge \frac{1}{2}=b \quad .
\end{array}
$$

From the conjecture that the sum of the first n numbers is a quadratic in n, we have thus calculated that

$$1+2+\ldots+n \; = \; \frac{1}{2}n+\frac{1}{2}n^2 \quad .$$

Extrapolating from this calculation, one can see that it embodies an algorithm to express $1^m+2^m+\ldots+n^m$ as a polynomial function for any given natural number m. The steps in the algorithm are: postulate that the summation is a polynomial in n with degree $m{+}1$. Use the principle of mathematical induction together with the facts that $S.0$ is 0 and $S.(n{+}1)$ is $S.n+(n{+}1)^m$ (where $S.n$ denotes $1^m+2^m+\ldots+n^m$) to determine a system of simultaneous equations in the coefficients. Finally, solve the system of equations.

Remark: In the case of the sum $1+2+\ldots+n$ there is an easier way to derive the correct formula. Simply write down the required sum

$$\begin{array}{ccccccc} 1 & + & 2 & + & \ldots & + & n\ , \end{array}$$

and immediately below it

$$\begin{array}{ccccccc} n & + & n{-}1 & + & \ldots & + & 1\ . \end{array}$$

Then add the two rows together:

$$\begin{array}{ccccccc} n{+}1 & + & n{+}1 & + & \ldots & + & n{+}1\ . \end{array}$$

From the fact that there are n occurrences of $n{+}1$ we conclude that the sum is $\frac{1}{2}n(n{+}1)$. However, this method cannot be used for determining $1^m+2^m+\ldots+n^m$ for m greater than 1. *End of remark.*

Exercise 6.3 Use the technique just demonstrated to construct closed formulae for

$$1^0+2^0+\ldots+n^0 \quad\text{and}\quad 1^2+2^2+\ldots+n^2\ .$$

□

Exercise 6.4 Consider a matchstick game with one pile of matches from which m thru n matches can be removed. By considering a few simple examples (for example, m is 1 and n is arbitrary, or m is 2 and n is 3), formulate a general rule for determining which are the winning positions and which are the losing positions, and what the winning strategy is.

Avoid guessing the complete solution. Try to identify a simple pattern in the way winning and losing positions are grouped. Introduce variables to represent the grouping, and calculate the values of the variables.

□

6.7 Fake-Coin Detection

The motto of section 6.6 can be summarised as “Don’t guess! Calculate.” We put this into practice in this section.

Suppose we are given a number of coins, each of the same size and shape. We are told that among them there is at most one "fake" coin, and all the rest are "genuine". All "genuine" coins have the same weight, whereas a "fake" coin has a different weight to a "genuine" coin. The problem is how to use the pair of scales optimally in order to find the fake coin, if there is one.

Note the element of vagueness in this problem statement; we don't say what we mean by using the scales "optimally". This is deliberate. Often, an essential element of problem solving is to clearly identify the problem itself. Our formulation of the problem and its eventual solution illustrates several other aspects of "real" problem solving. Several stages are needed, including some "backtracking" and revision.

6.7.1 Problem Formulation

When we use a pair of scales to compare two weights —an operation that we call a *comparison*— there are 3 possible outcomes: the scales may tip to the left, they may balance, or they may tip to the right. This means that with n comparisons, there are at most 3^n different outcomes[3]. This gives an upper bound on what can be achieved using a pair of scales.

Now, suppose we are given m coins, of which at most one is fake and the rest are genuine. Then there are $1+2m$ different possibilities that can be observed with a pair of scales: "1" possibility is that all coins are genuine; otherwise, there are "2" ways that each of the "m" coins may be fake (by being lighter or heavier than a genuine coin). This means that, with n comparisons, the number of coins among which at most one fake coin can be detected is at most m, where $1+2m = 3^n$. More precisely, if the number, m, of coins is greater than $(3^n{-}1)/2$, it is impossible to guarantee that a fake coin can be found with n comparisons.

We have almost reached the point at which we can state our problem precisely. We conjecture that, given $(3^n{-}1)/2$ coins of which at most one is fake, it is possible to establish that all are genuine or identify the fake coin (and whether it is lighter or heavier than a genuine coin) using at most n comparisons.

For n equal to 0, the conjecture is clearly true; in this case, there are no coins, all of which are genuine. For n equal to 1, however, we run into a problem. The assumption is that there is one coin (since $(3^1{-}1)/2 = 1$). But how can we tell whether this one coin is fake or genuine, if there are no other coins to compare it with? Our conjecture has broken down, and needs revision.

We propose to modify the conjecture by assuming that we have at our disposal at least one *additional* coin that is known to be genuine. Thus, we are given $(3^n{-}1)/2$ coins about which we know nothing except that at most one is fake, and we are also given

[3]Note the implicit use of induction here.

at least one coin that is known to be genuine. The problem is to construct an algorithm that will identify the fake coin, if it exists, or determine that all coins are genuine, using at most n comparisons.

6.7.2 Problem Solution

Our formulation of the problem begs the use of induction on the number of comparisons, n, in its solution.

The Basis With zero comparisons, we can report immediately that all coins in a collection of $(3^0-1)/2$ are genuine. The base case, n equal to 0, is thus solved.

Induction Step Now, we tackle the induction step. Suppose n is at least zero. For brevity, let us use $c.n$ to denote $(3^n-1)/2$. By induction, we may assume that a fake coin, if it exists, can be found among $c.n$ coins using at most n comparisons. We have to show how to find a fake coin, if it exists, among $c.(n+1)$ coins, using at most $n+1$ comparisons.

Consider the first comparison. It involves putting some number of coins on the left scale, some on the right scale, and leaving some on the table. To be able to draw any conclusion from the comparison, the number of coins on the two scales must be equal. One possible consequence of the comparison is that the scales balance, from which one infers that none of the coins on the scales is fake. The algorithm would then proceed to try to find a fake coin among the coins left on the table.

Combined with the induction hypothesis, this dictates that $c.n$ coins must be left on the table. This is because $c.n$ is the maximum number of coins among which a fake coin can be found with n comparisons.

It also dictates how many coins should be put on the scales — this is the difference between $c.(n+1)$ and $c.n$. Now,

$$c.(n+1) \quad = \quad (3^{n+1}-1)/2 \quad = \quad 3\times((3^n-1)/2)+1 \quad = \quad 3\times c.n+1 \quad .$$

So

$$c.(n+1)-c.n \quad = \quad 2\times c.n+1 \quad = \quad 3^n \quad .$$

This is an odd number; it can be made even by using one of the coins we know to be genuine. (Recall the assumption that we have at least one coin that is known to be genuine, in addition to the $c.(n+1)$ coins whose kind we must determine.) We conclude that in the first comparison, $c.n+1$ coins should be put on each of the two scales.

The next step is to determine what to do after the first comparison is made. There are three possible outcomes, of which we have already discussed one. If the scales balance, the fake coin should be sought among the $c.n$ coins left on the table. The problem is what to do if the scales tip either to the left or to the right.

At this point, we realise that the induction hypothesis doesn't help. It is too weak! If the scales tip to one side, we can conclude that all the coins left on the table are genuine, and can be eliminated from consideration. But we are still left with 3^n coins none of which we know to be genuine. And crucially, 3^n is greater than $c.n$. We are unable to apply the induction hypothesis to this number of coins.

The comparison does tell us something about the coins on the scales. If the scales tip to one side, we know that all the coins on that side are *possibly heavier* than a genuine coin, and all the coins on the other side are *possibly lighter* than a genuine coin. By "possibly lighter" we mean genuine, or fake and lighter. By "possibly heavier" we mean genuine, or fake and heavier. After the comparison, we can mark all the coins on the scales one way or the other.

The Marked Coin Problem In this way, in the case that the scales do not balance, the problem we started with has been reduced to a different problem. The new problem is this. Suppose a number of coins are supplied, each of which is marked either "possibly light" or "possibly heavy". Exactly one of the coins is fake, and all the rest are genuine. Construct an algorithm that will determine, with at most n comparisons, the fake coin among 3^n marked coins.

Again, the base case is easy. If n equals 0, there is one coin, which must be the fake coin. That is, 0 (i.e. n) comparisons are needed to determine this fact.

For the induction step, we proceed as for the earlier problem. Suppose we are supplied with 3^{n+1} marked coins. In the first comparison, some coins are put on the left scale, some on the right, and some are left on the table. In order to apply the induction hypothesis in the case that the scales balance, the coins must be divided equally: 3^n coins must be left on the table, and thus 3^n put on the left scale and 3^n on the right scale.

The coins are marked in two different ways. So, we need to determine how to place the coins according to their markings. We calculate the numbers as follows.

Suppose $l1$ possibly light coins are placed on the left scale and $l2$ possibly light coins on the right scale. Similarly, suppose $h1$ possibly heavy coins are placed on the left scale and $h2$ possibly heavy coins on the right scale.

To draw any conclusion from the comparison, we require that the number of coins on the left scale equals the number on the right. That is, $l1+h1$ and $l2+h2$ should be equal. Furthermore, as already determined, they should equal 3^n.

Now, if the comparison causes the scales to tip to the left, we conclude that all coins on the left scale are possibly heavy, and all the coins on the right scale are possibly light. Combining this with the markings, we conclude that the $l1$ possibly light coins on the left scale and the $h2$ possibly heavy coins on the right scale are in fact genuine (since possibly heavy and possibly light equals genuine); this leaves $h1+l2$ coins to be investigated further. Conversely, if the scale tips to the right, the $h1$ possibly heavy coins on the left scale and the $l2$ possibly heavy coins on the right scale are genuine, leaving $l1+h2$ coins to be investigated further.

Again, in order to apply the induction hypothesis, we require that the number of coins not eliminated be equal to 3^n, whatever the outcome of the comparison. This imposes the requirement that $h1+l2 = l1+h2 = 3^n$. Together with $l1+h1 = l2+h2$, we infer that $l1 = l2$ and $h1 = h2$. We must arrange the coins so that each scale contains equal numbers of coins of the same kind.

This requirement can be met. Simply place the coins on the scales two at a time, one on the left and one on the right, until each scale has its full complement of 3^n coins, always choosing two coins with the same marking. The choice can always be made because there are always at least three coins from which to choose; by choosing any three coins, at least two of them will have the same marking.

The Complete Solution This completes the solution to the marked-coin problem, and thus to the unmarked-coin problem. The fake coin is identified from a collection of 3^{n+1} marked coins by placing 3^n coins on each scale, in such a way that there is an equal number of possibly light coins on each of the scale. According to the outcome of the comparison, one of the following is executed.

- If the scales balance, all the coins on the scales are genuine. Proceed with the coins left on the table.
- If the scales tip to the left, the coins on the table are genuine. So too are the possibly light coins on the left scale and the possibly heavy coins on the right scale. Proceed with the possibly heavy coins on the left scale and the possibly light coins on the right scale.
- If the scales tip to the right, the coins on the table are genuine. So too are the possibly light coins on the right scale and the possibly heavy coins on the left scale. Proceed with the possibly heavy coins on the right scale and the possibly light coins on the left scale.

The solution to the unmarked-coin problem when the number of coins is $(3^{n+1}-1)/2$ is as follows.

Divide the coins into three groups of sizes $(3^n-1)/2$, $(3^n-1)/2+1$ and $(3^n-1)/2$. Place the first group on the left scale together with the supplied genuine coin. Place the second group on the right scale, and leave the third group on the table. Determine the outcome of the comparison, and proceed as follows:

- If the scales balance, all the coins on the balance are genuine. Apply the solution to the unmarked-coin problem (inductively) to the coins on the table.

- If the scales tip to the left, the coins on the table are genuine. Mark all the coins on the left scale, with the exception of the supplied genuine coin as “possibly heavy”. Mark the coins on the right scale as “possibly light”. Apply the solution to the marked-coin problem to the 3^n marked coins.

- If the scales tip to the right, the coins on the table are genuine. Mark all the coins on the left scale, with the exception of the supplied genuine coin as “possibly light”. Mark the coins on the right scale as “possibly heavy”. Apply the solution to the marked-coin problem to the 3^n marked coins.

We ask the reader to review the development of this algorithm. Note that at no stage is a *guess* made at an inductive hypothesis, even though the development necessitates several such hypotheses. Quite the opposite: each hypothesis is systematically *calculated* from the available information. This is the epitome of the art of effective reasoning.

Exercise 6.5 Suppose you are given a number of objects. All the objects have the same weight, with the exception of one, called the *unique* object, which has a different weight. In all other respects, the objects are identical. You are required to determine which is the unique object. For this purpose, you are provided with a pair of scales.

Show, by induction on m, that at most $2\times m$ comparisons are needed to identify the unique object when the total number of objects is 3^m. (Hint: for the induction step, 3^{m+1} objects can be split into 3 groups of 3^m objects.)

Can you identify whether the unique object is lighter or heavier than all the other objects?

□

Exercise 6.6 Given are n objects, where $1\leq n$, each of different weight. A pair of scales is provided so that it is possible to determine, for any two of the objects, which is the lighter and which is the heavier.
a) How many comparisons are needed to find the lightest object?
b) Show, by induction on n, that it is possible to determine which is the lightest and which is the heaviest object using $2n-3$ comparisons. Assume that $2\leq n$.
c) Suppose there are 4 objects with weights A, B, C and D, and suppose $A<B$ and $C<D$. Show how to find the lightest and heaviest of all four with two additional comparisons. Use this to show how to find the lightest and heaviest of 4 objects using 4 comparisons (and not 5, as in your solution to part (b)).

d) Suppose there are $2m$ objects, where $1 \leq m$. Show, by induction on m, that it is possible to find the lightest and heaviest objects using $3m-2$ comparisons. (Hint: make use of (c).)

□

6.8 Summary

Induction is one of the most important problem-solving principles. The principle of mathematical induction is that instances of a problem of arbitrary "size" can be solved for all "sizes" if

(a) instances of "size" 0 can be solved,

(b) given a method of solving instances of "size" n, for arbitrary n, it is possible to adapt the method to solve instances of "size" $n+1$.

Using induction means looking for patterns. The process may involve some creative guesswork, which is then subjected to the rigours of mathematical deduction. The key to success is to reduce the guesswork to a minimum, by striving for simplicity, and using mathematical calculation to fill in complicated details.

6.9 Bibliographic Notes

The solution to the fake-coin problem is a combination of two papers by Edsger W. Dijkstra [Dij90, Dij97].

Chapter 7

The Towers of Hanoi

This chapter is about the Towers of Hanoi problem. The problem is discussed in many mathematical texts, and is often used in computing science and artificial intelligence as an illustration of "recursion" as a problem-solving strategy.

The Towers of Hanoi problem is a puzzle that is quite difficult to solve without a systematic problem-solving strategy. Induction gives a systematic way of constructing a first solution. However, this solution is undesirable. A better solution is obtained by observing an invariant of the inductive solution. In this way, this chapter brings together a number of the techniques discussed earlier: principally induction and invariants, but also the properties of logical equivalence.

For this problem, we begin with the solution of the problem. One reason for doing so is to make clear where we are headed; the Towers of Hanoi problem is one that is not solved in one go; several steps are needed before a satisfactory solution is found. Another reason is to illustrate how difficult it can be to understand *why* a correct solution has been found if no information about the solution method is provided.

7.1 Specification and Solution

7.1.1 The End of the World!

The Towers of Hanoi problem comes from a puzzle marketed in 1883 by the French mathematician Édouard Lucas, under the pseudonym M. Claus.

The puzzle is based on a legend according to which there is a temple in Bramah where there are three giant poles fixed in the ground. On the first of these poles, at the time of the world's creation, God placed sixty-four golden disks, each of different size, in decreasing order of size. (See fig. 7.1.) The Brahmin monks were given the task of moving the disks, one per day, from one pole to another according to the rule that no disk may ever be above a smaller disk. The monks' task will be complete when they have succeeded in moving all the disks from the first of the poles to the second and, on the day that they complete their task, the world will come to an end!

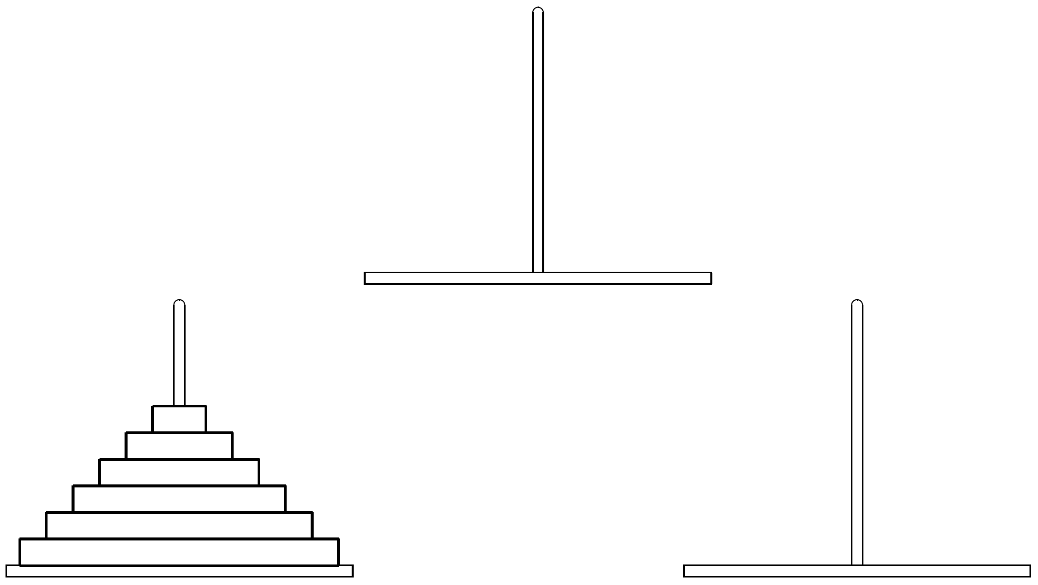


Figure 7.1: Towers of Hanoi Problem

7.1.2 Iterative Solution

There is a very easy solution to the Towers of Hanoi problem that is easy to remember and easy to execute. To formulate the solution, we assume that the poles are arranged at the three corners of a triangle. Movements of the disks can then be succinctly described as either clockwise or anticlockwise movements. We assume that the problem is to move all the disks from one pole to the next in a clockwise direction. We also assume that days are numbered from 0 onwards. On day 0, the disks are placed in their initial position and the monks begin moving the disks on day 1. With these assumptions, the solution is the following.

> On every alternate day, beginning on the first day, the smallest disk is moved. The rule for moving the smallest disk is that it should *cycle* around the poles. The direction of rotation depends on the total number of disks. If the total number of disks is odd, the smallest disk should cycle in a clockwise direction. Otherwise, it should cycle in an anticlockwise direction.

On every other day, a disk other than the smallest disk is moved — subject to the rule that no disk may ever be above a smaller disk. It is easy to see that because of this rule there is exactly one move possible so long as not all the disks are on one pole.

The algorithm terminates when no further moves are possible, that is, on an even-numbered day when all the disks are on one-and-the-same pole.

Try executing this algorithm yourself on, say, a 4-disk puzzle. Take care to cycle the smallest disk on the odd-numbered moves and to obey the rule not to place a disk on top of a disk smaller than itself on the even-numbered moves. If you do, you will find that the algorithm works. Depending on how much patience you have, you can execute the algorithm on larger and larger problems — 5-disk, 6-disk, and so on.

7.1.3 WHY?

Presenting the problem *and* its solution, like this, provides no help whatsoever in understanding how the solution is constructed. If anything, it only serves to impress —look at how clever I am!— but in a reprehensible way. Matters would be made even worse if we now proceeded to give a formal mathematical verification of the correctness of the algorithm. This is not how we intend to proceed! Instead, we first present an inductive solution of the problem. Then, by observing a number of invariants, we show how to derive the algorithm above from the inductive solution.

7.2 Inductive Solution

Constructing a solution by induction on the number of disks is an obvious strategy.

Let us begin with an attempt at a simple-minded inductive solution. Suppose that the task is to move M disks from one specific pole to another specific pole. Let us call these poles A and B, and the third pole C. (Later, we see that naming the poles is inadvisable.)

As often happens, the basis is easy. When the number of disks is 0 no steps are needed to complete the task. For the inductive step, we assume that we can move n disks from A to B, and the problem is to show how to move $n+1$ disks from A to B.

Here, we soon get stuck! There is only a couple of ways that the induction hypothesis can be used, but these lead nowhere:

1. Move the top n disks from A to B. After doing this, we have exhausted all possibilities of using the induction hypothesis because n disks are now on pole B, and we have no hypothesis about moving disks from this pole.

2. Move the smallest disk from A to C. Then move the remaining n disks from A to B. Once again, we have exhausted all possibilities of using the induction hypothesis, because n disks are now on pole B, and we have no hypothesis about moving disks from this pole.

The mistake we have made is to be too specific about the induction hypothesis. The way out is to generalise by introducing one or more parameters to model the start and finish positions of the disks.

At this point, we make a crucial decision. Rather than name the poles (A, B and C, say), we observe that the problem exhibits a *rotational* symmetry. The rotational symmetry is obvious when the poles are placed at the corners of an equilateral triangle, as we did in fig. 7.1. (This rotational symmetry is obscured by placing the poles in a line, as is often done.) The problem does not change when we rotate the poles and disks about the centre of the triangle.

The importance of this observation is that only *one* additional parameter needs to be introduced, namely, the direction of movement. That is, in order to specify how a particular disk is to be moved, we need only say whether it is to be moved clockwise or anticlockwise from its current position. Also, the generalisation of the Towers of Hanoi problem becomes how to move n disks from one pole to the next in the direction d, where d is either clockwise or anticlockwise. The alternative of naming the poles leads to the introduction of two additional parameters, the start and finish positions of the disks. This is much more complicated since it involves unnecessary additional detail.

Now, we can return to the inductive solution again. We need to take care in formulating the induction hypothesis. It is not sufficient to simply take the problem specification as induction hypothesis. This is because the problem specification assumes that there are exactly M disks that are to be moved. When using induction, it is necessary to move n disks in the presence of $M-n$ other disks. If some of these $M-n$ disks are smaller than the n disks being moved, the requirement that a larger disk may not be placed on top of a smaller disk may be violated. We need a stronger induction hypothesis.

The induction hypothesis we use is that it is possible to move the n *smallest* disks, from one pole to its neighbour in the direction d, beginning from any valid starting position (that is, a starting position in which the disks are distributed arbitrarily over the poles, but no disk is on top of a disk smaller than itself).

In the case that n is 0, the sequence of moves is the empty sequence. In the case of $n+1$ disks we assume that we have a method of moving the n smallest disks from one pole to either of its two neighbours. We must show how to move $n+1$ disks from one pole to its neighbour in direction d, where d is either clockwise or anticlockwise. For convenience, we assume that the disks are numbered from 1 upwards, with the smallest disk being given number 1.

Given the goal of exploiting the induction hypothesis, there is little choice of what to do. We can begin by moving the n smallest disks in the direction d, or in the direction $\neg d$. Any other initial choice of move would preclude the use of the induction hypothesis. Some further thought (preferably assisted by a physical model of the problem) reveals that the solution is to move the n smallest disks in the direction $\neg d$. Then disk $n{+}1$ can be moved in the direction d. (This action may place disk $n{+}1$ on top of another disk. However, the move is valid because the n disks smaller than disk $n{+}1$ are not on the pole to which disk $n{+}1$ is moved.) Finally, we use the induction hypothesis again to move the n smallest disks in the direction $\neg d$. This places them above disk $n{+}1$, and all $n{+}1$ smallest disks have now been moved from their original position to the neighbouring pole in direction d.

The following code summarises this inductive solution to the problem. The code defines $H_n.d$ to be a sequence of pairs $\langle k , d' \rangle$ where n is the number of disks, k is a disk number and d and d' are directions. Disks are numbered from 1 onwards, disk 1 being the smallest. Directions are boolean values, true representing a clockwise movement and false an anti-clockwise movement. The pair $\langle k , d' \rangle$ means move the disk numbered k from its current position in the direction d'. The semicolon operator concatenates sequences together, $[\,]$ denotes an empty sequence and $[x]$ is a sequence with exactly one element x. Taking the pairs in order from left to right, the complete sequence $H_n.d$ prescribes how to move the n smallest disks one-by-one from one pole to its neighbour in the direction d, following the rule of never placing a larger disk on top of a smaller disk.

$$\begin{aligned} H_0.d &= [\,] \\ H_{n+1}.d &= H_n.\neg d \;;\; [\langle n{+}1 \,,\, d \rangle] \;;\; H_n.\neg d \end{aligned}$$

Note that the procedure name H recurs on the right side of the equation for $H_{n+1}.d$. Because of this we have what is called a *recursive* solution to the problem. Recursion is a very powerful problem-solving technique, but unrestricted use of recursion can be unreliable. The form of recursion used here is limited; describing the solution as an "inductive" solution makes clear the limitation on the use of recursion.

This inductive procedure gives us a way to generate the solution to the Towers of Hanoi problem for any given value of n — we simply use the rules as left-to-right rewrite rules until all occurrences of H have been eliminated. For example, here is how we determine $H_2.\mathsf{cw}$. (We use cw and aw, meaning clockwise and anticlockwise, rather than true and false in order to improve readability.)

$$H_2.\mathsf{cw}$$

$$= \qquad \{ \qquad \text{2nd equation, } n,d := 1,\mathsf{cw} \qquad \}$$

$$
\begin{array}{ll}
 & H_1.\mathrm{aw}\ ;\ [\langle 2,\mathrm{cw}\rangle]\ ;\ H_1.\mathrm{aw} \\
= & \{\quad \text{2nd equation, } n,d := 0,\mathrm{aw} \quad\} \\
 & H_0.\mathrm{cw}\ ;\ [\langle 1,\mathrm{aw}\rangle]\ ;\ H_0.\mathrm{cw}\ ;\ [\langle 2,\mathrm{cw}\rangle]\ ;\ H_0.\mathrm{cw}\ ;\ [\langle 1,\mathrm{aw}\rangle]\ ;\ H_0.\mathrm{cw} \\
= & \{\quad \text{1st equation} \quad\} \\
 & [\,]\ ;\ [\langle 1,\mathrm{aw}\rangle]\ ;\ [\,]\ ;\ [\langle 2,\mathrm{cw}\rangle]\ ;\ [\,]\ ;\ [\langle 1,\mathrm{aw}\rangle]\ ;\ [\,] \\
= & \{\quad \text{concatenation of sequences} \quad\} \\
 & [\langle 1,\mathrm{aw}\rangle\ ,\ \langle 2,\mathrm{cw}\rangle\ ,\ \langle 1,\mathrm{aw}\rangle]\quad .
\end{array}
$$

As an exercise you should determine $H_3.\mathrm{aw}$ in the same way. If you do, you will quickly discover that this inductive solution to the problem takes a lot of effort to put into practice. The complete expansion of the equations in the case of $n=3$ takes 16 steps, in the case of $n=4$ takes 32 steps, and so on. This is not the easy solution that the Bramin monks are using! The solution given in section 7.1.1 is an *iterative* solution to the problem. That is, it is a solution that involves iteratively (i.e. repeatedly) executing a simple procedure dependent only on the current state. The implementation of the inductive solution, on the other hand, involves maintaining a stack of the sequence of moves yet to be executed. The memory of Bramin monks is unlikely to be large enough to do that!

Exercise 7.1 The number of days the monks need to complete their task is the length of the sequence $H_{64}.\mathrm{cw}$. Let $T_n.d$ denote the length of the sequence $H_n.d$. Derive an inductive definition of T from the inductive definition of H. (You should find that $T_n.d$ is independent of d.) Use this definition to evaluate T_o, T_1 and T_2. Hence, or otherwise, formulate a conjecture expressing T_n as an arithmetic function of n. Prove your conjecture by induction on n.

□

Exercise 7.2 Use induction to derive a formula for the number of different states in the Towers of Hanoi problem.

Use induction to show how to construct a state-transition diagram that shows all possible states of n disks on the poles, and the allowed moves between states.

Use the construction to show that the above solution optimises the number of times that disks are moved.

□

7.3 The Iterative Solution

Recall the iterative solution to the problem, presented in section 7.1.2. It has two main elements: the first is that the smallest disk cycles around the poles (that is, its direction of movement is invariantly clockwise or invariantly anticlockwise), the second is that the disk to be moved alternates between the smallest disk and some other disk. In this section, we show how these properties are derived from the inductive solution.

Cyclic Movement of the Disks

In this section, we show that the smallest disk always cycles around the poles. In fact, we do more than this. We show that *all* the disks cycle around the poles, and we calculate the direction of movement of each.

The key is that, for all pairs $\langle k, d' \rangle$ in the sequence $H_{n+1}.d$ the boolean value $even.k \equiv d'$ is invariant (that is always true or always false). This is a simple consequence of the rule of contraposition discussed in section 5.5. When the formula for $H_{n+1}.d$ is applied, the parameter "$n{+}1$" is replaced by "n" and "d" is replaced by "$\neg d$". Since $even.(n{+}1) \equiv \neg(even.n)$, the value of $even.(n{+}1) \equiv d$ remains constant under this assignment.

Whether $even.k \equiv d'$ is true or false (for all pairs $\langle k, d' \rangle$ in the sequence $H_{n+1}.d$) will depend on the initial values of n and d. Let us suppose these are N and D. Then, for all moves $\langle k, d \rangle$, we have

$$even.k \;\equiv\; d \;\equiv\; even.N \;\equiv\; D \quad .$$

This formula allows us to determine the direction of movement d of disk k. Specifically, if it is required to move an even number of disks in a clockwise direction, all even-numbered disks should cycle in a clockwise direction, and all odd-numbered disks should cycle in an anticlockwise direction. Vice-versa, if it is required to move an odd number of disks in a clockwise direction, all even-numbered disks should cycle in an anticlockwise direction, and all odd-numbered disks should cycle in a clockwise direction. In particular, the smallest disk (which is odd-numbered) should cycle in a direction opposite to D if N is even, and the same direction as D if N is odd.

Exercise 7.3 An explorer once discovered the Bramin temple and was able to secretly observe the monks performing their task. At the time of his discovery, the monks had got some way to completing their task, so that the disks were arranged on all three poles. The poles were arranged in a line and not at the corners of the triangle so he wasn't sure which direction was clockwise and which anticlockwise. However, on the day of his arrival he was able to observe the monks move the smallest disk from the middle pole to the rightmost pole. On the next day, he saw the monks move a disk from the middle pole to the leftmost pole. Did the disk being moved have an even number or an odd number?

□

Alternate Disks

We now turn to the second major element of the solution, namely that the disk that is moved alternates between the smallest disk and some other disk.

By examining the puzzle itself, it is not difficult to see that this must be the case. After all, two consecutive moves of the smallest disk are wasteful as they can always be combined into one. And, two consecutive moves of a disk other than the smallest have no effect on the state of the puzzle. We now want to give a formal proof that the sequence $H_n.d$ satisfies this property.

Let us call a sequence of numbers *alternating* if it has two properties. The first property is that consecutive elements alternate between one and a value greater than one; the second property is that if the sequence is non-empty then it begins and ends with the value one. We write $\mathsf{alt}.ks$ if the sequence ks has these two properties.

The sequence of disks moved on successive days, which we denote by $disk_n.d$, is obtained by taking the first component of each of the pairs in $H_n.d$ and ignoring the second. Let the sequence that is obtained in this way be denoted by $disk_n.d$. Then, from the definition of H we get:

$$\begin{aligned} disk_0.d &= [\,] \\ disk_{n+1}.d &= disk_n.\neg d \;\; ; \;\; [n{+}1] \;\; ; \;\; disk_n.\neg d \;\; . \end{aligned}$$

Our goal is to prove $\mathsf{alt}.(disk_n.d)$. The proof is by induction on n. The base case, $n=0$, is clearly true because an empty sequence has no consecutive elements. For the induction step, the property of alternating sequences on which the proof depends is that, for a sequence ks and number k,

$$\mathsf{alt}.(ks\,;\,[k]\,;ks) \quad \Leftarrow \quad \mathsf{alt}.ks \;\wedge\; ((ks=[\,]) \equiv (k=1)) \;\; .$$

The proof is then:

$$\begin{array}{ll} & \mathsf{alt}.(disk_{n+1}.d) \\ = & \qquad \{ \qquad \text{definition} \quad \} \\ & \mathsf{alt}.(disk_n.\neg d \;\; ; \;\; [n{+}1] \;\; ; \;\; disk_n.\neg d) \\ \Leftarrow & \qquad \{ \qquad \text{above property of alternating sequences} \quad \} \end{array}$$

$$
\begin{array}{lll}
 & \mathsf{alt}.(disk_n.\neg d) \ \wedge \ ((disk_n.\neg d = []) \equiv (n{+}1 = 1)) & \\
= & \{ \quad \text{induction hypothesis applied to the first conjunct,} & \\
 & \quad\ \ \text{straightforward property of } disk_n \text{ for the second.} \quad \} & \\
 & \mathsf{true} \ . &
\end{array}
$$

Exercise 7.4 The explorer left the area and did not return until several years later. On his return, he discovered the monks in a state of great despair. It transpired that one of the monks had made a mistake shortly after the explorer's first visit but it had taken the intervening time before they had discovered the mistake. The state of the disks was still valid but the monks had discovered that they were no longer making progress towards their goal; they had got into a never-ending loop!

Fortunately, the explorer was able to tell the monks how to proceed in order to return all the disks to one-and-the-same pole whilst still obeying the rules laid down to them on the day of the world's creation. They would then be able to begin their task afresh.

What was the algorithm the explorer gave to the monks? Say why the algorithm is correct. (Hint: The disk being moved will still alternate between the smallest and some other disk. You only have to decide in which direction the smallest disk should be moved. Because of the monks' mistake this will not be constant. Make use of the fact that, beginning in a state in which n disks are all on the same pole, maintaining invariant the relationship

$$even.n \equiv d \equiv even.k \equiv d'$$

for the direction d' moved by disk k will move n disks in the direction d.)

□

Exercise 7.5 (Coloured Disks) Suppose each disk is coloured, red white or blue. The colouring of disks is random; different disks may be coloured differently.

Devise an algorithm that will sort the disks so that all the red disks are on one pole, all the white disks are on another pole, and all the blue disks are on the third pole. You may assume that, initially, all disks are on one pole.

□

7.4 Summary

In this chapter we have seen how to use induction to construct a solution to the Towers of Hanoi problem. Several inductive constructions have been discussed. The chapter began with an inductive construction of a graph representing all possible moves of the disks in the general n-disk problem. The graph was used to justify an inductive solution to the problem itself. This solution was then tranformed to an iterative solution, inductive proofs of properties of the sequence of movements of the disks being used to establish the correctness of the iterative solutions.

The chapter has also illustrated two important design considerations: the inclusion of the 0-disk problem as the basis for the construction (rather than the 1-disk problem) and the avoidance of unnecessary detail by not naming the poles and referring to the direction of movement of the disks (clockwise or anticlockwise) instead.

7.5 Bibliographic Remarks

Information on the history of the Towers of Hanoi problem is taken from [Ste97]. A proof of the correctness of the iterative solution was published in [BL80]. The formulation and proof presented here is based on [BF01].

Chapter 8

The Torch Problem

In this chapter, we present a solution to a more general version of the torch problem in exercise 3.4. The generalisation is to consider an arbitrary number of people; the task is to get all the people across a bridge in the optimal time.

Specifically, the problem we discuss is the following.

> N people wish to cross a bridge. It is dark, and it is necessary to use a torch when crossing the bridge, but they only have one torch between them. The bridge is narrow and at most 2 people can be on it at any one time. The people are numbered from 1 thru N. Person i takes time $t.i$ to cross the bridge; when two cross together they must proceed at the speed of the slowest.
>
> Construct an algorithm that will get all N people across in the shortest time.

For simplicity, we assume that $t.i < t.j$ whenever $i < j$. (This means that we assume the people are ordered according to crossing time and that their crossing times are distinct. Assuming that the crossing times are distinct makes the arguments simpler, but is not essential. If the given times are such that $t.i = t.j$ for some i and j, where $i < j$, we can always consider pairs $(t.i\,,\,i)$, where i ranges over people, ordered lexicographically. Renaming the crossing "times" to be such pairs, we obtain a total ordering on times with the desired property.)

8.1 Lower and Upper Bounds

The derivation that follows is quite long and surprisingly difficult, particularly in comparison to the final algorithm, which is quite simple. It's important to appreciate where precisely the difficulties lie. This has to do with the difference between establishing an "upper bound" and a "lower bound" on the crossing times.

In the original problem given in chapter 1 , there are four people with crossing times of 1 minute, 2 minutes, 5 minutes and 10 minutes. Crucially, the question asked was to show that all four can cross the bridge *within* 17 minutes. In other words, the question asks for a so-called *upper bound* on the time taken. In general, an upper bound is established by exhibiting a sequence of crossings that takes the required time.

A much harder problem is to show that 17 minutes is a *lower bound* on the time taken. Showing that it is a lower bound means showing that the time can never be bettered.

We can use the same instance of the torch problem to further illustrate the difference between lower and upper bounds. Most of us, when confronted with the torch problem above, will first explore the solution in which the fastest person accompanies the others across the bridge. Such a solution takes a total time of $2+1+5+1+10$, i.e. 19 minutes. By exhibiting the crossing sequence, we have established that 19 minutes is an upper bound on the crossing time; we have *not* established that it is a lower bound. (Indeed, it is not.) Similarly, by exhibiting the crossing sequence that gets all four people across in 17 minutes does not prove that this time cannot be bettered. Doing so is much harder than just constructing the sequence.

In this chapter, the goal is to construct an algorithm for scheduling N people to cross the bridge. The algorithm we derive is quite simple but, on its own, it only establishes an upper bound on the optimal crossing time. The greatest effort goes into showing that the algorithm simultaneously establishes a lower bound on the crossing time. The combination of equal lower and upper bounds is called an *exact* bound; this is what is meant by an optimal solution.

In section 8.6, we present two algorithms for constructing an optimal sequence. The more efficient algorithm assumes a knowledge of algorithm development that goes beyond the material in this book.

8.2 Outline Strategy

Once again, the main issue we have to overcome is the avoidance of unnecessary detail. The problem asks for a *sequence* of crossings but there is an enormous amount of freedom in the order in which crossings are scheduled. It may be, for example, that the optimal solution is to let one person accompany all the others one-by-one across the bridge, each time returning with the torch for the next person. If our solution method requires that we detail in what order the people cross, then it is extremely ineffective. The number of different orderings is $(N-1)!$, which is a very large number even for quite small values of N.

The way to avoid unnecessary detail is to focus on what we call the “forward trips”.

Recall that, when crossing the bridge, the torch must always be carried. This means that crossings alternate between "forward"and "return" trips, where a *forward trip* is a crossing in the desired direction, and a *return trip* is a crossing in the opposite direction. Informally, the forward trips do the work whilst the return trips service the forward trips. The idea is that, if we can compute the optimal collection of forward trips, the return trips needed to sequence them correctly can be easily deduced.

In order to turn this idea into an effective solution, we need to proceed more formally. First, by the "collection" of forward trips, we mean a "bag" of sets of people. The mathematical notion of a "bag" (or "multiset" as it is sometimes called) is similar to a set but, whereas a set is defined solely by whether or not a value is an element of the set, a bag is defined by the number of times each value occurs in the set. For example, a bag of coloured marbles would be specified by saying how many red marbles are in the bag, how many blue marbles, and so on. We will write, for example, $\{1{*}a\,,2{*}b\,,0{*}c\}$ to denote a bag of as, bs and cs in which a occurs once, b occurs twice and c occurs no times. For brevity, we also write $\{1{*}a\,,2{*}b\}$ to denote the same bag.

It is important to stress that a bag is different from a sequence. Even though when we write down an expression denoting a bag we are forced to list the elements in a certain order (alphabetical order in $\{1{*}a\,,2{*}b\,,0{*}c\}$, for example), the order has no significance. The expressions $\{1{*}a\,,2{*}b\,,0{*}c\}$ and $\{2{*}b\,,1{*}a\,,0{*}c\}$ both denote the same bag.

A *trip* is given by the set of people involved in the trip. So, for example, $\{1,3\}$ is a trip in which persons 1 and 3 cross. If we are obliged to distinguish between forward and return trips, we prefix the trip with either "$+$" (for forward) or "$-$" (for return). So $+\{1,3\}$ denotes a forward trip made by persons 1 and 3 and $-\{2\}$ denotes a return trip made by person 2.

As we said above, our focus will be on computing the *bag* of forward trips in an optimal sequence of trips. We begin by establishing a number of properties of sequences of trips that allow us to do this.

We call a sequence of trips that gets everyone across in accordance with the rules a *valid sequence*. We will say that one valid sequence *subsumes* another valid sequence if the time taken by the first is at most the time taken for the second. Note that the subsumes relation is reflexive (every valid sequence subsumes itself) and transitive (if valid sequence a subsumes valid sequence b and valid sequence b subsumes valid sequence c then valid sequence a subsumes valid sequence c). The problem is to find a valid sequence that subsumes all valid sequences.

Formally, a *valid sequence* is a set of numbered trips with the following two properties:

- The trips are sets; each set has one or two elements, and the number given to a trip is its position in the sequence (where numbering begins from 1).

- Odd-numbered trips in the sequence are called forward trips; even-numbered trips are called return trips. The length of the sequence is odd.

- The trips made by each individual person alternate between forward and return trips, beginning and ending with forward trips. (A trip T is made by person i if $i{\in}T$.)

Immediate consequences of this definition which play a crucial role in finding an optimal sequence are:

- The number of forward trips is one more than the number of return trips.

- The number of forward trips made by each individual person is one more than the number of return trips made by that person.

A *regular forward trip* means a forward trip *made by two people*, and a *regular return trip* means a return trip *made by exactly one person*. A *regular sequence* is a valid sequence that consists entirely of regular forward and return trips.

The first step (lemma 8.1) is to show that every valid sequence is subsumed by one in which all trips are regular. The significance of this is threefold.

- The number of forward trips is $N{-}1$ and the number of return trips is $N{-}2$. (Recall that N is the number of people.)

- The time taken by a regular sequence can be evaluated knowing only which forward trips are made; not even the order in which they are made needs to be known. (Knowing the bag of forward trips, it is easy to determine how many times each person makes a return trip. This is because each person makes one fewer return trips than forward trips. In this way, the time taken for the return trips can be calculated.)

- Most importantly, knowing just the bag of forward trips in a regular sequence is sufficient to reconstruct a valid regular sequence. Since all such sequences take the same total time, we can thus replace the problem of finding an optimal sequence of forward and return trips by the problem of finding an optimal bag of forward trips

Finding an optimal bag of forward trips is then achieved by focusing on which people do not make a return trip. We prove the obvious property that, in an optimal solution, the two slowest people do not return. We can then use induction to determine the complete solution.

8.3 Regular Sequences

Recall that a "regular" sequence is a sequence in which each forward trip involves two people and each return trip involves one person. We can always restrict attention to regular sequences because of the following lemma.

Lemma 8.1 Every valid sequence containing irregular trips is subsumed by a strictly faster valid sequence without irregular trips.

Proof Suppose a given valid sequence contains irregular trips. We consider two cases: the first irregular trip is forward and the first irregular trip is backward.

If the first irregular trip is backward, choose an arbitrary person, p say, making the trip. Identify the forward trip made by p prior to the backward trip, and remove p from both trips. More formally, suppose the sequence has the form

$$u +\{p,q\}\ v\ -\{p,r\}\ w$$

where q and r are people, u, v and w are subsequences and p occurs nowhere in v. (Note that the forward trip made by p involves two people because it is assumed that the first irregular trip is backward.) Replace the sequence by

$$u +\{q\}\ v\ -\{r\}\ w$$

This results in a valid sequence, the time for which is no greater than the original sequence. (To check that the sequence remains valid, we have to check that the trips made by each individual continue to alternate between forward and return. This is true for individuals other than p because the points at which they cross remain unchanged, and it is true for p because the trips made by p have changed by the removal of consecutive forward and return trips. The time taken is no greater since, for any x and y, $t.p{\uparrow}x + t.p{\uparrow}y \ \geq\ x{+}y$.) The number of irregular crossings is not reduced, since a new irregular forward trip has been introduced, but the total number of person-trips *is* reduced.

Now suppose the first irregular trip is forward. There are two cases to consider: the irregular trip is the very first in the sequence, and it is not the very first.

If the first trip in the sequence is not regular, it means that one person crosses and then immediately returns. (We assume that N is at least 2.) These two crossings can be removed. Clearly, since times are positive, the total time taken is reduced. Also, the number of person-trips is reduced.

If the first irregular crossing is a forward trip but not the very first, let us suppose it is person p who crosses, and suppose q is the person who returns immediately before this forward trip. (There is only one such person because of the assumption that p's forward trip is the first irregular trip.) That is, suppose the sequence has the form

$$u\ -\{q\}\ +\{p\}\ v$$

Consider the latest crossing that precedes $q's$ return trip and involves p or q. There are two cases: it is a forward trip or it is a return trip.

If it is a forward trip, it must involve q and not p. Swap p with q in this trip and remove q's return trip and p's irregular crossing. That is, transform

$$u\ +\{q,r\}\ w\ -\{q\}\ +\{p\}\ v$$

(where w does not involve p or q) to

$$u\ +\{p,r\}\ w\ v\quad .$$

The result is a valid sequence. Moreover, the total crossing time is reduced (since, for any x, $t.q\uparrow x + t.q + t.p > t.p\uparrow x$), and the number of person-trips is also reduced.

If it is a return trip, it is made by one person only. (This is because we assume that $p's$ forward trip is the first irregular trip in the sequence.) That person must be p. Swap p with q in this return trip, and remove q's return trip and p's irregular crossing. That is, transform

$$u\ -\{p\}\ w\ -\{q\}\ +\{p\}\ v$$

(where w does not involve p or q) to

$$u\ -\{q\}\ w\ v\quad .$$

The result is a valid sequence. Moreover, the total crossing time is reduced (since, $t.p+t.q+t.p > t.q$), and the number of person-trips is also reduced.

We have now described how to transform a valid sequence that has at least one irregular crossing; the transformation has the effect of strictly decreasing the total time taken. Repeating this process whilst there are still irregular crossings is therefore guaranteed to terminate with a valid sequence that is regular, subsumes the given valid sequence and has a smaller person-trip count.
□

8.4 Sequencing Forward Trips

Lemma 8.1 has three significant corollaries. First, it means that the number of forward trips in an optimal sequence is $N-1$ and the number of return trips is $N-2$. This is because every subsequence comprising a forward trip followed by a return trip increases the number of people that have crossed by one, and the last trip increases the number by two. Thus, after the first $2\times(N-2)$ trips, $N-2$ people have crossed and 2 have not; so, after $2\times(N-2)+1$ trips everyone has crossed.

Second, it means that the total time taken to complete any regular sequence can be evaluated if only the bag of forward trips in the sequence is known; not even the order in which the trips are made is needed. This is because the bag of forward trips enables us to determine how many times each individual makes a forward trip. Hence, the number of times each individual returns can be computed, from which the total time for the return trips can be computed.

For example, suppose the forward trips in a regular sequence are as follows:

$$3*\{1,2\}\ ,\ 1*\{1,6\}\ ,\ 1*\{3,5\}\ ,\ 1*\{3,4\}\ ,\ 1*\{7,8\}$$

(The trips are seperated by commas; recall that $3*\{1,2\}$ means that persons 1 and 2 make 3 forward trips together, $1*\{1,6\}$ means that persons 1 and 6 make one forward trip together, etc. Note that no indication is given of the order in which the forward trips occur in the sequence.) Then, counting the number of occurrences of each person in the bag, person 1 makes 4 forward trips, and hence 3 return trips; similarly, person 2 makes 3 forward trips and hence 2 return trips, whilst person 3 makes 2 forward trips and hence 1 return trip. The remaining people (4, 5, 6, 7 and 8) all make 1 forward trip and, hence, no return trips. The total time taken is thus:

$$\begin{array}{rl} & 3\times(t.1\uparrow t.2)+1\times(t.1\uparrow t.6)+1\times(t.3\uparrow t.5)+1\times(t.3\uparrow t.4)+1\times(t.7\uparrow t.8)\\ + & 3\times t.1+2\times t.2+1\times t.3\end{array}$$

(The top line gives the time taken by the forward trips, and the bottom line gives the time taken by the return trips.) Note that the total number of forward trips is 7 (one less than the number of people), and the total number of return trips is 6.

The third important corollary of lemma 8.1 is that, given just the bag of forward trips corresponding to a regular sequence, it is possible to construct a regular sequence to get everyone across with the same collection of forward trips.

This is a non-obvious property of the forward trips and to prove that it is indeed the case we need to make some crucial observations.

Suppose F is a bag of forward trips corresponding to some regular sequence. That is, F is a collection of sets, each with exactly two elements and each having a certain multiplicity. The elements of the sets in F are people, which we identify with numbers in the range 1 thru N, and each number in the range must occur at least once. The number of times a person occurs is the number of forward trips made by that person.

We will call a person a *settler* if they make only one forward trip; we call a person a *nomad* if they make more than one forward trip. Division of people into these two types causes the trips in F to be divided into three types depending on the number of settlers in the trip. If both people in a trip are settlers we say the trip is *hard*; if one of the people is a settler and the other a nomad, we say the trip is *firm*; finally, if both people are nomads we say the trip is *soft*.

Now, suppose we use $\#nomad$, $\#hard$, $\#firm$ and $\#soft$ to denote the number of nomads, the number of hard trips, the number of firm trips and the number of soft trips, respectively, in the collection F. Then, the number of trips in F is

$$\#hard + \#firm + \#soft \quad .$$

The number of return trips equals the total number of forward trips made by individual nomads less the number of nomads, since each nomad makes one more forward trip than return trip. Since each soft trip contributes 2 to the number of forward trips made by nomads, and each firm trip contributes 1, the number of return trips is thus

$$2\times\#soft \ + \ \#firm \ - \ \#nomad \quad .$$

But the number of forward trips is one more than the number of return trips. That is,

$$\#hard + \#firm + \#soft \ = \ 2\times\#soft \ + \ \#firm \ - \ \#nomad + 1 \quad .$$

Equivalently,

$$\#hard + \#nomad \ = \ \#soft + 1 \quad .$$

We summarise these properties in the following definition.

Definition 8.2 (Regular Bag) Suppose N is at least 2. A bag of subsets of $\{i \mid 1 \le i \le N\}$ is called a *regular* N *bag* if it has the following properties:

- Each element of F has size two. (Informally, each trip in F involves two people.)
- $\langle\forall i \ : \ 1 \le i \le N \ : \ \langle\exists T : T\in F : i\in T\rangle\rangle$

  (Informally, every person is an element of at least one trip in F.)
- If $\#hard.F$, $\#soft.F$ and $\#nomad.F$ denote, respectively, the number of trips in F that involve two settlers, the number of trips in F that involve no settlers, and the number of nomads in F, then

  (8.3) $\quad \#hard.F \ + \ \#nomad.F \ = \ \#soft.F + 1 \quad .$

□

Formally, what we have proved is the following.

Lemma 8.4 Given a valid regular sequence of trips to get N people across, where N is at least 2, the bag of forward trips F that is obtained from the sequence by forgetting the numbering of the trips is a regular N bag. Moreover, the time taken by the sequence of trips can be evaluated from F. Let $\#_F T$ denote the multiplicity of T in F. Then, the time taken is

(8.5) $\quad \langle \Sigma T \; : \; T{\in}F \; : \; \langle \Uparrow i : i{\in}T : t.i\rangle \times \#_F T\rangle \; + \; \langle \Sigma i :: t.i \times r_F.i\rangle$

where

(8.6) $\quad r_F.i \; = \; \langle \Sigma T : T{\in}F \wedge i{\in}T : \#_F T\rangle - 1 \quad .$

($r_F.i$ is the number of times that person i returns.)

□

Immediate consequences of (8.3) are:

(8.7) $\quad \#nomad.F = 0 \;\; \Rightarrow \;\; \#hard.F = 1 \wedge \#firm.F = 0 = \#soft.F \quad .$

If $\#nomad.F$ is zero, there are no nomads and hence, by definition, no soft or firm trips. So, by (8.3), $\#hard.F$ is 1.

(8.8) $\quad \#nomad.F = 1 \;\; \equiv \;\; \#hard.F = 0 = \#soft.F \quad .$

If there is only one nomad, there are no trips involving two nomads. That is $\#soft.F$ is zero. It follows from (8.3) that $\#hard.F$ also equals zero. The converse is immediate from (8.3).

(8.9) $\quad N{>}2 \wedge \#hard.F \geq 1 \;\; \Rightarrow \;\; \#soft.F \geq 1 \quad .$

If N (the number of people) is greater than 2, not all can cross at once and, so, $\#nomad.F$ is at least 1. It follows from (8.3) that $\#soft.F$ is at least 1.

Now we can show how to construct a regular sequence from F.

Lemma 8.10 G iven N (at least 2) and a regular N bag, a valid regular sequence of trips can be constructed from F to get N people across. The time taken by the sequence is given by (8.5).

Proof The proof is by induction on the size of the bag F.

We need to consider three cases. The easiest case is when F consists of exactly one trip (with multiplicity one). The sequence is then just this one trip.

The second case is also easy. It is the case that $\#nomad.F = 1$. In this case, every trip in F is firm. That is, each trip has the form $\{n{,}s\}$ where n is the nomad and s is a settler. The sequence is simply obtained by listing all the elements of F in an arbitrary order and inserting a return trip by the nomad n in between each pair of forward trips of which the first is the trip $\{n,s\}$ for some s.

The third case is that $\#hard.F$ is non-zero and F has more than one trip. In this case, by (8.9), $\#soft.F$ is at least 1. It follows, by definition of $soft$ that $\#nomad.F$ is at least 2. Choose any soft trip in F. Suppose it is $\{n,m\}$ where n and m are both nomads. Construct the sequence which begins with the trip $\{n,m\}$ and is followed by the return of n, then an arbitrary hard trip and then the return of m. Reduce the multiplicity of the chosen hard trip and the chosen soft trip in F by 1. (That is, remove one occurrence of each from F.) We get a new bag F' in which the number of trips made by each of n and m has been reduced by 1 and the number of people has been reduced by 2. By induction, there is a regular sequence corresponding to F' which gets the remaining people across.
□

Optimisation Problem Lemmas 8.4 and 8.10 have a significant impact on how to solve the general case of the bridge problem. Instead of seeking a sequence of crossings of optimal duration, we seek a regular bag as defined in definition 8.2 that optimises the time given by (8.5). It is this problem that we now solve.

In solving this problem, it is useful to introduce some terminology when discussing the time taken as given by (8.5). There are two summands in this formula, The value of the first summand we call F's *total forward time* and the value of the second summand F's *total return time*. Given a bag F and a trip T in F, we call $\langle\Uparrow i : i{\in}T : t.i\rangle \times \#_F T$ the *forward time of* T *in* F. (Sometimes "in F" is omitted if this is clear from the context.) For each person i, we call the value of $t.i \times r_F.i$ the *return time of person* i (or person i's return time).

8.5 Choosing Settlers and Nomads

This section is about how to choose settlers and nomads. We establish that the settlers are the slowest people and the nomads are the fastest. More specifically, we establish that in an optimal solution there are at most 2 nomads. Person 1 is always a nomad if N is greater than 2; additionally, person 2 may also be a nomad.

Lemma 8.11 Every regular bag is subsumed by a regular bag for which all settlers are slower than all nomads.

Proof Suppose the regular N bag F is given. Call a pair of people (p,q) an *inversion* if, within F, p is a settler, q is a nomad and p is faster than q.

Choose any inversion (p, q). Suppose q and p are interchanged everywhere in F. We get a regular N bag. Moreover, the return time is clearly reduced by at least $t.q - t.p$.

The forward times for the trips not involving p or q are, of course, unchanged. The forward time for the trips originally involving q are not increased (because $t.p < t.q$). The forward time for the *one* trip originally involving p is increased by an amount that is at most $t.q - t.p$. This is verified by considering two cases. The first case is when the trip involving p is $\{p,q\}$. In this case, swapping p and q has no effect on the trip, and the increase in time taken is 0. In the second case, the trip involving p is $\{p,r\}$ where $r \neq q$. In this case, it suffices to observe that

$$
\begin{array}{ll}
 & t.p \uparrow t.r \;+\; (t.q - t.p) \\
= & \{ \quad \text{distributivity of sum over max, arithmetic} \quad \} \\
 & t.q \uparrow (t.r + (t.q - t.p)) \\
\geq & \{ \quad t.p \leq t.q \text{, monotonicity of max} \quad \} \\
 & t.q \uparrow t.r \quad .
\end{array}
$$

Finally, the times for all other forward trips are unchanged.

The net effect is that the total time taken does not increase. That is, the transformed bag subsumes the original bag. Also, the number of inversions is decreased by at least one. Thus, by repeating the process of identifying and eliminating inversions, a bag F is obtained that subsumes the given bag.
□

Corollary 8.12 Every regular N bag is subsumed by a regular N bag F with the following properties:

- In any firm trip in F, the nomad is person 1.
- Every soft trip in F is $\{1,2\}$. (Note: the multiplicity of this trip in the bag can be an arbitrary number, including 0.)
- The multiplicity of $\{1,2\}$ in F is j, for some j where $1 \leq j \leq N \div 2$, and the hard trips are $\{k: 0 \leq k < j-1: \{N - 2\times k\ ,\ N - 2\times k - 1\}\}$. (Note that this is the empty set when j equals 1.)

Proof Suppose F is a regular N bag that optimises the total travel time. From 8.11, we may assume that the nomads are slower than the settlers.

Suppose there is a firm trip in F in which the nomad is person i where i is not 1. Replace i in one occurrence of the trip by person 1. This has no effect on the forward time, since i is slower than the other person in the trip. However, the total return time is reduced (by $t.i - t.1$). We claim that this results in a regular N bag, which contradicts F being optimal. (Please refer to lemma 8.4 for the properties required of a regular bag.)

Of course, the size of each set in F is unchanged. The number of trips made by i decreases by one, but remains positive because i is a nomad in F. The number of trips made by all other persons either remains unchanged or, in the case of person 1, increases. So it remains to check property (8.3). If person i is still a nomad after the replacement, property (8.3) is maintained because the type (hard, firm or soft) of each trip remains unchanged. However, person i may become a settler. That is, the number of nomads may be decreased by the replacement. If so, person i is an element of two trips in F. The second trip is either a firm trip in F and becomes a hard trip, or it is a soft trip in F and becomes firm. In both cases, it is easy to check that (8.3) is maintained.

Now suppose there is a soft trip in F different from $\{1,2\}$. A similar argument to the one above shows that replacing the trip by $\{1,2\}$ results in a regular bag with a strictly smaller total travel time, contradicting F being optimal.

We may now conclude from (8.3) that either there are no soft trips or the multiplicity of $\{1,2\}$ in F is j, for some j where j is at least 2, and there are $j-1$ hard trips. When there are no soft trips, all the trips are firm or hard. But, as we have shown, person 1 is the only nomad in firm trips and there are no nomads in hard trips; it follows that person 1 is the only nomad in F and, from (8.3), that there are no hard trips. Thus persons 1 and 2 are the elements of one (firm) trip in F.

It remains to show that, when j is at least 2, the hard trips form the set

$$\{k : 0 \le k < j-1 : \{N - 2\times k \ , \ N - 2\times k - 1\}\} \ .$$

(The multiplicity of each hard trip is 1, so we can ignore the distinction between bags and sets.)

Assume that the number of soft trips is j where j is at least two. Then the settlers are persons 3 thru N, and $2\times(j-1)$ of them are elements of hard trips, the remaining $N - 2\times(j-2)$ being elements of firm trips. Any regular bad clearly remains regular under any permutation of the settlers. So we have to show that choosing the settlers so that the hard trips are filled in order of slowness gives the optimal arrangement. This is done by induction on the number of settlers in the hard trips.
□

8.6 The Algorithm

We can now solve the problem.

Lemma 8.13 Suppose F is an optimal solution satisfying the properties listed in corollary 8.12. Then, if j is the multiplicity of $\{1,2\}$ in F, the total time taken by F is

$$(8.14)\quad HF.j \;+\; FF.j \;+\; j\times t.2 \;+\; (N{-}j{-}1)\times t.1 \;+\; (j{-}1)\times t.2 \quad ,$$

where

$$HF.j \;=\; \langle \Sigma i \;:\; 0\leq i<j{-}1 \;:\; t.(N{-}2i)\rangle \quad \text{, and}$$

$$FF.j \;=\; \langle \Sigma i \;:\; 3\leq i\leq N-2\times(j{-}1) \;:\; t.i\rangle \quad .$$

(The first three terms give the forward times, and the last two terms give the return times.)

Proof There are two cases to consider. If there are no soft trips, the value of j is 1. In this case, the total time taken is

$$\langle \Sigma i{:}2\leq i\leq N{:}t.i\rangle \;+\; (N{-}2)\times t.1 \quad .$$

But

$$\begin{array}{ll}
 & HF.1 \;+\; FF.1 \;+\; 1\times t.2 \;+\; (N{-}1{-}1)\times t.1 \;+\; (1{-}1)\times t.2 \\
= & \qquad \{ \qquad \text{definition of } HF \text{ and } FF\text{, arithmetic} \quad \} \\
 & 0 \;+\; \langle \Sigma i{:}3\leq i\leq N{:}t.i\rangle \;+\; t.2 \;+\; (N{-}2)\times t.1 \\
= & \qquad \{ \qquad \text{arithmetic} \quad \} \\
 & \langle \Sigma i{:}2\leq i\leq N{:}t.i\rangle \;+\; (N{-}2)\times t.1 \quad .
\end{array}$$

If there are soft trips, the value of j is equal to the number of soft trips and is at least 2. In this case, $HF.j$ is the forward time for the hard trips in F and $FF.j$ is the forward time for the firm trips in F. Also, $j\times t.2$ is the forward time for the j soft trips. Finally, person 1's return time is $(N{-}j{-}1)\times t.1$ and person 2's return time is $(j{-}1)\times t.2$. (Person 2 is an element of j forward trips, and person 1 is an element of $j+(N-2\times(j{-}1)-3+1)$ forward trips. Note that the sum of $j{-}1$ and $N{-}j{-}1$ is $N{-}2$, which is what we expect the number of return trips to be.)
□

For all j, where j is at least 2, define $OT.j$ to be the optimal time taken by a regular N bag where the multiplicity of $\{1,2\}$ in the bag is j. That is, $OT.j$ is given by (8.14). Now, we observe that

$$HF.(j{+}1) - HF.j \;=\; t.(N{-}2j{+}2) \quad ,$$

and

$$FF.(j{+}1) - FF.j \;=\; -(t.(N{-}2j{+}2) + t.(N{-}2j{+}1)) \quad .$$

As a consequence,

$$OT.(j{+}1) - OT.j \;=\; -t.(N{-}2j{+}1) + 2\times t.2 - t.1 \quad .$$

Note that

$$\begin{array}{ll}
 & OT.(j{+}1) - OT.j \;\le\; OT.(k{+}1) - OT.k \\
= & \qquad \{ \qquad \text{above} \quad \} \\
 & -t.(N{-}2j{+}1) + 2\times t.2 - t.1 \;\le\; -t.(N{-}2k{+}1) + 2\times t.2 - t.1 \\
= & \qquad \{ \qquad \text{arithmetic} \quad \} \\
 & t.(N{-}2k{+}1) \;\le\; t.(N{-}2j{+}1) \\
= & \qquad \{ \qquad t \text{ is increasing} \quad \} \\
 & j \le k \quad .
\end{array}$$

That is, $OT.(j{+}1) - OT.j$ *increases* as j *increases*. A consequence is that the minimum value of $OT.j$ can be determined by a search for the point at which the difference function changes from being negative to being positive.

The simplest way to do this and simultaneously construct a regular sequence to get all N people across is to use a linear search, beginning with j assigned to 1. At each iteration the test $2\times t.2 \le t.1 + t.(N{-}2j{+}1)$ is performed. If it evaluates to $true$, the soft trip $\{1,2\}$ is scheduled; this is followed by the return of person 1, then the hard trip $\{N{-}2j{+}2\;,\;N{-}2j{+}1\}$ and then the return of person 2. If the test evaluates to false, the remaining $N{-}2j{+}2$ people are scheduled to cross in $N{-}2j{+}1$ firm trips. In each trip one person crosses accompanied by person 1; in between two such trips person 1 returns.

When the number N is large, the number of tests can be reduced by using binary search to determine the optimal value of j. This is encoded as follows.

```
$\{ \;\; 2 \le N \;\; \}$
$i,j \;:=\; 1\,,N{\div}2 \;\;;$
$\{$ **Invariant:**
        $1 \le i \le j \le N{\div}2$
    $\wedge \;\; \langle \forall k : 1 \le k < i : OT.(k{+}1) \le OT.k \rangle$
```

```
        $\wedge\ \ \langle\forall k : j \le k < N\div 2 : OT.(k{+}1) \ge OT.k\rangle\ \ \}$
do $i<j$ →     $m := (i{+}j)\div 2$ ;
               if   $2\times t.2\ \le\ t.1 + t.(N{-}2m{+}1)$ → $i := m$
               □   $2\times t.2\ \ge\ t.1 + t.(N{-}2m{+}1)$ → $j := m$
               fi
od
{       $1 \le j \le N\div 2$
    $\wedge\ \ \langle\forall k : 1 \le k < j : OT.(k{+}1) \le OT.k\rangle$
    $\wedge\ \ \langle\forall k : j \le k < N\div 2 : OT.(k{+}1) \ge OT.k\rangle\ \ \}$ .
```

On termination, j is the multiplicity of $\{1,2\}$ in an optimal regular bag and $OT.j$ is the required optimal time. Thus, $j-1$ is the number of hard trips. Corollary 8.12 specifies the composition of the hard trips and lemma 8.10 specifies how they are scheduled. The remaining $N-2j+2$ people are then scheduled to cross in $N-2j+1$ firm trips as described above.

8.7 Conclusion

In this chapter, we have presented an algorithm to solve the torch problem for an arbitrary number of people and arbitrary individual crossing times. The greatest challenge in an optimisation problem of this nature is to establish without doubt that the algorithm constructs a solution that can not be bettered. A major step in solving the problem was to eliminate the need to consider *sequences* of crossings and to focus on the bag of forward trips. Via a number of lemmas, we established a number of properties of an optimum bag of forward trips which then enabled us to construct the required algorithm.

Many of the properties we proved are not surprising. An optimal sequence is "regular" — that is, each forward trip is made by two people and each return trip is made by one; the "settlers" (the people who never make a return trip) are the slowest, and the "nomads" (the people who do make return trips) are the fastest. Less obvious is that there are at most two nomads and the number of "hard" trips (trips made by two settlers) is one less than the number of trips that the two fastest people make together. The proof of the fact that there are at most two nomads is made particularly easy by the focus on the bag of forward trips; if we had had to reason about the sequence of trips, this property could have been very difficult to establish.

Even though these properties may seem unsurprising and the final algorithm (in retrospect) perhaps even "obvious", it is important to appreciate that proof is required

— the interest in the torch problem is that the most “obvious” solution (letting the fastest person accompany everyone else across the bridge) is not always the best solution. Always beware of claims that something is “obvious”.

8.8 Bibliographic Remarks

In American-English, the torch problem is called the “flashlight problem”; it is also sometimes called the “bridge problem”. The solution presented here is based on a solution to the yet-more-general torch problem in which the capacity of the bridge is a parameter [Bac07]. In terms of this more general problem, this chapter is about the capacity 2 problem; the solution is essentially the same as a solution given by Rote [Rot02].

Rote describes the solution in terms of “multigraphs” rather than bags. This difference is superficial. Each edge of a “multigraph” connects two people and, hence, is just a set of two people. The main difference is lemma 8.10; Rote does not prove that any regular bag can be converted into a sequence of crossings. He does claim that this is indeed the case, but only proves that it is possible for the regular bags corresponding to optimal crossings.

Rote describes the linear-search method of determining the optimal solution, and doesn’t suggest using binary search. (Strictly, at one point, his account is incorrect. He says “the optimal value ... can be determined easily by locating the value $2t_2 - t_1$ in the sorted list of t_i’s” but, of course, $2t_2 - t_1$ might not occur in the list — the standard 1, 2, 5 and 10 minute problem is an example!)

Rote gives a comprehensive bibliography including pointing out one publication where the algorithm is incorrect. When the capacity of the bridge is also a parameter, the problem is much harder to solve. Many “obvious” properties of an optimal solution turn out to be false. For example, it is no longer the case that an optimal solution uses a minimum number of crossings. (If $N=5$, the capacity of the bridge is 3 and the travel times are 1, 1, 4, 4 and 4, the shortest time is 8, which is achieved using 5 crossings. The shortest time using 3 crossings is 9.) The notion of “regularity” of crossings has to be generalised in a way that allows for some forward trips not to be “full” (in the sense that the full capacity of the bridge is not utilised). Rote’s formulation of the solution in terms of “multigraphs” does not appear to generalise, in contrast to the use of bags which does.

Chapter 9

Knight's Circuit

The problem tackled in this chapter is a particularly hard one. Yet, by suitably decomposing the problem, combined with effective reasoning skills, the problem becomes solvable.

The problem is to find a Knight's *circuit* of a chessboard. That is, find a sequence of moves that will take a Knight in a circuit around all the squares of a chessboard, visiting each square exactly once, and ending at the square at which the circuit began.

The problem is an instance of a search problem; in principle, it can be solved by a systematic, exhaustive examination of all the paths a Knight can follow around a chessboard — a so-called *brute-force* search. However, there are 64 squares on a chessboard; that means 64 moves have to be chosen, one for each square. From each of the corner squares, there is a choice of just 2 moves, but from each of the 16 central squares, there is a choice of 8 moves (see fig. 9.1); from the remaining squares either 4 or 6 moves are possible. This gives a massive amount of choice in the paths that can be followed. Lots of choice is usually not a problem but, when combined with the very restrictive requirements that the path forms a circuit that visits every square exactly once, it does become a problem. The Knight's circuit problem is hard because of this critical combination of an *explosion* with an *implosion* of choice.

But, all is not lost. The squares on a chessboard are arranged in a very simple pattern, and the Knight's moves, although many, are specified by one simple rule (two squares horizontally or vertically, and one square in the opposite direction). There is a great deal of structure, which we must endeavour to exploit.

9.1 Straight-Move Circuits

Finding a Knight's circuit is too difficult to tackle head on. Some experience of tackling simpler circuit problems is demanded.

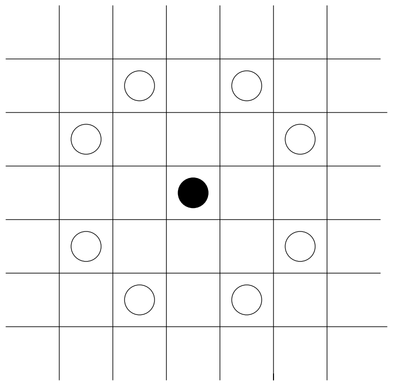


Figure 9.1: Knight's Moves.

Let's turn the problem on its head. Suppose you want to make a circuit of a chessboard and you are allowed to choose a set of moves that you are allowed to make. What sort of moves would you choose?

The obvious first answer is to allow moves from any square to any other square. In that case, it's always possible to construct a circuit of any board, whatever its size — starting from any square, choose a move to a square that has not yet been visited until all the squares are exhausted; then return to the starting square. But that is just too easy. Let's consider choosing from a restricted set of moves.

The simplest move is one square horizontally or vertically. (These are the moves that a King can make, but excluding diagonal moves.) We call these moves *straight* moves. Is it possible to make a circuit of a chessboard just with straight moves?

The answer is yes, although it isn't immediately obvious. You may be able to find a straight-move circuit by trial and error, but let us try to find one more systematically. As is often the case, it is easier to solve a more general problem; rather than restrict the problem to an 8×8 board, let us consider an arbitrary rectangular board. Assuming each move is by one square only, to the left or right, or up or down, is it possible to complete a straight-move circuit of the entire board? That is, is it possible to visit every square exactly once, beginning and ending at the same square, making "straight" moves at each step?

In order to gain some familiarity with the problem, please tackle the following exercise. Its solution is relatively straightforward.

Exercise 9.1

(a) What is the relation between the number of moves needed to complete a circuit of the board and the number of squares? Use your answer to show that it is impossible to complete a circuit of the board if both sides have odd length. (Hint: crucial is that each move is from one square to a different coloured square. Otherwise, the answer does not depend on the sort of moves that are allowed.)

(b) For what values of m is it possible to complete a straight-move circuit of a board of size $2m{\times}1$? (A $2m{\times}1$ board has one column of squares; the number of squares is $2m$.)

(c) Show that it is possible to complete a straight-move circuit of a $2{\times}n$ board for all (positive) values of n. (A $2{\times}n$ board has two rows, each row having n squares.)

□

The conclusion of exercise 9.1 is that a straight-move circuit is only possible if the board has size $2m{\times}n$, for positive numbers m and n. That is, one side has length $2m$ and the other has length n. (Both m and n must be non-zero because the problem assumes the existence of at least one starting square.) Also, a straight-move circuit can always be completed when the board has size $2{\times}n$, for positive n. This suggests that we now try to construct a straight-move circuit of a $2m{\times}n$ board, for m at least one and n greater than one, by induction on m, the $2{\times}n$ case providing the basis for the inductive construction.

To complete the inductive construction, we need to consider a board of size $2m \times n$, where m is greater than 1. Such a construction is hopeful because, when m is greater than 1, a $2m \times n$ board can be split into two boards of sizes $2p \times n$ and $2q \times n$, say, where both p and q are smaller than m and $p{+}q$ equals m. We may take as the inductive hypothesis that a straight-move circuit of both boards can be constructed. We just need to combine the two constructions.

The key to the combination is the corner squares. There are two straight moves from each of the corner squares, and any straight-move circuit must use both. In particular, it must use the horizontal moves. Now, imagine that a $2m \times n$ board is divided into a $2p \times n$ board and a $2q \times n$ board, with the former above the latter. (The convention we use is that the first number gives the number of rows and the second the nunber of columns of the board.) The bottom-left corner of the $2p \times n$ board is immediately above the top-left corner of the $2q \times n$ board. Construct straight-move circuits of these two boards. Figure 9.2 shows the result diagrammatically. The two horizontal, red lines at the middle-left of the diagram depict the horizontal moves that we know must form part of the two circuits. The blue dotted lines depict the rest of the circuits. (Of course, the shape of the dotted lines gives no indication of the shape of the circuit that is constructed.)

Now, to combine the circuits to form one circuit of the entire board, replace the horizontal moves from the bottom-left and top-left corners by vertical moves, as shown by the vertical green lines in fig. 9.3. This completes the construction.

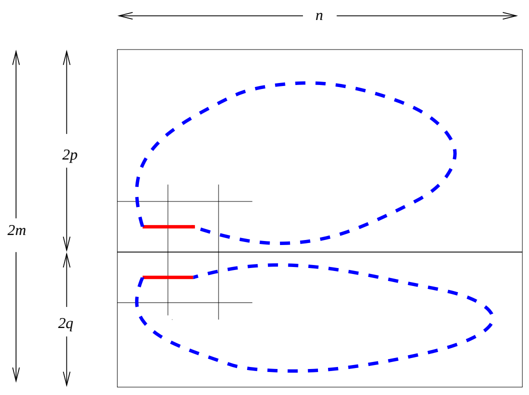


Figure 9.2: Combining straight-move circuits. First, split the board into two smaller boards and construct straight-move circuits of each.

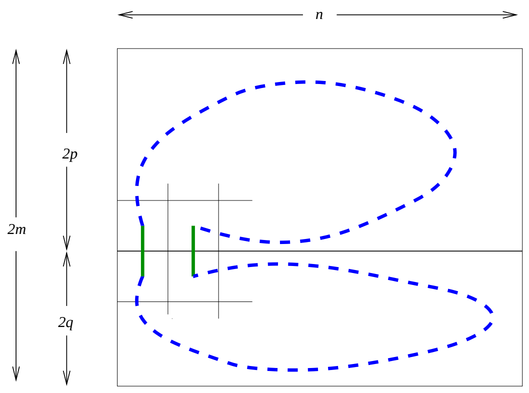


Figure 9.3: Second, combine the two circuits as shown.

Figure 9.4 shows the circuit that is constructed in this way for a 6×8 board. Effectively, the basis of the inductive algorithm constructs straight-move circuits of three 2×8 boards. The induction step then combines them by replacing horizontal moves by the green vertical moves shown in fig. 9.4.

Exercise 9.2 As mentioned above, when a board has an odd number of squares, no circuit is possible.

Consider a 3×3 board. It is easy to construct a straight-move circuit of all its squares but the middle square. (See fig. 9.5.) It is also possible to construct a straight-move circuit of all its squares but one of the corner squares. However, a straight-move circuit of all but one of the other four squares —the squares adjacent to a corner square, for example, the middle-left square— cannot be constructed.

Explore when it is possible, and when it is not possible, to construct a straight-move circuit of all the squares but one in a board of odd size.

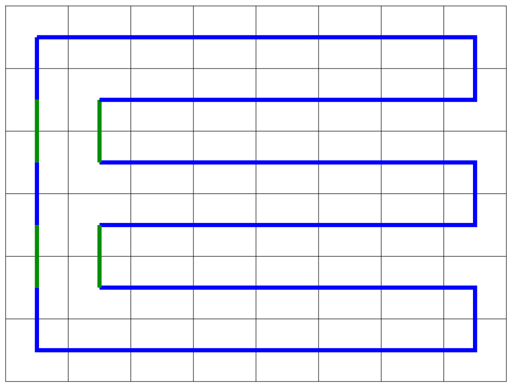


Figure 9.4: A straight-move circuit for a 6×8 board.

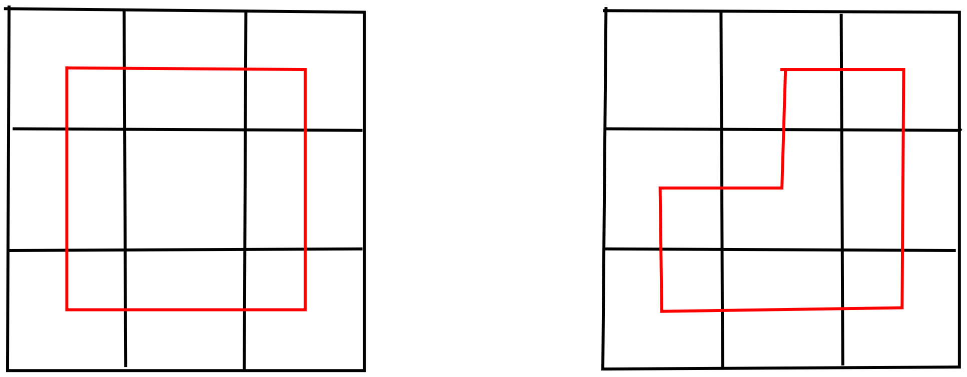


Figure 9.5: Straight-move circuits (shown in red) of a 3×3 board, omitting one of the squares.

□

9.2 Supersquares

Let us now return to the Knight's-circuit problem. The key to a solution is to exploit what we know about straight moves. The way this is done is to imagine that the 8×8 chessboard is divided into a 4×4 board by grouping together 2×2 squares into "supersquares", as shown in fig. 9.6.

If this is done, the Knight's moves can be classified into two types: *Straight moves* are moves that are "straight" with respect to the supersquares; that is, a Knight's move is *straight* if it takes it from one supersquare to another supersquare either vertically above or below, or horizontally to the left or to the right. *Diagonal moves* are moves that are not straight with respect to the supersquares; a move is *diagonal* if it takes the

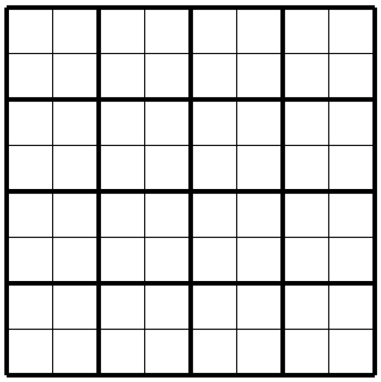


Figure 9.6: Chessboard divided into a 4×4 board of supersquares.

Knight from one supersquare to another along one of the diagonals through the starting supersquare. In fig. 9.7, the boundaries of the supersquares are indicated by thickened lines; the starting position of the Knight is shown in black, the straight moves are to the blue positions, and the diagonal moves are to the red positions.

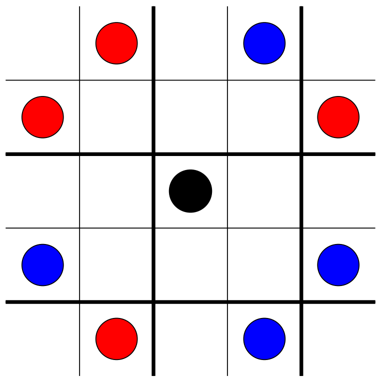


Figure 9.7: Straight (Blue) and Diagonal (Red) Knight's Moves From Some Starting Position (Black). Boundaries of the supersquares are indicated by thickened lines.

Focusing on the straight moves, we now make a crucial observation. Figure 9.8 shows the straight moves from one supersquare —the bottom-left supersquare— vertically upwards and horizontally rightwards. The colours indicate the moves that can be made. For example, from the bottom-left red square a straight move can be made to the top-left red square or to the bottom-right red square.

Observe the pattern of the colours. Vertical moves flip the colours around a vertical axis, whilst horizontal moves flip them around a horizontal axis. (The vertical moves interchange the red-above-yellow and blue-above-green columns; the horizontal moves interchange the red-next-to-blue row with the yellow-next-to-green row.)

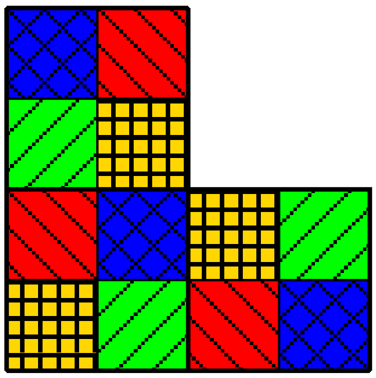


Figure 9.8: Straight moves from the bottom-left supersquare.

Let us denote by v the operation of flipping the columns of a 2×2 square (v is short for "vertical"). Similarly, let us denote by h (short for "horizontal") the operation of flipping the rows of the square. Now, let an infix semicolon denote doing one operation after another. So, for example, $v;h$ denotes the operation of first flipping the columns and then flipping the rows of the square. Flipping the columns and then flipping the rows is the same as flipping the rows and then the columns. That is,

$$v;h \;=\; h;v \quad .$$

Both are equivalent to rotating the 2×2 square through 180^0 about its centre. So, let us use c (short for "centre") as its name. That is, by definition of c,

(9.3) $\quad v;h \;=\; c \;=\; h;v \quad .$

We have now identified three operations on a 2×2 square. There is a fourth operation, which is the do-nothing operation. Elsewhere, we have used skip to name such an operation. Here, for greater brevity we use n (short for "no change"). Flipping twice vertically, or twice horizontally, or rotating twice through 180^0 about the centre, all amount to doing nothing. That is;

(9.4) $\quad v;v \;=\; h;h \;=\; c;c \;=\; n \quad .$

Also, doing nothing before or after any operation is the same as doing the operation.

(9.5) $\quad n;x \;=\; x;n \;=\; x \quad .$

The three equations (9.3), (9.4) and (9.5), together with the fact that doing one operation after another is associative (that is, doing one operation x followed by two operations y and z in turn is the same as doing first x followed by y and then concluding with z — in symbols, $x;(y;z) = (x;y);z$) allow the simplification of any sequence of the operations to one operation. For example,

$$
\begin{array}{ll}
 & v;c;h \\
= & \{ \quad v;h = c \quad \} \\
 & v;v;h;h \\
= & \{ \quad v;v = h;h = n \quad \} \\
 & n;n \\
= & \{ \quad n;x = x\text{, with } x{:=}n \quad \} \\
 & n \ .
\end{array}
$$

In words, flipping vertically, then rotating through 180^0 about the centre, and then flipping horizontally is the same as doing nothing. (Note how associativity is used implicitly between the first and second steps. The use of an infix operator for "followed by" facilitates this all-important calculational technique.)

Exercise 9.6 Construct a two-dimensional table that shows the effect of executing two operations x and y in turn. The table should have four rows and four columns, each labelled by one of n, v, h and c. (Use the physical process of flipping squares to construct the entries.)

Use the table to verify that, for x and y in the set $\{n,v,h,c\}$,

(9.7) $x;y = y;x$.

Check also that, for x and y in the set $\{n,v,h,c\}$,

(9.8) $x;(y;z) = (x;y);z$.

(In principle, you need to consider 4^3, i.e. 64, different combinations. Think of ways to reduce the amount of work.)

□

Exercise 9.9 Two other operations that can be done on a 2×2 square are to rotate it about the centre through $90°$, in one case clockwise and in the other anticlockwise. Let r denote the clockwise rotation and let a denote the anticlockwise rotation. Construct a table that shows the effect of performing any two of the operations n, r, a or c in sequence.

Identify a complete set of operations on a 2×2 square and extend your solution to exercise 9.6 so that it is possible to determine the effect of composing any pair of operations. (Avoid constructing the complete table because it is quite large!)

□

9.3 Partitioning the Board

The identification of the four operations on supersquares is a significant step towards solving the Knight's-circuit problem. Suppose one of the supersquares is labelled "n". Then the remaining fifteen supersquares can be uniquely labelled as n, v, h or c squares, depending on their position relative to the starting square. Figure 9.9 shows how this is done. Suppose we agree that the bottom-left square is an "n" square. Then immediately above it is a "v" square, to the right of it is an "h" square, and diagonally adjacent to it is a "c" square. Supersquares further away are labelled using the rules for composing the operations.

As a consequence, all 64 squares of the chessboard are split into four disjoint sets. In fig. 9.9, the different sets are easily identified by the colour of the square. Two squares have the same colour equivales they can be reached from each other by straight Knight's moves. (That is, two squares of the same colour *can* be reached from each other by straight Knight's moves, and two squares of different colour *cannot* be reached from each other by straight Knight's moves.)

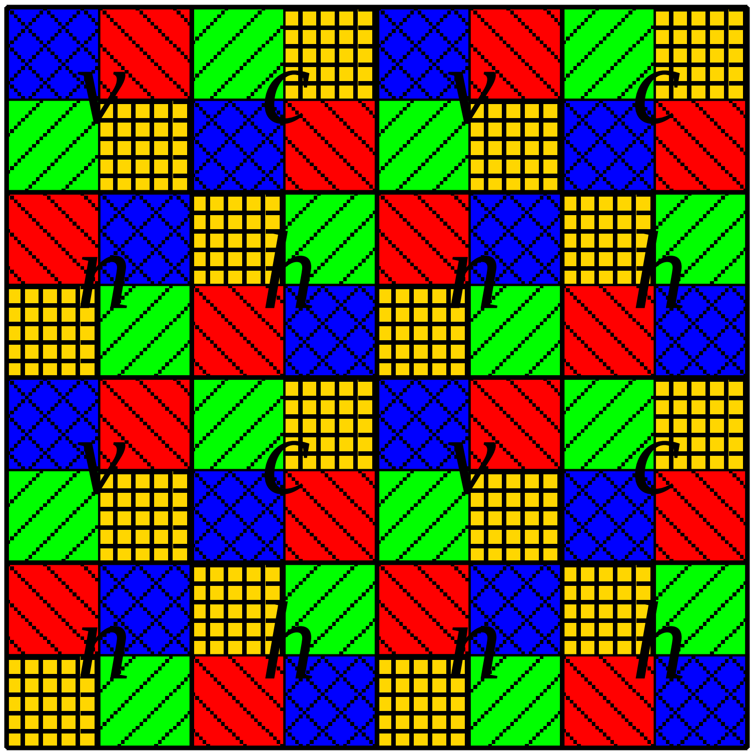


Figure 9.9: Labelling Supersquares

Recall the discussion of straight-move circuits in section 9.1. There we established the simple fact that it is possible to construct a straight-move circuit of a board of which one side has even length and the other side has length at least two. In particular, we can construct a straight-move circuit of a 4×4 board.

Now, a "straight" Knight's move is "straight" with respect to the supersquares of a chessboard. That means we can construct straight-move circuits of each of the four sets of squares on the chessboard. In fig. 9.9, this means constructing a circuit of all the red squares, a circuit of all the green squares, a circuit of all the blue squares, and a circuit of all the yellow squares.

We now have four *disjoint* circuits that together visit all the squares of the chessboard. The final step is to combine the circuits into one. The way to do this is to exploit the "diagonal" Knight's moves. (Refer back to fig. 9.7 for the meaning of "diagonal" in this context.)

A simple way of combining the four circuits is to combine them in pairs, and then combine the two pairs. For example, we can combine the red and blue circuits into a "red-blue" circuit of half the board; similarly, we can combine the green and yellow circuits into a "green-yellow" circuit. Finally, by combining the red-blue and green-yellow circuits, a complete circuit of the board is obtained.

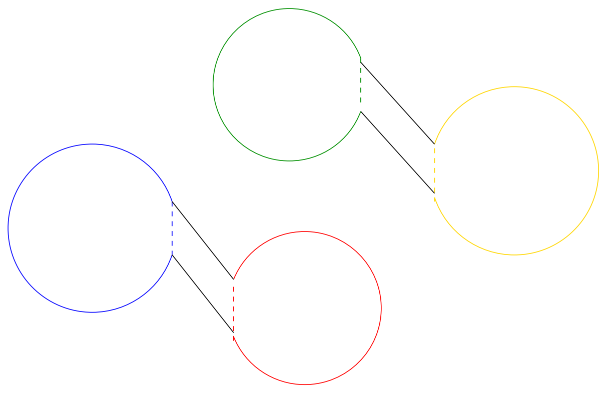


Figure 9.10: Schema for forming "red-blue" and "green-yellow" circuits. The straight-move circuits are depicted as coloured circles, a single move in each being depicted by a dotted line. These straight moves are replaced by the diagonal moves, shown as solid lines.

Figure 9.10 shows schematically how red-blue and green-yellow circuits are formed; in each case, two straight moves (depicted by dotted lines) in the respective circuits are replaced by diagonal moves (depicted by straight lines). Figure 9.11 shows one way of choosing the straight and diagonal moves in order to combine red and blue circuits, and green and yellow circuits — in each case, two "parallel" straight moves are replaced by two "parallel" diagonal moves. Exploiting symmetry, it is easy to find similar "parallel moves" with which to combine red and yellow circuits, or green and blue circuits. On the other hand, there are no diagonal moves from red to green, or from yellow to blue squares; consequently, it is impossible to construct a "red-green" or a "yellow-blue" circuit.

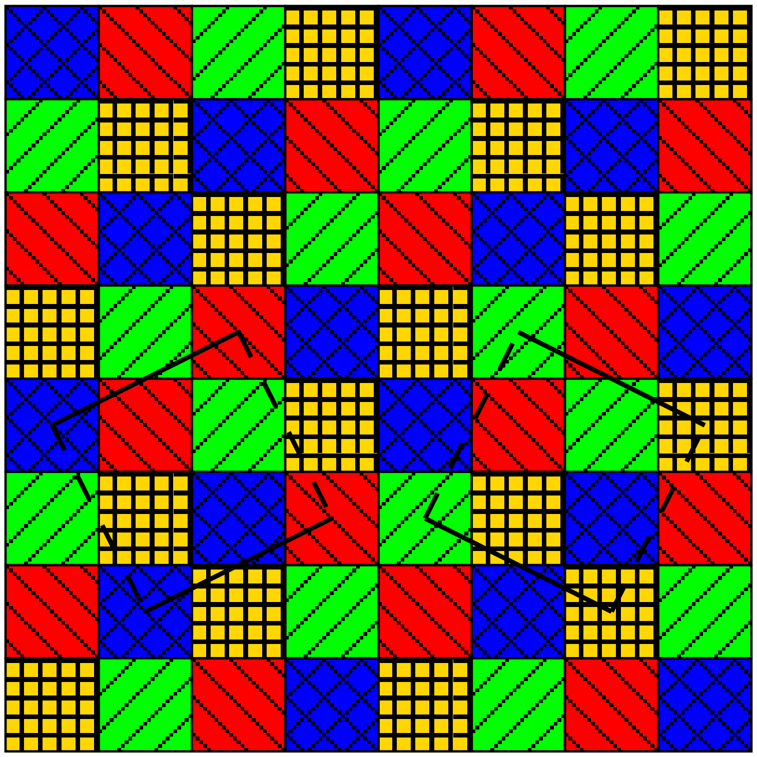


Figure 9.11: Forming "red-blue" and "green-yellow" circuits. The straight moves shown as dotted lines are replaced by the diagonal moves shown as solid lines.

Red and blue straight-move circuits have been combined in fig. 9.12 to form a "red-blue" circuit. The method of combination is indicated by the dotted and solid lines: the straight moves (dotted lines) are replaced by diagonal moves (solid lines). To complete a circuit of the whole board, with this red-blue circuit as basis, a green-yellow circuit has to be constructed, and this circuit combined with the red-blue circuit. This is left as an exercise.

A slight difficulty of this method is that it constrains the straight-move circuits that can be made. For example, considering the suggested method for combining red and blue circuits in fig. 9.11, no constraint is placed on the blue circuit (because there is only one way a straight-move circuit of the blue squares can enter and leave the bottom-left corner of the board). However, the straight-move circuit of the red squares is constrained by the requirement that it make use of the move shown as a dotted line. The difficulty is resolved by first choosing the combining moves and then constructing the straight-move

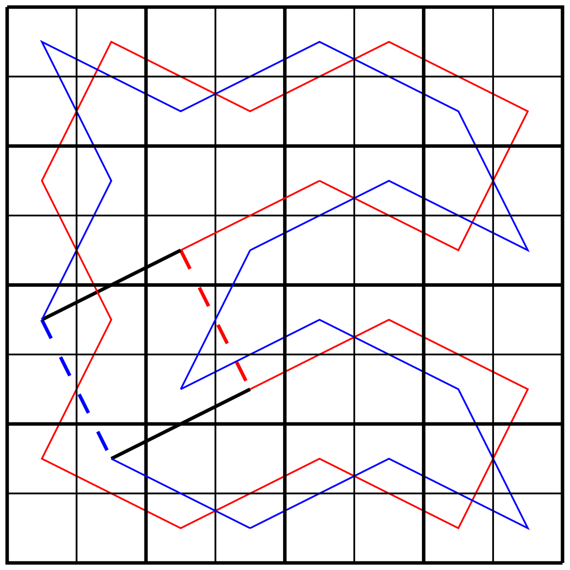


Figure 9.12: A "red-blue" circuit. "Parallel" red and blue moves, shown as dotted lines, are replaced by diagonal moves, shown as thicker solid lines, thus combining the two circuits.

circuits appropriately.

In order to construct a Knight's circuit of smaller size boards, the different pairs of combining moves need to be positioned as close as possible together. This is possible in the case of an 8×6 board, but not for smaller boards.

Exercise 9.10 Construct a Knight's circuit of an 8×8 board using the scheme discussed above. Do the same for an 8×6 board. Indicate clearly how the individual circuits have been combined to form the entire circuit.

□

Exercise 9.11 Figure 9.13 illustrates another way that the circuits can be combined. The four straight-move circuits are depicted as circles, one segment of which has been flattened and replaced by a dotted line. The dotted lines represent straight moves between consecutive points. If these are replaced by diagonal moves (represented in the diagram by solid black lines), the result is a circuit of the complete board.

To carry out this plan, the four diagonal moves in fig. 9.13 have to be identified. The key to doing this with a minimum of effort is to seek parallel red and green moves, and parallel blue and yellow moves, whilst exploiting symmetry. (In contrast, the above solution involved seeking parallel red and blue moves.) Choosing to start from, say, the

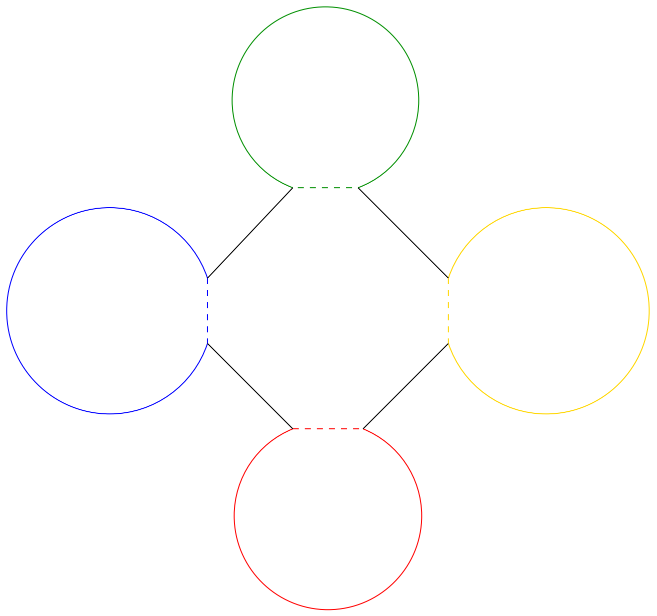


Figure 9.13: Schema for Combining Straight-Move Circuits. Four straight moves (indicated by dotted lines) are replaced by four diagonal moves (indicated by solid black lines).

pair of green moves in the bottom-left corner, severly restricts the choice of diagonal moves; in combination with symmetry, this makes the appropriate moves easy to find.

Construct a knight's circuit of an 8×8 *and a* 6×8 *board using the above scheme. Explain how to extend your construction to any board of size* $4m \times 2n$ *for any* m *and* n *such that* $m \geq 2$ *and* $n \geq 3$.

(The construction of the circuit is easier for an 8×8 board than for a 6×8 board because, in the latter case, more care has to be taken in the construction of the straight-move circuits. If you encounter difficulties, try turning the board through $90°$ whilst maintaining the orientation of the combining moves.)

□

Exercise 9.12 Division of a board of size $(4m+2) \times (4n+2)$ into supersquares yields a $(2m+1) \times (2n+1)$ "super" board. Because this superboard has an odd number of (super) squares, no straight-move circuit is possible, and the strategy used in exercise 9.10 is not applicable. However, it is possible to construct Knight's circuits for boards of size $(4m+2) \times (4n+2)$, whenever, both m and n are at least 1, by exploiting exercise 9.2.

The strategy is to construct four straight-move circuits of the board omitting one of the supersquares. (Recall exercise 9.2 for how this is done.) Then, for each circuit, one move is replaced by two moves —a straight move and a diagonal move— both with end points in the omitted supersquare. This scheme is illustrated in fig. 9.14.

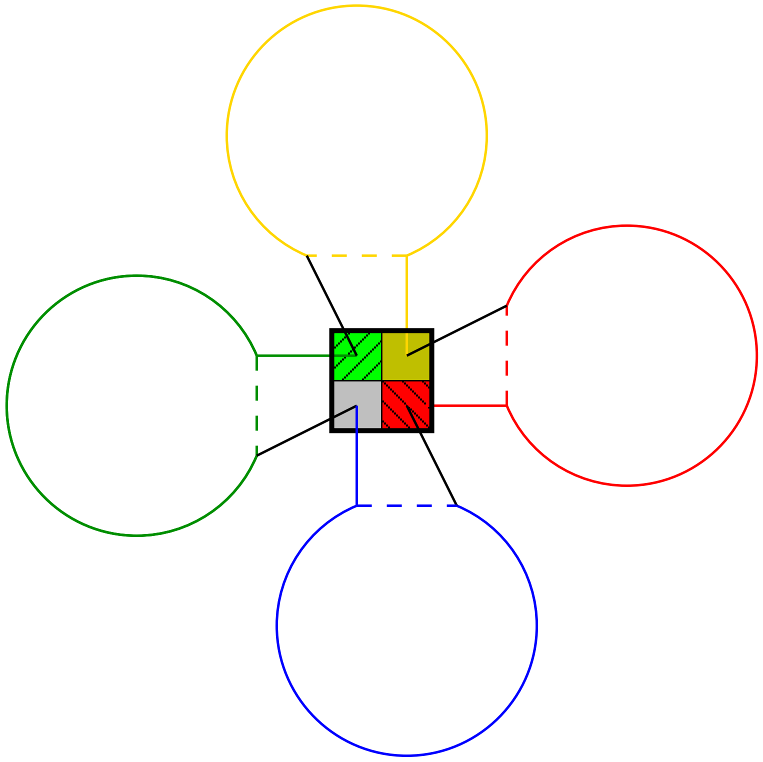


Figure 9.14: Strategy for Constructing a Knight's Circuit of $(4m+2)\times(4n+2)$ boards. One supersquare is chosen, and four straight-move circuits are constructed around the remaining squares. These are then connected as shown.

Complete the details of this strategy for a 6×6 board. Make full advantage of the symmetry of a 6×6 board. In order to construct the twelve combining moves depicted in fig. 9.14 , it suffices to construct just three; the remaining nine can be found by rotating the moves through a right angle.

Explain how to use your solution for the 6×6 board to construct Knight's circuits of any board of size $(4m+2)\times(4n+2)$, whenever, both m and n are at least 1.

□

9.4 Discussion

In the absence of a systematic strategy, the Knight's circuit problem is a truly difficult problem to solve, which means it is a very good example of disciplined problem-solving skills. The method we have used to solve the problem is essentially problem decomposition — reducing the Knight's circuit problem to constructing straight-move circuits, and combining these together.

The key criterion for a good *method* is whether or not it can be extended to other related problems. This is indeed the case for the method we have used to solve the Knight's circuit problem. The method has been applied to construct a circuit of an 8×8 chessboard, but the method can clearly be applied to much larger chessboards.

The key ingredients are

- the classification of moves as "straight" or "diagonal",
- straight-move circuits of supersquares, and
- using diagonal moves to combine straight-move circuits .

Once the method has been fully understood, it is easy to remember these ingredients, and reproduce a Knight's circuit on demand. Contrast this with remembering the circuit itself, which is obviously highly impractical, if not impossible.

A drawback of the method is that it can only be applied to boards that can be divided into supersquares. As we have seen, it is not possible to construct a Knight's circuit of a board with an odd number of squares. That leaves open the cases where the board has size $(2m)\times(2n+1)$, for some m and n. (That is, one side has even length and the other has odd length.) For those interested, a complete characterisation of the sizes of board for which a Knight's circuit exists is given in section 9.5.

The Knight's-circuit problem exemplifies a number of mathematical concepts which you will probably encounter elsewhere. The n, v, h and c operations together form an example of a "group"; the relation on squares of being connected by straight moves is an example of an "equivalence relation", and the fact that this relation partitions the squares into four disjoint sets is an example of a general theorem about "equivalence relations".

The Knight's-circuit problem is an instance of a general class of problems called "Hamiltonian-Circuit Problems". In general, the input to a Hamiltonian-circuit problem is a so-called "graph" (a network of "nodes" with "edges" connecting the nodes) and the requirement is to find a path through the graph that visits each node exactly once, before returning to the starting node. Hamiltonian-circuit problems are, in turn, instances of a class of problems called "NP-complete" problems. NP-complete problems are problems characterised by ease of verification but difficulty of construction. That is, given a putative solution, it is easy to check whether or not it is correct (for example, given any sequencing of the squares on a chessboard, it is easy to check whether or not it is a Knight's circuit); however, for the class of NP-complete problems, no efficient methods are known at this time for constructing solutions. "Complexity theory" is the name given to the area of computing science devoted to trying to quantify how difficult algorithmic problems really are.

9.5 Boards of Other Sizes

To be written.

9.6 Bibliographic Remarks

Solutions and historical information on the Knight's circuit problem can easily be found on the internet. According to one of these [MacQuarrie, St. Andrews Univ.] the first Knight's tour is believed to have been found by al-Adli ar-Rumni in the ninth century, long before chess was invented.

The use of supersquares in solving the Knight's circuit problem is due to Edsger W. Dijkstra [Dij92]. His solution is for a standard-sized chessboard and uses the method of combining straight-move circuits described in exercise 9.11. The pairwise combination of straight-move circuits is due to Diethard Michaelis [private communication]. Both solutions involve searching for "parallel moves". Michaelis's solution is slightly preferable because just two pairs of "parallel moves" have to be found at each stage. Dijkstra's solution involves searching for four pairs (twice as many) at the same time, making it a little bit harder to do. The solution to exercise 9.12 was constructed by the author, with useful feedback from Michaelis.

Solutions to Exercises

2.1 1233 games must be played. Let k be the number of players that have been knocked out, and let g be the number of games that have been played. Initially, k and g are both equal to 0. Every time a game is played, one more player is knocked out. So, k and g are always equal. To decide the tournament, $1234-1$ players must be knocked out. Hence, this number of games must be played.

In general, if there are p players, the tournament consists of $p-1$ games.
□

2.2 Let m, n and p denote the number of objects of each kind.

The replacement process is modelled by the assignment

$$m,n,p \ := \ m+1\,,n-1\,,p-1 \ .$$

Consider the *differences* $m-n$, $n-p$, $p-m$. It is easily checked that the parity of each is unchanged by the assignment. (In each case, the difference either is unchanged or increases or decreases by 2.) Also, the number of odd differences of any three numbers is always even (i.e. either zero or two). Since the goal is to reach a state in which there are two odd differences, we conclude that the goal is impossible to reach if the starting state has zero odd differences. The goal is also impossible to reach if the objects are all of the same kind and there is more than one of them.

The algorithm to remove objects maintains the invariant that all objects are of the same kind equivales there is only one object remaining. If there is more than one object remaining, there must be two objects of different kind. Choosing to increase the number of objects of the kind that occurs least frequently will maintain this invariant, and reduce the number of objects.
□

3.1

```
       { 5C || }
       2C,3H |3W| ; 2C,3H |1W| 2W ; 5H |3W| 2W
   ;   { 5H || 5W }
```

5H |2*W*| 3*W* ; 2C |3H| 3*W*

; { 2C || 3C }

2C |1C| 2C

; { 3C || 2C }

3*W* |3H| 2C ; 3*W* |2*W*| 5H

; { 5*W* || 5H }

2*W* |3*W*| 5H ; 2*W* |1*W*| 2C,3H ; |3*W*| 2C,3H

{ || 5C } .

□

3.2 We modify the solution to the five-couple problem, effectively by leaving one couple behind on the left bank. We get:

{ 4C || }

1C,3H |3*W*| ; 1C,3H |1*W*| 2*W* ; 4H |2*W*| 2*W*

; { 4H || 4*W* }

4H |2*W*| 2*W* ; 2C |2H| 2*W*

; { 2C || 2C }

2C |1C| 1C

; { 3C || 1C }

3*W* |3H| 1C ; 3*W* |1*W*| 4H

; { 4*W* || 4H }

2*W* |2*W*| 4H ; 2*W* |1*W*| 1C,3H ; |3*W*| 1C,3H

{ || 4C } .

By reversing left and right, we get the second solution:

{ 4C || }

1C,3H |3*W*| ; 1C,3H |1*W*| 2*W* ; 4H |2*W*| 2*W*

; { 4H || 4*W* }

4H |1*W*| 3*W* ; 1C |3H| 3*W*

; { 1C || 3C }

1C |1C| 2C

$; \quad \{ \ 2C \,||\, 2C \ \}$

$2W\,|2H|\,2C \ ; \ 2W\,|2W|\,4H$

$; \quad \{ \ 4W \,||\, 4H \ \}$

$2W\,|2W|\,4H \ ; \ 2W\,|1W|\,1C{,}3H \ ; \ |3W|\,1C{,}3H$

$\{ \ \,||\, 4C \ \} \ .$

□

3.3 Let M denote the capacity of the boat, and let N denote the number of couples. We assume that N is at least 2 and M is at most the minimum of 3 and $N/2$. (These properties are common to the cases of a boat of capacity 2 and 4 couples, and a boat of capacity 3 and 6 couples.)

Let lH denote the number of husbands on the left bank. The number of husbands on the right bank, denoted rH, is then $N{-}lH$. Similarly, let lW denote the number of wives on the left bank. The number of wives on the right bank, denoted rW, is then $N{-}lW$.

We note that an invariant is

(13) $\quad (lH = lW) \vee (lH = 0) \vee (lH = N) \ .$

That is, either there are no single individuals on either bank, or all husbands are on one of the two banks. It is a requirement of any solution that this property is an invariant. (If not, either $0 < lH < lW$ or $0 < rH < rW$. In words, the wives outnumber the husbands on the left bank or on the right bank. In both cases, the solution is invalid.)

Now, we claim that, under the given assumptions on M and N, either

(a) The boat is on the left bank and

(14) $\quad M < lH \ ,$

or

(b) the boat is on the right bank and

(15) $\quad M \leq lH \ .$

Property (a) is clearly true initially. (Recall the assumptions about M and N.)

Now, suppose (a) is true and, then, a crossing is made from left to right. Note that $lH \neq 0$ both before and after the crossing. (Before the crossing, $lH = 0$ is excluded by the assumption that $M < lH$. After the crossing, $lH = 0$ is impossible because at most M

husbands can cross together.) So, the invariant (13) can be strengthened: the crossing is made starting in a state in which

(16) $(lH = lW) \vee (lH = N)$,

and must result in a state satisfying this property. Otherwise, the crossing is invalid. Note also that a left-to-right crossing causes lH and/or lW to decrease.

We consider two cases before the crossing: $lH = N$, and $(lH = lW) \wedge (lH \neq N)$.

Since at most $N/2$ husbands can cross together, if $lH = N$ before the crossing, at least $N/2$ are left at the left bank. That is, (b) is true after the crossing.

If $(lH = lW) \wedge (lH \neq N)$ before the crossing, the property $lH = lW$ must be maintained by the crossing. (This is because (16) must be maintained and the crossing cannot increase lH .) That is, an equal number of wives and husbands must make the crossing. Since only one couple can cross at one time, the value of lH is decreased by at most 1 . But (14) is assumed to be true before the crossing; consequently, (15) is true after the crossing, and the boat is at the right bank. Thus, (b) is true after the crossing.

In both cases, we have established that, after the crossing, (b) is true.

Now suppose (b) is true and a crossing is made from right to left. A right-to-left crossing causes lH and/or lW to increase. Since $lH \neq 0$ before the crossing, and lH does not decrease, $lH \neq 0$ after the crossing. (Strictly, we need to assume that M is non-zero in order to assert that $lH \neq 0$. Obviously, the case that the boat has capacity 0 can be excluded from consideration!)

Again, we consider two cases: this time, no husbands cross, and some husbands cross.

If husbands cross, the act of crossing increases lH ; so, after the crossing $M < lH$ and (of course) the boat is at the left bank. Thus, (a) is true after the crossing.

If only wives cross, it must be the case that $lH = N$ (because (16) must be maintained and, if only wives cross, it is impossible to maintain that $lH = lW$). That is, lH is unchanged, and $M < lH$ both before and after the crossing. After the crossing, the boat is, of course, on the left side of the bank. Thus, (a) is true after the crossing.

In both cases, we have established that (a) is true after the crossing.

In summary, property (a) is true initially. Also, if (a) is true and a left-to-right crossing is made, (b) becomes true; vice-versa, if (b) is true and a right-to-left crossing is made, (a) becomes true. So, at all times, either (a) or (b) is true. That is, it can never be the case that all husbands are at the right bank.

□

3.4 As stated in the hint, we assume that every forward trip involves two people and every return trip involves one person. (We omit the justification of this assumption here. See chapter 8 for a general argument why this assumption may be made.)

A consequence is that an optimal solution consists of exactly 5 crossings, of which 2 are return trips. Since each return trip is made by just one person, 2 people never make a return trip. Clearly, an optimal solution is found when persons 3 and 4 —the two slowest— do not return. (Consider any sequence of crossings that gets all four across the bridge. If one or both of the two slowest makes a return journey, then one or both of the two fastest does not make a return journey. By interchanging a slower person with a faster person in the sequence, a new sequence is found which is at least as fast as the original sequence — since the time for *at least one* return trip is reduced and the time for *at most one* forward trip is increased by the same amount.)

There are two strategies for getting the two slowest across: let them cross together or let them cross seperately. The strategy "let the two slowest cross together" is implemented by letting persons 1 and 2 (the two fastest) cross, with person 1 returning. Then persons 3 and 4 cross, and person 2 returns. Finally, persons 1 and 2 cross again. The total time taken using this strategy is

$$t_1{\uparrow}t_2 + t_1 + t_3{\uparrow}t_4 + t_2 + t_1{\uparrow}t_2 \ \ .$$

(The infix operator "$\uparrow$" denotes maximum. Because we assume that $t_1 \le t_2 \le t_3 \le t_4$, $t_1{\uparrow}t_2$ simplifies to t_2 and $t_3{\uparrow}t_4$ simplifies to t_4. However, not simplifying the formula just yet makes it is easy to identify the people in each crossing.) If the two slowest cross seperately, then the best strategy is to let person 1 cross with them and then return for the other. Implementing this strategy takes a total time of

$$t_1{\uparrow}t_2 + t_1 + t_1{\uparrow}t_4 + t_1 + t_1{\uparrow}t_3 \ \ .$$

(This corresponds to person 1 and person 2 crossing, then person 1 returning, then person 1 and person 4 crossing, then person 1 returning, and finally persons 1 and 3 crossing. The order in which persons 2, 3 and 4 cross is, of course, immaterial. The order chosen here facilitates the comparison of the times.)

Comparing the total times, the first strategy should be used when $t_2{+}t_2 \le t_1{+}t_3$ and the second strategy when $t_1{+}t_3 \le t_2{+}t_2$. (There is a small element of nondeterminism in this solution: when $t_2 + t_2 = t_1 + t_3$ an arbitrary choice may be made between the two strategies.)

Applying this solution to the two specific cases, we get:

(a) The times taken are 1 minute, 1 minute, 3 minutes and 3 minutes: Since $1{+}1 \le 1{+}3$, the two slowest should cross together. The shortest time is $1{+}1{+}3{+}1{+}1$, i.e. 7 minutes.

(b) The times taken are 1 minute, 4 minutes, 4 minutes and 5 minutes. Since $1{+}5 \le 4{+}4$, the two slowest should cross seperately. The shortest time is $4{+}1{+}5{+}1{+}4$, i.e. 15 minutes. (The shortest time if the two slowest cross together is $4{+}1{+}5{+}4{+}4$, i.e. 18 minutes.)

□

3.5 $6{\times}2{\times}3{\times}3{\times}1$, i.e. 108.
□

4.1 a) Naming any day in December, other than 31st December results in losing. This is forced by the opponent naming 30th November (that is, the last day of November). Similarly, naming any day other than 30th November results in losing, because the opponent can then name 30th November. This is forced by the opponent naming 31st October. In general, the winning strategy is to name the last day of the month. The opponent is then forced to name the 1st of the next month. Whether the year is a leap year or not makes no difference.

b) In December, the losing positions are the odd-numbered days and the winning positions are the even-numbered days. (Take care: The "losing positions" are the days that the *winning* player names. This is in line with the terminology of losing and winning positions.) That is, if the last-named day is an odd-numbered day, the player whose turn it is must name an even-numbered day and, so, will eventually lose.

In particular, the player who names 1st December wins. Any day in November is thus a winning position. In October, like December, the odd-numbered days are losing positions, and any day in September is a winning position. Similarly, in August, the odd-numbered days are losing positions, and any day in July is a winning position.

The pattern changes in June, which has an even number of days. The player who names 1st July loses; consequently, any even-numbered day in June is a losing day. This means that every even-numbered day in May is a winning day; ; also, every even-numbered day in April is a winning day. This means that 31st March is a losing day. Since March has an odd number of days, the pattern we saw for December and November recurs. Every odd-numbered day in March is a losing day, and every day in February is a winning day. Finally, the odd-numbered days in January are losing days.

We conclude that the second player is guaranteed to win. The strategy is to name the 1st day of the following month when the last-named day is in November, September, July or February. Otherwise, the strategy is to name the next day of the year.

Again, it does not matter if it is a leap-year.
□

4.2 The first eleven positions are shown in table 1.

The pattern repeats in the second eleven positions. See table 2.
□

4.3 The squares that are not positions are the ones at the foot of a ladder or at the head of a snake. Positions that cannot be identified as winning or losing positions are

Position	0	1	2	3	4	5	6	7	8	9	10
Type	L	L	W	W	L	W	W	W	L	W	W
Move			2	2		5	6	6		5	6

Table 1: Winning (W) and Losing (L) Positions for subtraction set $\{2,5,6\}$

Position	11	12	13	14	15	16	17	18	19	20	21
Type	L	L	W	W	L	W	W	W	L	W	W
Move			2	2		5	6	6		5	6

Table 2: Winning (W) and Losing (L) positions for subtraction set $\{2,5,6\}$

attributable to cycles in the positions; a *cycle* is a sequence of moves that begins and ends at the same position. Labelling of winning and losing positions assumes that every game is guaranteed to terminate no matter how either player moves. If cycles occur this assumption is not valid.

When a game has cycles, the positions are characterised as losing positions, winning positions or stalemate positions. A losing position is one from which every move is to a winning position; a winning position is one from which there is a move to a losing position; and a stalemate position is one from which there is a move to a stalemate position and there are no moves to losing positions.

From a stalemate position the best strategy is to move to a stalemate position since, if there is an alternative of moving to a winning position, this is clearly the wrong thing to do. The opponent will then use the same strategy and the game will continue forever.

Table 3 shows all positions and the winning move from winning positions. Position 4 is the only stalemate position. From this position, a move to square 6 has the effect of returning the counter to position 4. Any other move from 4 would be to a winning position.

Position	1	2	4	5	7	13	14	16	18	21	22	23	24	25
Type	W	W	S	W	W	L	W	W	L	W	W	W	W	L
Move to square	3	3	6	9	9		18	18		25	25	25	25	

Table 3: Snakes and Ladders. Winning (W), Losing (L) and Stalemate (S) positions

□

4.4 a) See table 4 for the mex numbers up to and including position 10. The mex numbers repeat from here on; that is, the mex number for position m is equal to the mex number for position $m \bmod 11$.

Position	0	1	2	3	4	5	6	7	8	9	10
Type	L	L	W	W	L	W	W	W	L	W	W
Mex Number	0	0	1	1	0	2	1	3	0	2	1

Table 4: Mex numbers for subtraction set $\{2,5,6\}$.

Left Game	Right Game	"losing" or winning move
10	20	R14
20	20	losing
15	5	R0
6	9	R4
37	43	losing

Table 5: Winning moves

In the left game, the mex number of position m is $m \bmod 3$. Together with the mex numbers for the right game given above, we can complete table 5. (Other answers can be given for the winning moves.)
□

4.5 (a) The losing positions are positions $2^{i+1}-1$ where i is a natural number; all other positions are winning positions.

The proof is in two parts: we show that, for all i, every move from position $2^{i+1}-1$ is to a position n where $2^i-1 < n < 2^{i+1}-1$; also, from a position n where, for all i, $n \neq 2^{i+1}-1$ we show that we can choose i so that there is a move from n to position 2^i-1.

When i equals 0, $2^{i+1}-1$ equals 1. Position 1 is an end position and thus a losing position. When i is greater than 0, every move from position $2^{i+1}-1$ is to a position n where $n < 2^{i+1}-1 \leq 2\times n$. But

$$n < 2^{i+1}-1 \leq 2\times n$$

$$= \qquad \{ \quad \text{meaning of continued equalities} \quad \}$$

$$n < 2^{i+1}-1 \;\wedge\; 2^{i+1}-1 \leq 2\times n$$

$$= \qquad \{ \quad \text{integer inequalities , symmetry of } \wedge \quad \}$$

$$
\begin{array}{cl}
 & 2^{i+1}{-}2 < 2{\times}n \;\;\wedge\;\; n < 2^{i+1}{-}1 \\
= & \{ \quad \text{monotonicity of } 2{\times} \quad \} \\
 & 2^{i}{-}1 < n \;\;\wedge\;\; n < 2^{i+1}{-}1 \\
= & \{ \quad \text{meaning of continued equalities} \quad \} \\
 & 2^{i}{-}1 < n < 2^{i+1}{-}1 \quad .
\end{array}
$$

This establishes the first part.

For the second part, suppose that, for all i, $n \neq 2^{i+1}{-}1$. Let i be the largest number such that $2^{i}{-}1 < n$. Then, by definition of i, $n \leq 2^{i+1}{-}1$. That is, $2^{i}{-}1 < n \leq 2^{i+1}{-}1$. But then

$$
\begin{array}{cl}
 & \text{there is a move from } n \text{ to } 2^{i}{-}1 \\
= & \{ \quad \text{definition of legal moves} \quad \} \\
 & 2^{i}{-}1 < n \leq 2{\times}(2^{i}{-}1) \\
= & \{ \quad \text{arithmetic} \quad \} \\
 & 2^{i}{-}1 < n \leq 2^{i+1}{-}2 \\
= & \{ \quad \text{integer inequalities} \quad \} \\
 & 2^{i}{-}1 < n < 2^{i+1}{-}1 \\
= & \{ \quad \text{assumption: for all } i,\; n \neq 2^{i+1}{-}1 \quad \} \\
 & 2^{i}{-}1 < n \leq 2^{i+1}{-}1 \\
= & \{ \quad \text{definition of } i \quad \} \\
 & \text{true} \quad .
\end{array}
$$

(b)

Position:	1
Mex Number:	0

Position:	2	3
Mex Number:	1	0

Position:	4	5	6	7
Mex Number:	2	1	3	0

Position:	8	9	10	11	12	13	14	15
Mex Number:	4	2	5	1	6	3	7	0

No. of Columns	No. of Rows	"losing" or winning move
2	15	C1 (or R11)
4	11	C2 (or R9)
4	14	R9
13	6	losing
21	19	C19 (or R10)

Table 6: Solution to rectangle game

In general, $\text{mex}.(2{\times}n)$ equals n and $\text{mex}.(2{\times}n+1)$ equals $\text{mex}.n$.

□

5.5 Let col be the colour of the square, and n be the number of moves. A move is then

$$col\,,n \;:=\; \neg col\,,n{+}1 \quad .$$

An invariant of this assignment is

$$col \equiv \mathit{even}.n \quad .$$

An odd number of moves (63) is needed, but the colour of the square doesn't change. So, in order to move the knight as required, a change has to be made to $col \equiv \mathit{even}.n$, which is impossible.
□

5.6 Suppose the number of couples is n. There are $2n$ people, including the host, who each shake hands with between 0 and $2n-2$ people. If $2n-1$ of them —everyone but the host— shake hands with a different number of people, there must be someone who shakes hands with k people for each k between 0 and $2n-2$ (inclusive).

If n is 1, the only person other than the host is the host's partner. Since couples do not shake hands, both shake hands 0 times.

Now suppose that n is greater than 1. In this case, there are at least two people other than the host and the host's partner. Consider the two people who shake hands 0 and $2n-2$ times. The person who shakes hands $2n-2$ times does so with everyone except their partner (and themself, of course). By the symmetry of the shake-hands relation, it is thus the case that everyone except that person's partner shakes hands with at least one person. It follows that the two people who shake hands 0 and $2n-2$ times are husband and wife. Because neither is the host, it also follows that neither is the host's partner.

Now suppose we discount this couple. That is, we consider the party consisting of the other $n{-}1$ couples. The number of times each person shakes hands is then reduced by one. So, again, all but the host have shaken hands a distinct number of times.

Repeating this process, we eliminate all the couples one by one until the party has been reduced to just the host and the host's partner. Each time, the number of times the host and the host's partner shake hands is reduced by one. The host and the host's partner must therefore have both shaken hands $n{-}1$ times.

□

5.8

(a) false

(b) false

(c) false

(d) p

(e) false

(f) $q \not\equiv r$

(g) p

(h) true

□

5.9

$$
\begin{array}{ll}
 & \neg\mathsf{true} \\
= & \{ \quad \text{law } \neg p \equiv p \equiv \mathsf{false} \text{ with } p := \mathsf{true} \quad \} \\
 & \mathsf{true} \equiv \mathsf{false} \\
= & \{ \quad \text{law } \mathsf{true} \equiv p \equiv p \text{ with } p := \mathsf{false} \quad \} \\
 & \mathsf{false} \ .
\end{array}
$$

□

5.10

$$
\begin{array}{ll}
 & \neg\neg p \\
= & \{ \quad \text{law } \neg p \equiv p \equiv \mathsf{false} \text{ with } p := \neg p \quad \}
\end{array}
$$

$$\begin{array}{ll} & \neg p \equiv \mathsf{false} \\ = & \{ \quad \text{law } \neg p \equiv p \equiv \mathsf{false} \text{ with } p := p \\ & \quad\ \text{and symmetry of equivalence} \quad \} \\ & p \ . \end{array}$$

□

5.11 The process of decryption after encryption computes $a \not\equiv (a \not\equiv b)$. But,

$$\begin{array}{ll} & a \not\equiv (a \not\equiv b) \\ = & \{ \quad \not\equiv \text{ is associative} \quad \} \\ & (a \not\equiv a) \not\equiv b \\ = & \{ \quad (\, a \not\equiv a \equiv \mathsf{false}\,) \quad \} \\ & \mathsf{false} \not\equiv b \\ = & \{ \quad \text{definition of } \not\equiv \quad \} \\ & \mathsf{false} \equiv \neg b \\ = & \{ \quad \text{definition of negation: (5.3)} \quad \} \\ & b \ . \end{array}$$

□

5.12 Let Q be the question. Then, $Q \equiv A \equiv A \not\equiv B$, i.e. $Q \equiv \neg B$. In words, ask A whether B is a knave.
□

6.1 It is required that any two lines intersect in a single point. If the lines are not straight and they intersect in a segment of a line, inverting the colours of one of the two regions does not guarantee that the colouring of adjacent regions at the boundary of the left and right regions is satisfactory. This is because, along the line segment, the colours of the adjacent regions are not the same before the inversion takes place contrary to the assertion made above.

The solution remains valid provided every line cuts the surface in two. A line on a ball does this, whereas a line on a doughnut need not.

The number of colourings is always two no matter how many lines there are. This is clearly the case when there are no lines. When there are $n+1$ lines, choose any one of the lines. Cut the paper along the chosen line. Assume inductively that, for each half, there are exactly two colourings. Combining these gives four different ways of colouring the entire sheet of paper. However, two of these are unsatisfactory because the colours

of regions adjacent at the chosen line must be different. This leaves exactly two ways of colouring the paper with $n+1$ lines.
□

6.5 When m is 0, there is just one object. This is the unique object and 0 (which equals 2×0) comparisons are needed to discover that fact.

Suppose now that m is greater than 0. Split the 3^m objects into 3 groups each of 3^{m-1} objects. One of these 3 groups will have a different weight to the other two, which will be of equal weight. At most 2 comparisons are needed to determine which of the groups it is. Then, by induction, at most a further $2\times(m-1)$ comparisons are required to find the unique object in that group. This gives a total of $2\times(m-1)+2$, i.e. $2\times m$, comparisons as required by the induction hypothesis.

It is possible to determine whether the unique object is lighter or heavier than the others (although, in the case that there is just one object, the answer is that it is both lighter and heavier than all the rest). It can be decided in the first two comparisons.
□

6.6 a) For $n=1$, it is clear that 0 comparisons are needed. For the induction step, assume that $n-1$ comparisons are needed to find the lightest of n objects. To find the lightest of $n+1$ objects, use $n-1$ comparisons to find the lightest of n objects, then compare this object with the $(n+1)$th object. The lightest of the two is the lightest of them all. Also, one extra comparison has been made, making n in total.
b) For $n=2$, it is clear that 1 comparison is needed. For the induction step, assume that $2n-3$ comparisons are needed to find the lightest and heaviest of n objects. To find the lightest and heaviest of $n+1$ objects, use $2n-3$ comparisons to find the lightest and heaviest of n objects. Call these L and H. Call the $(n+1)$th object N. The lightest of L and N is the lightest of them all, and the heaviest of H and N is the heaviest of them all. This requires two extra comparisons, making $(2n-3)+2$, i.e. $2(n+1)-3$ in total.
c) Compare A and C. The lightest of the two is the lightest of the four. Compare B and D. The heaviest of the two is the heaviest of the four.

To weigh four objects, first compare two. Call the lighter one A and the heavier one B. Likewise, compare the remaining two objects and call the lighter one C and the heavier one D. Then proceed as above.
d) For $m=1$, it is clear that 1 comparison is needed to find the lightest and heaviest of 2 objects. And, $1=3\times1-2$.

Suppose there are $2(m+1)$ objects. Select and compare any two of the objects. Let the lightest be A and the heaviest B. By induction, we can find the lightest and heaviest of the remaining $2m$ objects in $3m-2$ comparisons. Let these be C and D, respectively. We now have four objects, A, B, C and D, such that $A<B$ and

$C < D$. By part (c), the lightest and heaviest of these four can be found in 2 further comparisons. These are then the lightest and heaviest of all $2(m+1)$ objects. And, the total number of comparisons is $1 + (3m-2) + 2$ which equals $3(m+1)-2$.
□

7.1 Formally we have

$$
\begin{array}{ll}
 & T_o.d \\
= & \{ \quad \text{definition of } T \quad \} \\
 & length(H_0.d) \\
= & \{ \quad \text{definition of } H_0.d \quad \} \\
 & length.[\,] \\
= & \{ \quad \text{definition of } length \quad \} \\
 & 0 \quad ,
\end{array}
$$

and

$$
\begin{array}{ll}
 & T_{n+1}.d \\
= & \{ \quad \text{definition of } T \quad \} \\
 & length(H_{n+1}.d) \\
= & \{ \quad \text{definition of } H_{n+1}.d \quad \} \\
 & length(H_n.\neg d \ ; \ [\langle n{+}1\,,\neg d\rangle] \ ; \ H_n.\neg d) \\
= & \{ \quad \text{definition of } length \quad \} \\
 & length(H_n.\neg d) + length([\langle n{+}1\,,\neg d\rangle]) + length(H_n.\neg d) \\
= & \{ \quad \text{definition of } T \text{ (twice) and } length \quad \} \\
 & T_n.\neg d \ + \ 1 \ + \ T_n.\neg d \quad .
\end{array}
$$

That is,

$$
\begin{array}{rcl}
T_o.d & = & 0 \\
T_{n+1}.d & = & 2 \times T_n.\neg d \ + \ 1 \quad .
\end{array}
$$

If we expand these equations for $n = 0\,,1\,,2\,,\ldots$, just as we did for the equations for H, we discover that $T_o.d$ is 0, $T_1.d$ is 1 and $T_2.d$ is 3 (in each case for all d). This and the form of the equation for $T_{n+1}.d$ (in particular the repeated multiplication by 2) suggest that $T_n.d$ is $2^n{-}1$. The simple inductive proof is omitted.
□

Figure 15: State-transition diagram for 0-disk problem.

7.2 We begin by considering the permissible states that the puzzle may be in. In any state, the disks on any one pole are in order of decreasing size. So, if we want to specify the state of the puzzle we only need to specify which pole each disk is on. For example, suppose there are five disks and suppose we specify that disk 1 is on pole A, disk 2 is on pole B, disks 3 and 4 are on pole A and disk 5 is on pole B. Then disk 4 must be on the bottom of pole A, disk 3 must be on top of it, and disk 1 must be on top of disk 3. Also, disk 5 must be on the bottom of pole B and disk 2 must be on top of it. No other arrangement of the disks satisfies the rule that no disk is above a disk smaller than itself.

The state of an n-disk puzzle can thus be specified by a sequence of n pole names. The first name in the sequence is the location of disk 1, the second is the location of disk 2, and so on. That is, the kth name in the sequence is the location (pole name) of disk k. Since each disk may be on one of three poles we conclude that there are 3^n different states in the n-disk problem.

Now we consider the transitions between states. We consider first the problem where there are no disks, then the 1-disk problem, then the 2-disk problem, and then we consider the general n-disk problem.

When there are no disks there is exactly one state: the state when there are no disks on any of the poles. This is shown in fig. 15. (You may have difficulty seeing the figure. It consists of a single dot!)

We now explain how to construct the state-transition diagram for the $(n+1)$-disk problem, for an arbitrary n, given that we have constructed the diagram for the n-disk problem. (See fig. 16.) Each state is a sequence of $n+1$ pole names. The first n names specify the location of the smallest n disks and the $(n+1)$th specifies the location of the largest disk. Thus, each state in the state-transition diagram for the n-disk problem gives rise to 3 states in the state-transition diagram for the ($n+1$)-disk problem. That is, a state in the state-transition diagram for the ($n+1$)-disk problem is specified by a sequence of n pole numbers followed by the pole name A, B or C. We split the permissible moves into two sets: those where the largest disk (the disk numbered $n+1$) is moved and those where a disk other than the largest disk is moved.

Consider first moving a disk other than the largest disk. When doing so, the largest disk may be on pole A, B or C. But its position doesn't affect the permissibility or otherwise of a move of a smaller disk. That means that every transition from state s to state t in the n-disk problem is also a valid transition from state sp to state tp in the ($n+1$)-disk problem, where the pole name p is either A, B or C. The first step in the construction of the state-transition diagram for the ($n+1$)-disk problem given the state-transition diagram for the n-disk problem is to make three copies of the latter. The pth copy is then modified by simply adding p at the end of each sequence of pole numbers labelling the nodes.

Now consider moving the largest disk, the disk numbered $n+1$. Being the largest disk it may only be moved if all the other disks are on one and the same pole different to the pole that the largest disk is on. This gives six possibilities for moving disk $n+1$, or three edges in the undirected state-transition diagram: an edge connecting the states A^nB and A^nC, an edge connecting the states B^nC and B^nA and an edge connecting the states C^nA and C^nB. The construction is shown schematically in fig. 16, the three inner triangles representing the set of all moves that do not move disk $n+1$.
□

7.3 Even, because the direction of movement is opposite to that of the smallest disk (which has an odd number).
□

7.4 The algorithm is to repeatedly execute the following procedure until it can no longer be executed (i.e. when it is no longer possible to determine k in step 1).

1. Suppose it is possible to move disk k in the direction d', where $k>1$. (Recall that disk 1 is the smallest disk.) Set d to $odd.k \equiv d'$.

2. Move disk k (in the direction d', of course).

3. Move the smallest disk in the direction d.

The correctness is justified as follows. When step 1 is executed, we know that the first $k-1$ disks are all on the pole in direction $\neg d'$ from disk k. Progress is made if these k smallest disks can be transferred to the same pole. To do this, it is necessary to move the $k-1$ smallest disks in the direction $\neg d'$. The direction that disk 1 has to be moved is thus d where

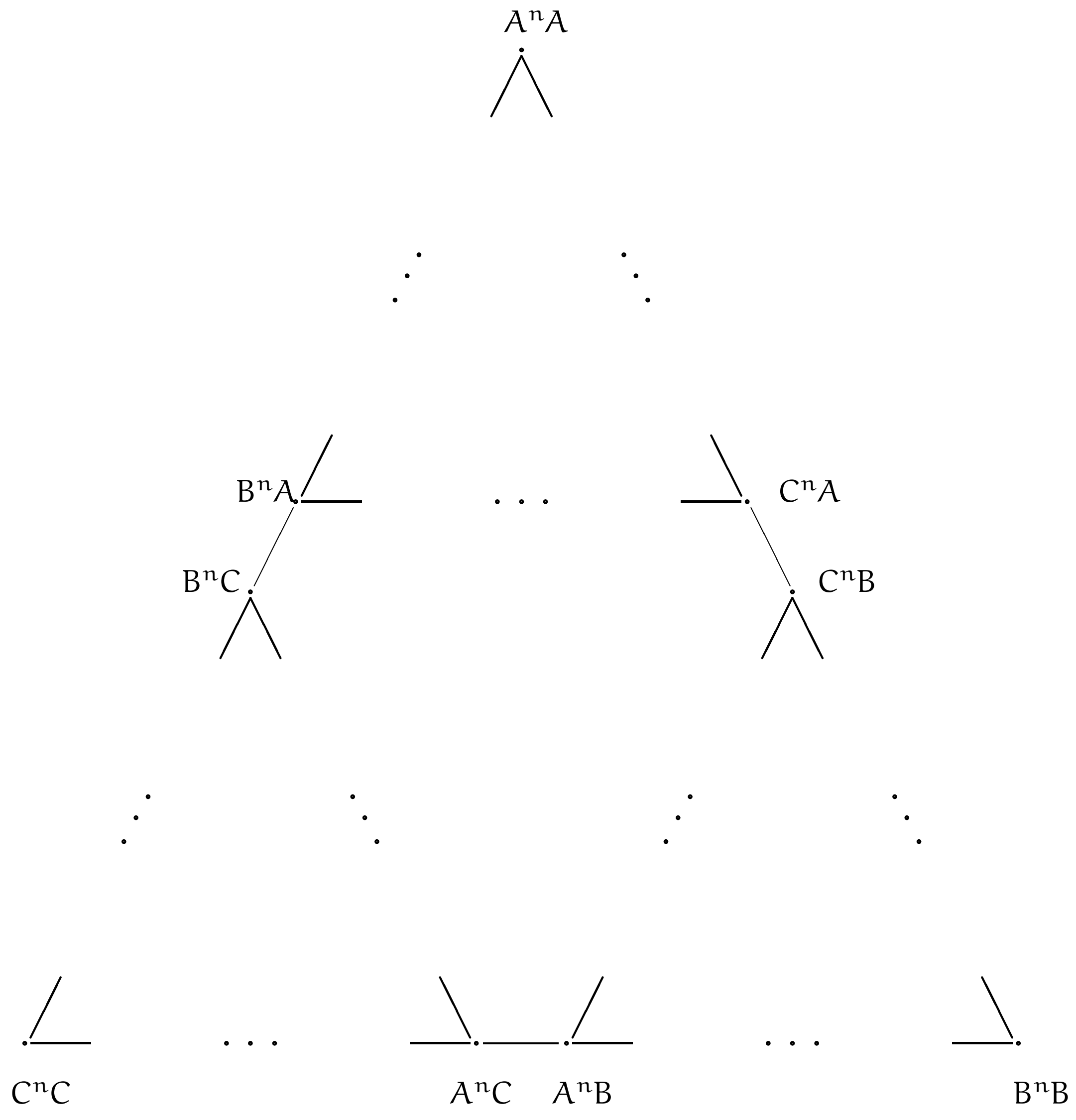


Figure 16: Construction of the state-transition diagram for the $(n+1)$-disk problem

$$even.(k-1) \equiv \neg d' \equiv even.1 \equiv d \quad .$$

Simplifying, we get that $d = (odd.k \equiv d')$. (In words, the direction that the smallest disk is moved should be the same as the direction that disk k is moved, if k is also odd; otherwise the smallest disk is moved in the opposite direction to disk k.) The correctness of the Towers of Hanoi program then guarantees that this action will initiate a sequence of moves after which all $k-1$ disks will have been moved onto disk k. During this sequence of moves the smallest disk will continue to move in the same direction. On completion, however, the direction of the smallest disk may or may not be reversed.

The only time that step 1 cannot be executed is when all the disks are on the same pole, as required.

□

7.5 The solution is to place the disks in order, starting with the largest and ending with the smallest. Let k denote the number of the disks still to be replaced; so, initially k is N and we are done when k is 0. Each time the value of k is reassigned, we ensure that the k smallest disks are on the same pole.

If the kth disk is on the right pole, decrease k by 1. Otherwise, suppose it needs to be moved in direction d from its current position. Move the smallest $k-1$ disks in the direction $\neg d$, then move disk k to its rightful position. Finally, decrease k by 1. Continue this process until k is 0.

□

9.1

(a) The number of moves that have to be made equals the number of squares. After an odd number of moves, the colour of the current square is different from the colour of the starting square. So, after an odd number of moves, it is impossible to return to the starting square.

(b) It's easy to see that a straight-move circuit of a $2{\times}1$ board is possible —starting at one of the squares, move to the other square and then move back again— , but, otherwise, no straight-move circuit is possible. If m is greater than 1, at least one square is two moves from the starting square; it is impossible to visit such a square and return to the starting square without visiting the in-between square more than once.

(c) See (b) for a circuit of a $2{\times}1$ board. For n greater than 1, a straight-move circuit of a $2{\times}n$ board is completed by starting at a corner, moving one-by-one to all the squares in the same row, then returning via the second row.

□

9.2 For the 3×3 board, a circuit can be constructed exactly when the omitted square is not adjacent to a corner square. For larger boards, the same condition applies. Suppose the coordinates of the omitted square are (m, n). (It doesn't matter whether numbering starts at zero or one.) Then a circuit can be constructed of the remaining squares exactly when $\text{even}.m = \text{even}.n$. The construction is to split the board into four rectangular boards in such a way that the to-be-omitted square is at a corner of a board with an odd number of squares. The other three boards each have an even number of squares, and at least one of them has at least one square. Construct circuits of these three boards, and —inductively, with the 3×3 board as the base case— a circuit of the board with the omitted square. Then, connect the circuits together as shown in fig. 17 .

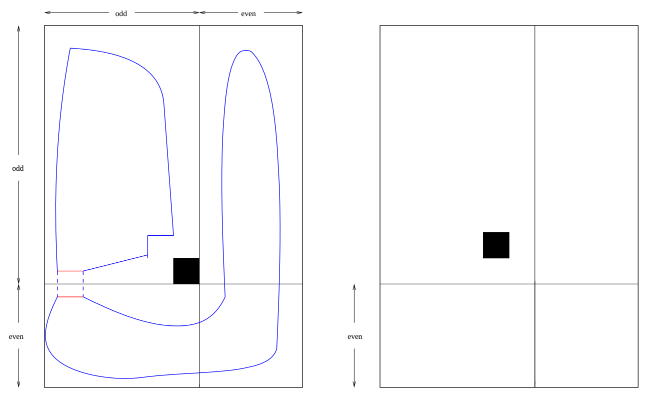


Figure 17: Straight-move circuits (shown in red) of a 3×3 board, omitting one of the squares.

□

9.6

;	n	v	h	c
n	n	v	h	c
v	v	n	c	h
h	h	c	n	v
c	c	h	v	n

Table 7: Sequential Composition of Flip Operations

Property (9.7) is verified by observing that the table is symmetric about the top-left to bottom-right diagonal. Verification of the associativity property is much more tedious. The case that x, y or z is n can be dealt with simply. This leaves 27 other cases to

consider. This is an example of a “tedious, but straightforward” proof!
□

9.9

$;$	n	r	a	c
n	n	r	a	c
r	r	c	n	a
a	a	n	c	r
c	c	a	r	n

Table 8: Sequential Composition of Rotation Operations

There are 24 different ways to assign a different colour to each of the squares in a 2×2 board, so that the size of the full table is 24×24! Rather than complete the full table, it suffices to understand how all the entries are generated by a small set of primitive transformations. (To be completed.)
□

9.10 See fig. 18.

The parallel moves used to connect circuits of different colours are indicated by dotted lines depicting the straight moves, and solid black lines, depicting the diagonal moves.

Note how the choice of parallel moves constrains the choice of red circuit. In fact, the red moves are entirely dictated by this choice. In contrast, there is complete freedom in choosing a blue or yellow circuit. There is some freedom in choosing a green circuit, but not complete freedom. In this way, a substantial number of circuits can be found all based on the same set of combining parallel moves. In the circuit shown, the green, blue and yellow circuits were constructed by “copying” the red circuit.

The same set of combining parallel moves can be used to construct a circuit of an 8×6 board; all that is required is to “shorten” the straight-move circuits in order to accommodate the smaller board. (But note that they cannot be used to construct a circuit of a 6×8 board.)
□

9.11 Figure 19 shows details of how the straight-move circuits are combined. Moves indicated by dotted lines are replaced by the diagonal moves indicated by solid black lines.

Figure 20 shows the circuits obtained in this way. The dotted lines are not part of the circuit; these are the moves that are replaced.

In order to construct a circuit for any board of size $4m\times 2n$, where m is at least 2 and n is at least 3, it suffices to use the technique detailed in figs. 9.2 and 9.3 for

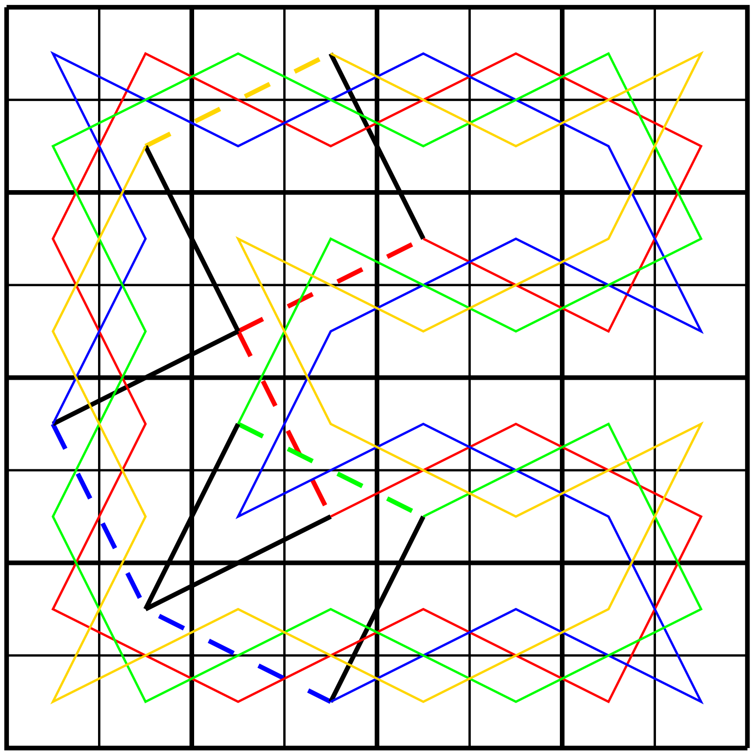


Figure 18: A Knight's Circuit. Solid lines indicate the circuit. The dotted lines depict straight moves that are replaced. The diagonal moves that replace them are depicted by solid black lines.

extending straight-move circuits to boards of arbitrary size. This construction has to be applied four times, once for each of the straight-move circuits in the solution to the 8×6-board problem.

□

9.12 We begin by identifying the moves shown in fig. 9.14. See fig. 21. (Note the symmetry.)

Now it is easy to fill in the straight-move circuits around the remaining squares. See fig. 22.

For the general problem, it is easy to extend the straight-move circuits.

□

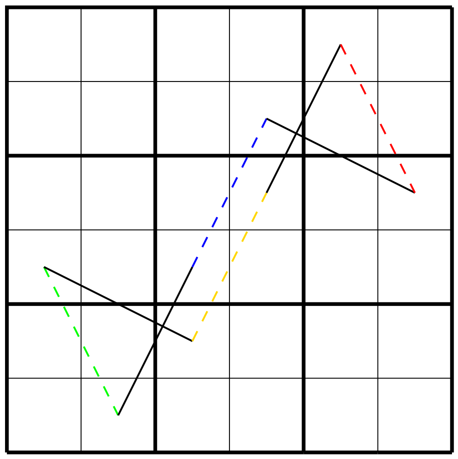
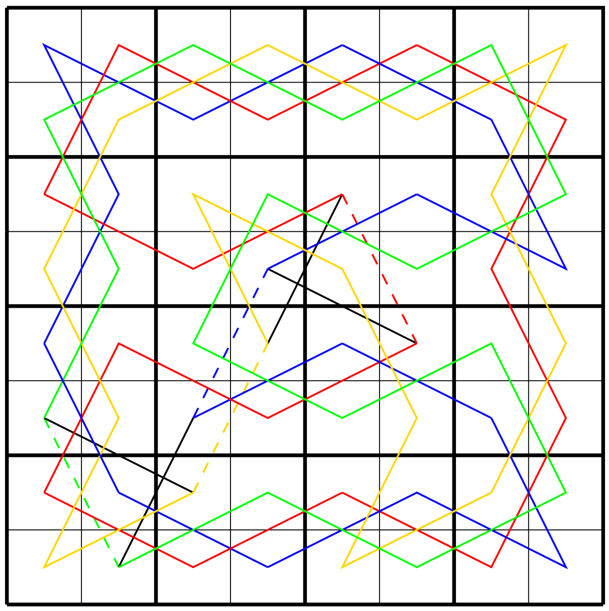


Figure 19: Details of how the four straight-move circuits are combined; the straight moves indicated by dotted lines are replaced by diagonal moves indicated by solid black lines.)



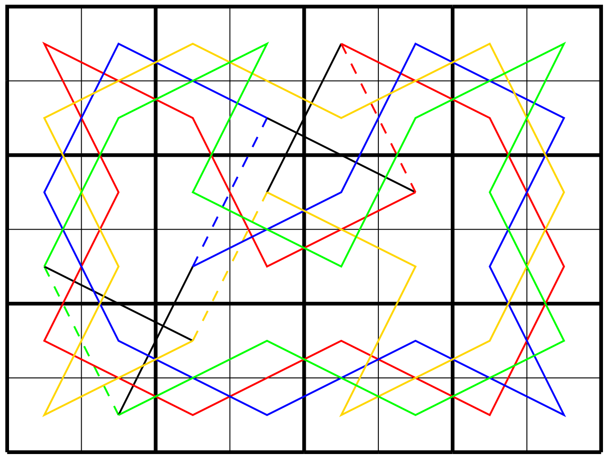


Figure 20: Knight's Circuit of an 8×6 and an 8×8 board. (Dotted lines are not part of the circuit; these are the moves that are replaced by diagonal moves, as detailed in fig. 19.)

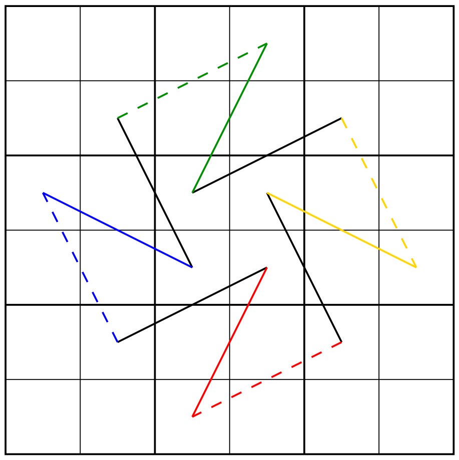


Figure 21: Details of Combining Circuits. Diagonal moves are shown in black. Straight moves are coloured. The dotted lines represent the moves that are replaced. Solid lines represent the moves that replace them.

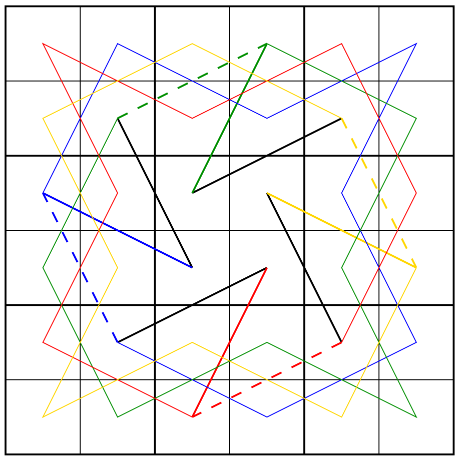


Figure 22: Knight's Circuit of a 6×6 board. The dotted lines do not form part of the circuit.

Bibliography

[Bac03] Roland Backhouse. *Program Construction. Calculating Implementations From Specifications.* John Wiley Sons, Ltd., 2003.

[Bac07] Roland Backhouse. The torch problem. Available at http://www.cs.nott.ac.uk/ rcb/MPC/papers, August 2007.

[BCG82] Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways*, volume I and II. Academic Press, 1982.

[BF01] Roland Backhouse and Maarten Fokkinga. The associativity of equivalence and the Towers of Hanoi Problem. *Information Processing Letters*, 77:71–76, 2001.

[BL80] P. Buneman and L. Levy. The Towers of Hanoi Problem. *Information Processing Letters*, 10:243–244, 1980.

[Dij90] Edsger W. Dijkstra. EWD1083: The balance and the coins. `http://www.cs.utexas.edu/users/EWD/ewd10xx/EWD1083.PDF`, September 1990.

[Dij92] Edsger W. Dijkstra. EWD1135: The knight's tour. `http://www.cs.utexas.edu/users/EWD/ewd11xx/EWD1135.PDF`, September 1992.

[Dij97] Edsger W. Dijkstra. EWD1260: The marked coins and the scale. `http://www.cs.utexas.edu/users/EWD/ewd12xx/EWD1260.PDF`, March 1997.

[DS90] Edsger W. Dijkstra and Carel S. Scholten. *Predicate Calculus and Program Semantics.* Texts and monographs in Computer Science. Springer-Verlag, 1990.

[DW00] John P. D'Angelo and Douglas B. West. *Mathematical Thinking. Problem-Solving and Proofs.* Prentice Hall, 2000.

[Gri81] David Gries. *The Science of Programming.* Springer-Verlag, 1981.

[GS93] David Gries and Fred B. Schneider. *A Logical Approach to Discrete Math.* Springer-Verlag, 1993.

[Lev03] Anany Levitin. *Introduction to The Design And Analysis of Algorithms.* Addison Wesley, 2003.

[Rot02] Günter Rote. Crossing the bridge at night. *Bulletin of the European Association for Theoretical Computer Science*, 78:241–246, October 2002.

[Smu78] Raymond Smullyan. *What Is The Name Of This Book?* Prentice-Hall, Englewood Cliffs, N.J., 1978.

[Ste97] Ian Stewart. *The Magical Maze.* Weidenfield and Nicolson, London, 1997.

[Tar56] Alfred Tarski. *Logic, Semantics, Metamathematics, papers from 1923 to 1938.* Oxford University Press, 1956. Translated by J.H.Woodger.

[Wil87] J.G. Wiltink. A deficiency of natural deduction. *Information Processing Letters*, 25:233–234, 1987.