Principles and Applications of Algorithmic Problem Solving

João Fernando Peixoto Ferreira

Thesis submitted to The University of Nottingham
for the degree of Doctor of Philosophy

December 2010

To the memory of my father, Fernando Ferreira, whose encouragement and inspiration shaped who I am today.

Abstract

Algorithmic problem solving provides a radically new way of approaching and solving problems in general by using the advances that have been made in the basic principles of correct-by-construction algorithm design. The aim of this thesis is to provide educational material that shows how these advances can be used to support the teaching of mathematics and computing.

We rewrite material on elementary number theory and we show how the focus on the algorithmic content of the theory allows the systematisation of existing proofs and, more importantly, the construction of new knowledge in a practical and elegant way. For example, based on Euclid's algorithm, we derive a new and efficient algorithm to enumerate the positive rational numbers in two different ways, and we develop a new and constructive proof of the two-squares theorem.

Because the teaching of any subject can only be effective if the teacher has access to abundant and sufficiently varied educational material, we also include a catalogue of teaching scenarios. Teaching scenarios are fully worked out solutions to algorithmic problems together with detailed guidelines on the principles captured by the problem, how the problem is tackled, and how it is solved. Most of the scenarios have a recreational flavour and are designed to promote self-discovery by the students.

Based on the material developed, we are convinced that goal-oriented, calculational algorithmic skills can be used to enrich and reinvigorate the teaching of mathematics and computing.

Acknowledgements

Many people supported me in preparing this thesis, and I would like to express my thanks and gratitude.

First, I would like to thank my supervisor Roland Backhouse for accepting me as his PhD student and for being an excellent supervisor. His experience and impressive ability to simplify and solve problems are inspiring, and always helped me whenever I got stuck. Also, his detailed and valuable feedback on my work will surely have an impact on the rest of my career. I have learnt a great deal from Roland.

I would also like to thank Luís Soares Barbosa, my co-supervisor, for his constant support and encouragement. Luis is a dear and caring person, without whom I would never have started my PhD studies. I owe him a lot. Thanks, too, to José Nuno Oliveira, who, through his enthusiasm, inspired me to pursuit a research career.

Many people gave me useful and constructive feedback on parts of this thesis. In particular, I would like to thank my internal examiner Natasha Alechina, who accompanied my progress since the first year. Also, thanks to my external examiner Shin-Cheng Mu for all the valuable comments and corrections. I am also grateful to Wei Chen, who is a good friend, collaborator, and who is always posing me interesting problems. Thanks to Arjan Mooij, Eric Macaulay, Jeremy Weissmann, and Paula Valença for their comments and suggestions on parts of this thesis. Thanks, too, to Filipe Oliveira, who was one of my favourite teachers at university and who introduced me to number theory.

Thanks to Nocas, Marta, Hélder, and Silvino for their good advice and for making me feel special. Also, thanks to Paulo Abrantes for the support and for always trying to make me smile. Special thanks to Sarawut Jindarat, who helped me a lot when I first arrived to Nottingham.

My mother, Irene Ferreira, has always been a constant source of inspiration and love. Mãe, obrigado pela tua presença e amor constantes. Sem ti, nunca teria chegado aqui.

Finally, my deepest and biggest thanks go to my beloved wife Alexandra. Her love, support, and encouragement are what make my life worthwhile.

A Fundação para a Ciência e a Tecnologia apoiou o trabalho desenvolvido nesta tese com a bolsa de investigação SFRH/BD/24269/2005, financiada pelo POPH - QREN - Tipologia 4.1 - Formação Avançada (comparticipado pelo Fundo Social Europeu e por fundos nacionais do MCTES).

Contents

1 Introduction **1**

1.1 Algorithmic problem solving: what is it all about? . . . 1

1.2 A first example . . . 2

1.2.1 A conventional proof . . . 2

1.2.2 An algorithmic proof . . . 3

1.3 Contributions . . . 5

1.4 Related work . . . 6

1.4.1 Mathematics of program construction . . . 6

1.4.2 Calculational proofs and structured derivations . . . 8

1.4.3 Education and research on algorithmic problem solving . . . 8

1.4.4 Classical problem solving . . . 10

1.5 Structure and organisation . . . 11

2 Principles of Algorithmic Problem Solving **13**

2.1 Identifying algorithmic problems . . . 14

2.2 On the use of formalism . . . 17

2.3 Goal-oriented investigations . . . 27

2.4 On concision and avoidance of unnecessary detail . . . 32

3 Techniques for Algorithmic Problem Solving **36**

3.1 Problem decomposition . . . 37

3.2 Symmetry . . . 39

3.3 Distributivity . . . 48

3.4 Invariants . . . 52

3.5 Proving program termination . . . 57

3.6 Algorithm Inversion . . . 59

4 A Calculational and Algorithmic Approach to Elementary Number Theory **65**

4.1 Introduction . . . 65

4.2 Divisibility theory . . . 68

4.2.1 Integer division . . . 68

4.2.2 Division relation . . . 75
4.2.3 Constructing Euclid's algorithm . . . 77
4.2.4 Greatest common divisor . . . 79
4.3 Euclid's algorithm as a verification interface . . . 81
4.3.1 Exploring the invariant . . . 81
4.3.2 $\bigtriangledown$ on the left side . . . 83
4.3.3 A geometrical property . . . 85
4.4 Euclid's algorithm as a construction interface . . . 87
4.4.1 Distributivity properties . . . 87
4.4.2 Enumerating the rationals . . . 92
4.5 The theory of congruences . . . 102
4.5.1 Basic properties of congruences . . . 103
4.5.2 Modular exponentiation . . . 114
4.5.3 On a simple version of the Chinese remainder theorem . . . 117
4.6 Designing an algorithmic proof of the two-squares theorem . . . 124
4.6.1 Euclid's algorithm . . . 125
4.6.2 Inverting Euclid's algorithm . . . 128
4.6.3 Reversed sequences of vectors . . . 131
4.6.4 Length of the sequence of vectors . . . 135
4.6.5 Sum of two positive squares . . . 136
4.6.6 Discussion . . . 138
4.7 Conclusion . . . 138
4.8 Appendix: historical remarks on the trees of rationals . . . 139
4.8.1 Stern's paper . . . 140
4.8.2 Brocot, the watchmaker . . . 144
4.8.3 Conclusion . . . 147

5 Supporting the Teaching of Algorithmic Problem Solving 148
5.1 Teaching scenarios . . . 148
5.2 How to create a teaching scenario . . . 150
5.2.1 Brief description and goals . . . 150
5.2.2 Problem statement . . . 150
5.2.3 Prerequisites . . . 152
5.2.4 Resolution . . . 152
5.2.5 Notes for the teacher . . . 152
5.2.6 Extensions and exercises . . . 154
5.2.7 Further reading . . . 154

5.3 A catalogue of teaching scenarios 155

6 Conclusion **158**

6.1 Future work 161

References **163**

I Teaching Scenarios for Teaching Algorithmic Problem Solving **172**

1 Exploring Algebraic Symmetries **173**

2 Calculating Orderings Between Two Numbers **184**

3 The Island of Knights and Knaves **192**

4 Portia's Casket **203**

5 A Logical Race **216**

6 A Calculational Proof of the Handshaking Lemma **227**

7 Moving a Heavy Armchair **238**

8 Exchanging the Values of Two Variables **247**

9 The Chameleons of Camelot **263**

10 Will This Algorithm Terminate? **287**

11 Constructing Euclid's Algorithm **303**

12 The King Who Loved Diagonals **323**

CHAPTER 1

Introduction

While realizing that the solution of problems is one of the lowest forms of mathematical research, and that, in general, it has no scientific value, yet its educational value cannot be overestimated. It is the ladder by which the mind ascends into the higher fields of original research and investigation. Many dormant minds have been aroused into activity through the mastery of a single problem.

— BENJAMIN FINKEL AND JOHN M. COLAW (1894)

1.1 Algorithmic problem solving: what is it all about?

Algorithmic problem solving is about the formulation and solution of problems where the solution involves, possibly implicitly, the principles and techniques that have been developed to assist in the construction of correct *algorithms*[1]. Algorithms have been studied and developed since the beginning of civilisation, but, over the last few decades, the unprecedented scale of programming problems and the consequent demands on the reliability of computer software led to massive improvements in our algorithmic-problem-solving skills. The improvements are centred on goal-directed, calculational *construction* of algorithms as opposed to the traditional guess-and-verify methodology.

In spite of these improvements, and although much of mathematics is algorithmic in nature[2], the skills needed to formulate and solve algorithmic problems do not form an integral part of mathematics education. Also, the teaching of computer-related topics at pre-university level focuses on enabling students to be effective users of information

[1] An algorithm is a finite sequence of instructions that can be systematically executed in the solution of a given problem.

[2] When we say that mathematics is algorithmic in nature, we *do not* mean that we have an algorithm to do mathematics. Instead, we want to say that the principles and techniques that have been developed to formulate and solve algorithmic problems can be used to solve many mathematical problems.

technology, rather than equip them with the skills to develop new applications or to solve new problems. In this thesis, we argue that this situation should change.

One of our main claims is that goal-oriented, calculational algorithmic skills can be used to enrich and reinvigorate the teaching of mathematics and computing. As a result, the main contribution of this thesis is educational material that supports that claim. The material is problem-driven and it is aimed at the boundary between pre-university and university level. Along the way, we also elaborate on some principles and techniques that can be used to solve algorithmic problems more effectively. In section 1.3, we explain in more detail the contributions of this thesis. But first, we show an example that may help to understand the nature of algorithms and their relation to problem solving.

1.2 A first example

A concrete example may help to understand better how algorithmic techniques can be used to do mathematics in a precise and effective way. We consider a well-known theorem on the Fibonacci numbers. The theorem and the first proof we show were taken from the book [Bur05, p. 286], where the Fibonacci sequence is defined as follows[3]:

$$\mathsf{fib}.1 = \mathsf{fib}.2 = 1 \qquad\qquad \mathsf{fib}.n \;=\; \mathsf{fib}.(n-1) + \mathsf{fib}.(n-2) \quad \text{for } n \geq 3 \quad .$$

The theorem states that any two consecutive Fibonacci numbers are coprime, that is, their greatest common divisor (gcd) is 1:

Theorem 1.2.1 For the Fibonacci sequence, $\mathsf{gcd}(\mathsf{fib}.n, \mathsf{fib}.(n+1)) = 1$ for every $n \geq 1$.

□

You may want to prove the theorem before reading further. We first show the proof taken from [Bur05], which, we believe, is representative of the conventional style in which these proofs are shown to students. We label it as "conventional proof". We then show how we can use an algorithm and the notion of invariance to prove it.

1.2.1 A conventional proof

Let us suppose that the integer $d > 1$ divides both $\mathsf{fib}.n$ and $\mathsf{fib}.(n+1)$. Then their difference $\mathsf{fib}.(n+1) - \mathsf{fib}.n \;=\; \mathsf{fib}.(n-1)$ is also divisible by d. From this and from the relation

[3] In [Bur05], the n^{th} Fibonacci number is denoted as u_n. We denote it as $\mathsf{fib}.n$; the name fib is more informative and we use an infix dot for function application.

$\mathsf{fib}.n - \mathsf{fib}.(n-1) = \mathsf{fib}.(n-2)$, it may be concluded that $d \setminus \mathsf{fib}.(n-2)$. Working backward, the same argument shows that $d \setminus \mathsf{fib}.(n-3)$, $d \setminus \mathsf{fib}.(n-4)$, ..., and finally that $d \setminus \mathsf{fib}.1$. But $\mathsf{fib}.1 = 1$, which is certainly not divisible by any $d > 1$. This contradiction ends our proof.

Some comments on the conventional proof This proof captures several aspects of conventional mathematical method. First, it uses mostly natural language to express the connection between the steps. Second, it is based on implication rather than equality (the use of natural language usually forces one to use the connective "then", which corresponds to implication). Third, it is a proof by contradiction: we start by assuming that d is a common divisor of $\mathsf{fib}.n$ and $\mathsf{fib}.(n+1)$ at least 2, but we reach the contradiction that d has to divide 1. Other conventional aspects are the use of a prefix notation to the associative gcd operator and the use of the so-called *dot-dot-dot notation* (...), which reveals some imprecision in the argument. (By using a prefix notation for gcd, the author forces a syntactic distinction between the equivalent expressions $\mathsf{gcd}(m, \mathsf{gcd}(n, p))$ and $\mathsf{gcd}(\mathsf{gcd}(m, n), p)$. In what follows, we change to an infix notation.) Finally, one could also argue that the proof is not at all clear about the properties being used (e.g., which property is the author using when he writes "the difference is also divisible"?).

In this thesis, we will discuss most of these conventional aspects and we will propose some alternatives that we think are better. We will, for example, avoid the use of natural language to connect steps in our arguments, and we will use a systematic proof format that allows us to be more precise about the properties that we use.

1.2.2 An algorithmic proof

More than two thousand years ago (c. 300 B.C.), in Book 7, Propositions 1 and 2, of his seminal mathematical treatise *Elements* [HE56, p. 296], Euclid has given a method to compute the greatest common divisor of two positive numbers[4]. That method is now known as Euclid's algorithm and, using the Guarded Command Language (GCL), we can formulate it as:

$$\{\ 0 < m\ \wedge\ 0 < n\ \}$$

$$x, y\ :=\ m, n\ ;$$

$$\{\ \textbf{Invariant}\colon\quad m\ \mathsf{gcd}\ n = x\ \mathsf{gcd}\ y\ \}$$

[4] According to Donald Knuth [Knu97, p. 334], some scholars believe that the method was known up to 200 years earlier. Knuth also writes that we might call Euclid's method the granddaddy of all algorithms, because it is the oldest nontrivial algorithm that has survived to the present day.

$$
\begin{array}{l}
\mathsf{do}\ \ y < x \ \rightarrow\ x \ :=\ x - y \\
\Box\ \ \ x < y \ \rightarrow\ y \ :=\ y - x \\
\mathsf{od} \\
\{\ \ x = y \ \ \wedge \ \ \ m \ \mathsf{gcd}\ n = x = y\ \ \}\ .
\end{array}
$$

The algorithm maintains two variables, x and y, which are set initially to be m and n, respectively. The do $\cdots$ od statement is a loop that executes while one of the guards ($y < x$ and $x < y$) is true. The algorithm stops when $x = y$. If $y < x$, x is decreased by y. If $x < y$, y is decreased by x. Expressions in curly brackets are assertions. The first assertion, usually called precondition, states that the arguments of the algorithm, m and n, have to be positive numbers; the last assertion, usually called postcondition, states that, on termination, the value of x and y is the greatest common divisor of the arguments m and n. The assertion in the middle expresses that the value of $m \ \mathsf{gcd}\ n = x \ \mathsf{gcd}\ y$ is an invariant of the loop body, that is, it is true throughout the execution of the algorithm. In other words, it is true initially and true after each iteration of the loop body. As a result, it will be true on termination when $x = y$. Since the greatest common divisor is idempotent ($m \ \mathsf{gcd}\ m = m$, for all m), we can conclude that the greatest common divisor of m and n is x (or y, since they are equal).

Because the theorem that we want to prove involves computing the greatest common divisor of two positive numbers, it seems sensible to investigate whether we can use Euclid's algorithm to prove it. Indeed we can. We first observe that the theorem states that if the initial value of the variables x and y are two consecutive Fibonacci numbers, their final value is 1. Suppose then that x and y are two consecutive Fibonacci numbers. Comparing the two assignments in the loop body with the definition of the Fibonacci sequence, an invariant is immediately suggested: *x and y are two consecutive Fibonacci numbers*. The proof is an immediate consequence of the Fibonacci definition. This means that we can refine the algorithm shown above as follows:

$$
\begin{array}{l}
x\,,y \ :=\ \mathsf{fib}.(n{+}1)\,,\mathsf{fib}.n\ \ ; \\
\{\ \textbf{Invariant}\colon \qquad x \text{ and } y \text{ are two consecutive Fibonacci numbers} \\
\qquad\qquad\qquad\quad \wedge\ \ \mathsf{fib}.(n{+}1) \ \mathsf{gcd}\ \mathsf{fib}.n \ =\ x \ \mathsf{gcd}\ y\ \ \} \\
\mathsf{do}\ \ y < x \ \rightarrow\ x \ :=\ x - y \\
\Box\ \ \ x < y \ \rightarrow\ y \ :=\ y - x \\
\mathsf{od} \\
\{\ \qquad x \text{ and } y \text{ are two consecutive Fibonacci numbers } \ \wedge\ \ x = y \\
\quad \wedge\ \ \mathsf{fib}.(n{+}1) \ \mathsf{gcd}\ \mathsf{fib}.n \ =\ x \ =\ y\ \ \}\ .
\end{array}
$$

The goal is to prove that $\mathsf{fib}.(n{+}1)\ \mathsf{gcd}\ \mathsf{fib}.n = 1$, i.e., the final value of the variables x and y is 1. Observing that the Fibonacci sequence is increasing, we can simplify part of the postcondition as follows:

$$
\begin{array}{ll}
 & x \text{ and } y \text{ are two consecutive Fibonacci numbers } \ \wedge \ x = y \\
= & \quad \{ \quad \text{there are only two equal consecutive Fibonacci numbers:} \\
 & \qquad \text{fib.1 and fib.2, which are both 1} \quad \} \\
 & x = y = 1 \ .
\end{array}
$$

This proves that on termination the value of $\mathsf{fib}.(n{+}1)\ \mathsf{gcd}\ \mathsf{fib}.n$ is 1. Because Euclid's algorithm always terminates, we conclude that the following algorithm establishes theorem 1.2.1:

$$
\begin{array}{l}
x\,,y \ := \ \mathsf{fib}.(n{+}1)\,,\mathsf{fib}.n \ ; \\
\{ \ \textbf{Invariant}: \quad x \text{ and } y \text{ are two consecutive Fibonacci numbers} \\
\qquad\qquad \wedge \ \ \mathsf{fib}.(n{+}1)\ \mathsf{gcd}\ \mathsf{fib}.n \ = \ x\ \mathsf{gcd}\ y \ \ \} \\
\mathsf{do} \ \ y < x \ \rightarrow \ x \ := \ x{-}y \\
\Box \ \ x < y \ \rightarrow \ y \ := \ y{-}x \\
\mathsf{od} \\
\{ \ x = y = 1 \ \wedge \ \mathsf{fib}.(n{+}1)\ \mathsf{gcd}\ \mathsf{fib}.n \ = \ 1 \ = \ 1 \ \}
\end{array}
$$

Although this proof based on Euclid's algorithm may seem more complex than the one labelled as conventional, we will show in this thesis that we can systematise it. We hope to convince the reader that the emphasis on algorithmic skills and techniques can indeed be used to reinvigorate mathematics education!

1.3 Contributions

The main contribution of this thesis is educational material, capturing calculational and algorithmic problem-solving techniques, that supports the teaching of mathematics and computing. The material is problem-driven, it is aimed at the boundary between pre-university and university level, and it is divided in two main parts.

First, in chapter 4, we show how a fresh approach to introductory number theory that focuses on the algorithmic content of the theory can combine practicality with mathematical elegance. We prove both old and well-known, and new and previously un-

known, theorems related with the greatest common divisor and rational numbers. For example, based on Euclid's algorithm, we derive the following new results: we calculate sufficient conditions for a natural-valued function to distribute over the greatest common divisor, we derive an efficient algorithm to enumerate the positive rational numbers in two different ways, and we develop a new and constructive proof of the two-squares theorem.

We believe that the material on number theory that we have developed can be used to support a course on elementary number theory. Nevertheless, and although the material shown contains educational remarks, we are convinced that the teaching of algorithmic problem solving is more effective if the teacher has access to detailed guidelines on how to solve and present specific algorithmic problems. Towards that end, we propose the introduction of educational material in the form of *teaching scenarios*, which are fully worked out solutions to algorithmic problems together with detailed guidelines on the principles captured by the problem, how the problem is tackled, and how it is solved. So, the second part of the material is a set of teaching scenarios that illustrate the principles and techniques discussed in this thesis. The scenarios are example-driven and have a recreational flavour, making them especially suitable for extra-curricular math clubs. Although they can be directly used by the students, they are primarily written for the teacher. Moreover, they are designed to promote self-discovery, since we believe that the success of teaching depends on the amount of discovery that is left for the students: if the teacher discloses all the information needed to solve a problem, students act only as spectators and become discouraged; if the teacher leaves all the work to the students, they may find the problem too difficult and become discouraged too. Scenarios are designed to maintain a balance between these two extremes. Some of the problems and solutions shown in the teaching scenarios are not new, but we capture them in a new and accessible way: as a catalogue of problems and solutions having a consistent format.

1.4 Related work

1.4.1 Mathematics of program construction

Many principles and ideas discussed in this thesis were created or developed by computing scientists working in the area of *mathematics of program construction*. Adopting a simplistic view, we can say that this area started in the 1960s, when programmers started recognising that there were serious problems in the programming field and that it was necessary to prove the correctness of programs. At the time, software engi-

neering was facing a software crisis and programming was not very well understood. Many software projects ran over budget and schedule, and some of them even caused property damage and loss of life[5]. (Note that, fifty years later, many still do.)

To solve these problems, computer scientists focused on programming methodology and on ways to build programs in a systematic way. A common consensus was that programs should be proved correct, and in the late 1960s, a number of landmark papers had an important impact on the field. For example, in 1968, Edsger W. Dijkstra published an article [Dij68] on the harmfulness of the Go To statement, where he claims that its use makes it impossible to determine the progress of a program. Also, one year later, Tony Hoare published a seminal article [Hoa69] where he introduces what is now known as Hoare triples and an axiomatic approach to language definition.

Although Hoare's theory had a great impact, it was quite difficult to use it to prove existing programs correct, since one was forced to find an invariant for each loop. Programmers started studying alternatives, and the most plausible one was to develop the program together with its proof. Some years later, in 1975, Edsger W. Dijkstra published a paper [Dij75] where he introduced weakest preconditions. One year later he published a book [Dij76] and showed how to use weakest preconditions as a "calculus for the derivation of programs". Programmers were now able to build programs in a more reliable and systematic way, and the *art* of programming became more and more a *discipline* of programming.

From here, Dijkstra and others, dedicated themselves to the *mathematization* of programming and to the methodology of program derivation. As the programs were becoming more and more complicated, the solutions were becoming less and less simple and beautiful. According to [Fei87, page 9], the reason was the "standard mathematical reasoning patterns", which were not suitable for the task at hand. The conclusion was that computer scientists would have to learn how to construct proofs more effectively, in order to solve more ambitious problems. This was the beginning of a new period: computer scientists started to investigate ways of streamlining mathematical arguments. Mathematics and mathematicians were now faced with this new side of Computing Science.

During this new period, several problems were identified with traditional mathematics. One of the first problems was that mathematicians hardly manipulate their formulae: instead, they interpret them; and one of the reasons is that the notation they use is not adequate for manipulation. This and other observations were also presented in the

[5] A list of accidents caused by computer programs can be found at "The Risks Digest" (`http://catless.ncl.ac.uk/Risks`)

PhD thesis of Netty van Gasteren "On the shape of mathematical arguments" [vG90], where she presents a study about proofs (proofs of correctness of programs included). In particular, in chapter 11 of her thesis, she shows how Euclid's algorithm can be used to prove theorems about the greatest common divisor of two numbers. In this thesis we expand substantially the material shown in that chapter.

In a way, this project is a continuation of Van Gasteren's study, but while she did a broad study, we are concerned specifically with algorithmic problems: construction of new algorithms and usage of algorithmic skills to demystify mathematical invention.

1.4.2 Calculational proofs and structured derivations

One of the products of the attempts to streamline mathematical arguments mentioned above is the so-called calculational method, which aims at reducing proofs as much as possible to elementary syntactic calculation. The calculational proof style has been adopted widely by the community of computing scientists working on formal programming methods. For example, the textbooks written by Roland Backhouse [Bac03], by Richard Bird and Oege de Moor [BdM96], by David Gries and Fred Schneider [GS93], and by Jan van de Snepscheut [vdS93] are well-known examples.

Gries and Schneider have also studied the use of calculational proofs for teaching mathematics [GS95]. Also, Back *et al.* introduced the concept of *structured derivations* [BGvW96, BvW97, BPSvW04, Bac09, BvW06], which is a further development of the calculational proof style created by Feijen and Dijkstra. Essentially, structured derivations add a mechanism for doing subderivations and for handling assumptions in proofs. Moreover, in 2001, they have initiated a study in Finland to investigate whether structured derivations could be used to integrate logic, proof and formal reasoning throughout secondary-school mathematics education [BvW06]. The results were positive and the test group outperformed the control group.

Two of the main resources on the calculational method are the websites "E. W. Dijkstra Archive" (`http://userweb.cs.utexas.edu/users/EWD`) and "mathmeth.com – Discipline in Thought" (`http://mathmeth.com`).

1.4.3 Education and research on algorithmic problem solving

Computing science is all about solving algorithmic problems (or, as some authors prefer to say, it is all about instructing computers to solve problems). Below, we briefly survey related projects that aim at using and improving algorithmic skills and tech-

niques.

Algorithmic Problem Solving The algorithmic problem solving research group at the University of Nottingham conducts research into mathematical method, in particular the problem-solving skills involved in the formulation and solution of algorithmic problems. It also offers a module entitled "Algorithmic Problem Solving" to first-year undergraduates in computing science. As Roland Backhouse explains in [Bac06], the name of the module is deliberately ambiguous. Parsed as algorithmic-problem solving, it is about solving problems that involve the construction of an algorithm for their solution. Parsed as algorithmic problem-solving, it is about problem solving in general, using the principles that have been learnt in the development of correct-by-construction algorithm-design techniques.

The material included in this thesis was developed in the context of the group's research plan. It was also done in the context of MATHIS, a project that aims to reinvigorate secondary-school mathematics by exploiting insights of the dynamics of algorithmic problem solving [FMBB09].

Computer Science Unplugged Our work is related to the work developed within the project "Computer Science Unplugged" [BWF06], whose goal is to teach principles of computing science through games and puzzles. They provide a series of activity worksheets that can be directly used in the classroom. These worksheets are similar to the teaching scenarios that we propose, but their goals are slightly different: whilst they want to convey general principles and ideas of computing, we want to focus on calculational and algorithmic principles and techniques that can be used to reinvigorate mathematics. Also, their material is suitable for people of all ages and the material shown in this thesis is aimed at the boundary between pre-university and university level.

The project was started by Tim Bell, Mike Fellows and Ian Witten, and is now being explored by several dozen contributors working in many countries (including New Zealand, USA, Sweden, Australia, China, Korea, Taiwan and Canada). Additional information about the project can be found at the website `http://csunplugged.org`.

Computational thinking We can say that our work fits with what is now usually called "computational thinking" [Win06]. We, too, want to transfer skills created and developed within computing science and we want to illustrate the value of computational thinking to everyone interested in problem solving. In particular, we believe

that mathematics education can be reinvigorated by exploring the algorithmic nature of much of its contents. Research in computational thinking is being led by the Center of Computational Thinking at Carnegie Mellon where their major activity is conducting PROBEs (PROBlem-oriented Explorations). These PROBEs are experiments that apply novel computing concepts to problems to show the value of computational thinking. It is worth mentioning that, currently, there is not any PROBE on mathematics education.

1.4.4 Classical problem solving

We cannot conclude this section without mentioning some of the efforts of the mathematical community in articulating psychological and technical approaches to problem solving. The best known work, and one of the major influences in problem solving, is George Pólya's *How to Solve It*. Even though other mathematicians had considered questions of problem solving in earlier generations, it was Pólya's *How to Solve It* that had the tremendous impact on the way people viewed the techniques of attacking mathematical problems. In particular, it was there that Pólya suggested the division of the problem-solving process into the now widely accepted four phases: *understanding the problem*, *devising a plan*, *carrying out the plan*, and *looking back*. Other works include Jacques Hadamard's *Essay on the Psychology of Invention in the Mathematical Field*, Karl Duncker's *On Problem Solving*, and Max Wertheimer's *Productive Thinking*. Incidentally, all these books appeared in 1945, which led Alan H. Schoenfeld to point out that 1945 was indeed a great year for problem solving.

Another trend in mathematics and problem solving — and related with the opening quote of this chapter — is based on the belief that mathematical games and recreations can be used for educational purposes and incorporated into various curricula. Recreational problems have been used for many centuries. The *Rhind Mathematical Papyrus* shown in figure 1.1, for example, shows that Egyptian mathematicians were interested in puzzle type problems. Although most of the problems are on division, weights, measures, and rational numbers, the papyrus also includes the famous multiple-of-seven riddle, rewritten in the Medieval era as the nursery rhyme "As I was going to St. Ives", whose common modern version is:

> As I was going to St Ives
> I met a man with seven wives
> Each wife had seven sacks
> Each sack had seven cats
> Each cat had seven kits

Kits, cats, sacks, wives
How many were going to St Ives?

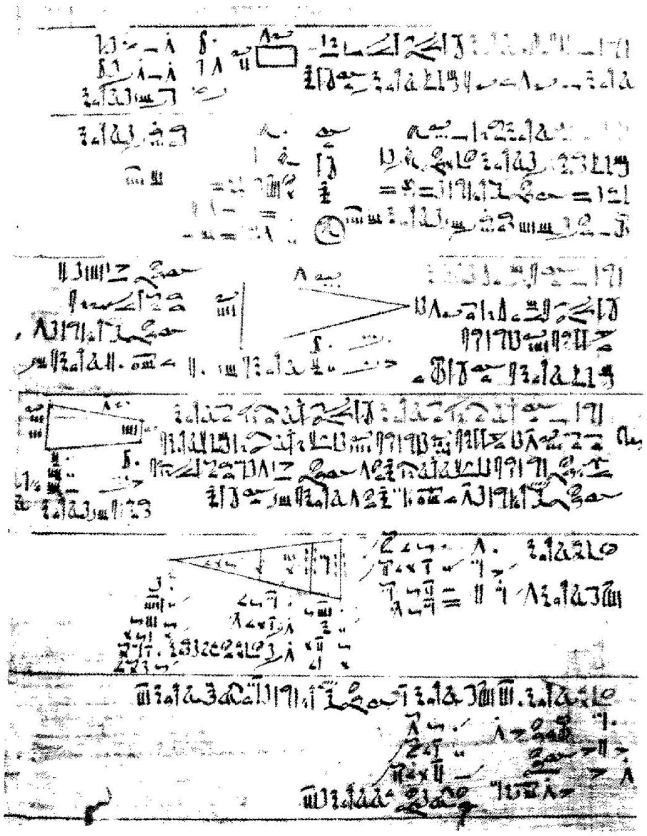


Figure 1.1: The Rhind Mathematical Papyrus, written around 1650 B.C.

One of the contemporary leaders of this trend was Martin Gardner, who published thousands of puzzles in his books and various journals. Also, Zbigniew and Matthew Michalewicz recently published the book *Puzzle-Based Learning: An introduction to critical thinking, mathematics, and problem solving* [MM08], where they support the teaching of problem solving based on puzzles that are inherently unstructured. We are sympathetic with their view and we also support a problem-driven approach to algorithmic problem solving. For example, most of the problems discussed in the teaching scenarios are recreational.

1.5 Structure and organisation

The goal-oriented, calculational algorithmic skills that we believe can be used to enrich and reinvigorate the teaching of mathematics and computing are presented and discussed in chapters 2 and 3. In chapter 2, we discuss *principles* of algorithmic problem

solving, that is, general rules that we think should be used whenever solving (algorithmic) problems. In chapter 3, we discuss *techniques* for algorithmic problem solving. The main difference between the two is that, whilst principles apply to all problems, the same problem can be solved using different techniques. To make these chapters accessible to a wider audience, we illustrate the principles and techniques with simple and recreational examples. The material shown in these two chapters is essentially a summary of some relevant material that the community of "mathematics and program construction" has been developing.

To prove that these principles and techniques can be used to do mathematics in a practical way, we use them in chapter 4 to rewrite some material on elementary number theory. We prove both old and well-known, and new and previously unknown, theorems.

We think that the best way to convey our message is to provide abundant and sufficiently varied educational material. So, in chapter 5, we propose the introduction of educational material in the form of teaching scenarios, we explain how we think teaching scenarios should be constructed, and we describe a catalogue of scenarios that is included in appendix I. The catalogue in appendix suggests problems that can be used to illustrate the principles and techniques discussed in this thesis.

Finally, in chapter 6, we discuss what was achieved and we suggest some directions for future work.

CHAPTER 2

Principles of Algorithmic Problem Solving

A modern mathematical proof is not very different from a modern machine, or a modern test setup: the simple fundamental principles are hidden and almost invisible under a mass of technical details.

— HERMANN WEYL (1932)

This chapter discusses some important principles of algorithmic problem solving. All the principles are introduced via simple examples, to make the material accessible to a wider audience.

We start, in section 2.1, by distinguishing three main types of problems that can be solved using algorithmic techniques. Before using any of the techniques described in chapter 3, it is useful to know which kind of problem we have.

In section 2.2, we argue that precision can only be achieved if we use effective formalisms that support our reasoning processes. An example of such a formalism is the calculational method and its proof format, which allow us to conveniently record and justify every step in our arguments. As remarked earlier, the development of computing science over the last few decades led to improvements in our problem-solving skills. These improvements are centred on goal-directed and calculational *construction* of algorithms as opposed to the traditional guess-and-verify methodology. Section 2.2 explains what we mean by *calculational* constructions. In section 2.3, we explain what we mean by *goal-oriented* constructions.

We conclude the chapter with section 2.4, where we discuss the importance of naming the elements of a problem. We hope to convince the reader that the combination of concision and precision is a prerequisite for effective problem solving.

2.1 Identifying algorithmic problems

The focus of this dissertation are problems of algorithmic nature. It is only natural that one of the first sections is on the identification of such problems. We usually say that a problem is algorithmic when its solution involves the construction of an algorithm. 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. We do not know any algorithm to determine if a given problem is an algorithmic problem. However, there are some types of problems that can be immediately recognised as algorithmic. The goal of this section is to discuss some of them. For example, consider the following problem:

> **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?

Implicit in the problem statement is that all the four elements are at the same riverbank and that the farmer has to accompany each of the other elements when crossing the river. This is clearly an algorithmic problem, because the solution consists of a *sequence of instructions* indicating who or what should cross. A typical instruction would be: "the farmer crosses with the wolf" or "the farmer returns alone".

There are many algorithmic problems in our daily lives: how to dress up in the correct order, how to cook a certain dish, how to tie a shoelace, how to reach a certain destination quicker, and so on. This would suggest that we are naturally excellent algorithmic problem solvers. However, we have learnt and practised these routine algorithms with never thinking about the underlying principles or techniques. Not surprisingly, when confronted with new algorithmic problems, most people do not exploit the connection.

Note, however, that some algorithmic problems do not require us to construct an algorithm. Instead, an algorithm is provided and we are required to prove some of its properties. For example, our catalogue of teaching scenarios includes the following problem (scenario 9):

The Chameleons of Camelot

On the island of Camelot there are three different types of chameleons: grey chameleons, brown chameleons, and crimson chameleons. Whenever two chameleons of different colours meet, they both change colour to the third colour.

For which number of grey, brown, and crimson chameleons is it possible to arrange a succession of meetings that results in all the chameleons displaying the same colour?

This is an algorithmic problem, because there is an underlying algorithm that organises meetings between chameleons. The goal is to know for which initial numbers of chameleons it is possible to organise meetings that result in one single monochromatic colony of chameleons. To solve the problem, we have to identify relevant properties of the algorithm that can be used to characterise the initial number of chameleons for which the goal is possible. Therefore, it is desirable to model the algorithm in a way that facilitates the inference of properties — that is where a formal approach is beneficial, as we hope to demonstrate later.

A domain that has many algorithmic problems is mathematics. For example, there are theorems, usually called *existence theorems*, which assert the existence of certain objects. So, suppose that a theorem states that there exists a value x that satisfies a property Q. There are two different ways of establishing the theorem: the *constructive* and the *nonconstructive* proof. In a constructive proof, we design an algorithm that constructs a value x guaranteed to satisfy Q. We call such an x a *witness*. In a nonconstructive proof, we prove the theorem without constructing a witness. Usually, we achieve that by translating the problem into a counting problem, and we conclude something of the form "there is at least one object satisfying property Q". In section 4.3.2, we deal with an existence theorem that states that the greatest common divisor of two numbers m and n can be written as a linear combination of m and n. More formally, we express the theorem as:

$$\langle \exists a, b :: \ m \ \mathsf{gcd} \ n \ = \ m \times a + n \times b \rangle \quad .$$

The notation will be explained later; for now, we just need to understand that it is stating that there exist two numbers, a and b, such that

$$m \ \mathsf{gcd} \ n = m \times a + n \times b \quad .$$

Given m and n, a constructive argument computes the two witnesses, a and b, that satisfy the requirement. But we have already seen that there is an algorithm — Euclid's

algorithm — that computes m gcd n. A reasonable strategy is to change or extend the algorithm to compute the witnesses a and b. That is indeed what we do in section 4.3.2.

From the perspective of a computing scientist, constructive arguments are attractive: they give more (i.e., the witnesses), they can be automated, and their design can use the advances that have been made in our understanding of the basic principles of algorithm development over the last few decades. Moreover, they can be simpler and more useful than counting arguments — it is useful to know, for example, that we can write m gcd n as a linear combination of m and n, but if we want to compute the witnesses, a counting argument is useless.

It is not difficult to see that any existence theorem can, in principle, be turned into an algorithmic problem. However, not all theorems in mathematics are existential. So a question that arises is: are there any other types of theorems that we can solve using algorithmic techniques? Consider, for example, theorem 1.2.1 that we have proved in chapter 1. It states that

$$\text{For the Fibonacci sequence, fib.}(n+1) \text{ gcd fib.}n \;=\; 1 \;\text{ for every } n \geq 1 \quad .$$

We have shown how to use Euclid's algorithm (and the notion of invariant) to prove the theorem: we have assumed that the initial arguments of the algorithm were two consecutive Fibonacci numbers, and we have concluded that the algorithm would always compute as their greatest common divisor the value 1. The use of Euclid's algorithm in this case is well-justified, since the theorem is about greatest common divisors. In fact, we will see (in section 4.3) that many theorems related with greatest common divisors can be proved using Euclid's algorithm.

Finally, there are some problems that can be solved by using algorithms seemingly unrelated with the original statement. For example, there is a well-known theorem in number theory that characterises the numbers that can be expressed as the sum of two positive integer squares (the number 5, for example, can be written as 1^2+2^2; on the other hand, the number 6 can not be expressed as the sum of two squares). In section 4.6, we prove this theorem using the algorithm developed to prove the existential proof about the greatest common divisor mentioned above. The relation between this theorem and the algorithm is not straightforward and requires details that we do not want to include in this section. Nevertheless, this example illustrates that some problems can be transformed into specific algorithmic problems in a surprising way.

The conclusion of this section is that algorithmic problems can take different shapes. Some ask directly for a sequence of instructions, like the problem "Goat, Cabbage and Wolf"; others ask us to establish properties about given algorithms, like the problem

"Chameleons of Camelot"; others are about establishing the existence of values satisfying certain properties, like the existential theorem about the greatest common divisor; and, finally, other problems can be solved by using algorithms seemingly unrelated with the original statement, like the problem on writing a number as the sum of two squares. The only way we know to gain proficiency in identifying (and solving) algorithmic problems is by practising.

2.2 On the use of formalism

The demands on the reliability and precision of computer software led computing scientists to develop formalisms where programs are, essentially, mathematical formulae. From this perspective, the statement that a program meets a functional specification is a mathematical theorem. However, the complexity of non-trivial programs leads to long formulae, which are difficult to interpret without error or loss of precision. Therefore, besides concision and precision, manipulability of mathematical formulae becomes important to reason about programs. In particular, we are interested in manipulability *without interpretation*, because we want reasoning about programs to be as simple as possible. Moreover, reducing calculations to elementary syntactic manipulation helps to avoid errors, since all the steps are justified by previously established syntactic rules that are easy to check.

The mathematical method of reducing proofs as much as possible to elementary syntactic calculation is usually called the *calculational method* [IPL95]. In this section, we briefly discuss some aspects of the method. We start with the relevance of notation and the proof format, and we illustrate, with two examples, how an emphasis on a calculational and equational logic can be used to rewrite and reinvigorate school mathematics. We conclude the section with a simple example of how the calculational method can support formal manipulation of algorithms. For more details and considerations on the use of formalism, we recommend the reader the chapter 16 of [vG90] (some of the observations contained in this section were taken from that chapter).

Relevance of notation If we want to manipulate formulae without interpretation, we have to rely on the symbols written down. As a result, the notations we use become an important and technical issue: clumsy notations can hinder our reasoning, whilst well-designed notations geared to our manipulative needs can help us establishing new and surprising results.

By using notations geared to manipulation, rather than using them only for descriptive

purposes, we sometimes deviate from conventional practice. We have already shown such an instance in chapter 1, when we use an infix notation for the gcd operator. In general, whenever we have an associative operator op, we use infix notation. It is much nicer to write

$$a \;\mathsf{op}\; b \;\mathsf{op}\; c$$

than

$$\mathsf{op}(a, \mathsf{op}(b, c)) \quad \text{or} \quad \mathsf{op}(\mathsf{op}(a, b), c) \quad .$$

Moreover, the second notation forces a totally irrelevant choice that results in calculations longer than necessary, because we have to include steps that pass from one choice to the other.

Another example that is even more distant from conventional practice is the introduction of a new operator to exploit the associativity of Boolean equality. Whenever we want to use an associative reading, we use the symbol $\equiv$ (we call it equivales). On the other hand, when we want to use a conjunctive reading, we use the symbol $=$ (this is the conventional symbol for equality). So, for example, if we have the expression

$$\mathsf{false} \equiv \mathsf{false} \equiv \mathsf{true} \quad ,$$

we can evaluate it as

$$(\mathsf{false} \equiv \mathsf{false}) \equiv \mathsf{true} \quad .$$

Because $\mathsf{false} \equiv \mathsf{false}$ and $\mathsf{true} \equiv \mathsf{true}$ are both true, we can conclude that the value of this continued equivalence is true. However, if we have the expression

$$\mathsf{false} = \mathsf{false} = \mathsf{true} \quad ,$$

we evaluate it conjunctively:

$$\mathsf{false} = \mathsf{false} \;\wedge\; \mathsf{false} = \mathsf{true} \quad .$$

Because $\mathsf{false} = \mathsf{true}$ is false and false is the zero of conjunction, the value of this continued equality is false. This shows that the associative and conjunctive readings conflict. That is why we introduce a new operator for the less conventional associative semantics. (The introduction of a new operator avoids context-dependent parsing. In page 146 of her thesis, Netty van Gasteren uses the expression “context-dependent parsing" to denote parsing that depends on the type of the operands, but we think it can be extended to include parsing that depends on the semantic context as well.)

Mathematicians usually use the symbol $\Leftrightarrow$ to denote Boolean equality, and they read it as "if and only if". We avoid this notation because it is normally associated with mutual implication, whereas we want to highlight equality and substitution of equals for equals.

Another example of a non-conventional notation that we use is the so-called Eindhoven quantifier notation. Unlike the traditional mathematical notation for quantifiers, the Eindhoven notation clearly identifies all the relevant parts of the quantification (dummies, range, and term). Because it is more uniform, we can apply the same calculational rules to different quantifiers. We use and explain the Eindhoven quantifier notation in section 2.3, in chapter 4, and scenario 6. For more details about quantifiers, we recommend [Bac03, Chapter 11] and [BM06].

Finally, sometimes it is desirable to introduce new notations that are especially suited for the algebraic properties involved in a given problem. An example is the problem shown in exercise 5.6.2 of the scenario 5 (page 224), where we are required to express that exactly one of three propositions is true. Given three propositions P, Q, and R, we could introduce a binary operator $\odot$ to denote that exactly one of them is true

(2.2.1) $P \odot Q \odot R$.

However, there is not any binary Boolean operator that can be used to express that exactly one of three propositions is true (see [Fer09b] and [Fer09a] for more details).

As a result, we introduce a new *bracket notation*, i.e., we write (2.2.1) as:

$$\langle\!\langle P,Q,R \rangle\!\rangle \quad .$$

This notation is easy to use and to handwrite, it delimits well the list of propositions, and it makes it simple to express the relevant algebraic properties that are used to solve the problem:

$$p \wedge \langle\!\langle q_0,q_1,\cdots,q_n \rangle\!\rangle = \langle\!\langle p \wedge q_0 \ , \ p \wedge q_1 \ , \ \cdots \ , \ p \wedge q_n \rangle\!\rangle \quad ,$$

$$\langle\!\langle p \rangle\!\rangle = p \quad \text{, and}$$

$$\langle\!\langle p, \mathsf{false} \rangle\!\rangle = p = \langle\!\langle \mathsf{false}, p \rangle\!\rangle \quad .$$

Relevance of the proof format A proof of a theorem should demonstrate, using certain facts (also known as axioms) or previously proved theorems, why it is true. Additionally, a good proof should explain clearly how the facts are combined and it should express the design considerations so that readers can understand it better, explain it to others, and prove other theorems in a similar fashion.

For example, suppose that we want to write a three-step proof of the statement $A \Leftarrow D$, where[1]

- the first step establishes $A = B$;
- the second step establishes $B = C$;
- the third step establishes $C \Leftarrow D$.

A conventional proof would use mostly natural language and would possibly start by justifying why A equals B, then it would show that B equals C, and that C follows from D. Then, by transitivity, it would conclude that $A \Leftarrow D$. But note how just articulating how a conventional argument would be, we have repeated the intermediate expressions B and C twice! Clearly, writing the conventional proof would lead to the same repetitions. In general, B and C can be long expressions, so we need a proof format that allows us to be concise and omit unnecessary intermediate expressions. We use Wim Feijen's proof format (described in detail in [Dij87], [DS90] and [Bac03, Chapter 3]), which for this small example would render:

$$
\begin{array}{ll}
 & A \\
= & \quad \{ \quad \text{hint why } A = B \quad \} \\
 & B \\
= & \quad \{ \quad \text{hint why } B = C \quad \} \\
 & C \\
\Leftarrow & \quad \{ \quad \text{hint why } C \Leftarrow D \quad \} \\
 & D \ .
\end{array}
$$

Although we repeat the expressions in the hints, we will not do it in general. As the examples below show, when justifying a step, we focus on the relevant properties. Note that this format forces the writer to provide explanation for each step, avoiding holes in the argument and making it easier to check. It also allows us to conclude immediately that $A \Leftarrow D$ without reading the intermediate expressions. The example shows that we can use different relations between the steps; in fact, we can use any transitive relation. Another advantage is that the use of a systematic proof format allows us to compare two different proofs of the same theorem more effectively.

[1] $A \Leftarrow D$ is read as "A if D" and is the same as $D \Rightarrow A$.

There is a good reason for writing the different expressions aligned in different lines: the main operation that we perform most frequently when manipulating formulae is substitution of an expression by another expression. Therefore, if all the expressions are aligned, it is much easier to do a syntactic comparison and verify what changed from one step to the other.

We are convinced that the calculational method, with its emphasis on manipulation and with its proof format that forces the justification of each step, can have a tremendous impact on school mathematics. Moreover, we believe that students would benefit from an earlier introduction and explicit use of formal logic. In particular, we think that equational logic, which is based on equality and Leibniz's rule of "substitution of equals for equals", is suitable and easy to teach. The next paragraphs show two examples of the calculational method in action.

Rewriting a proof on sets Consider the following property that holds for all sets A, B, and C:

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

The following proof, which establishes this property by mutual inclusion, is from a math textbook (but we extracted it from [GS95]) and illustrates how proofs about sets are conventionally done:

> We first show that $A \cup (B \cap C) \subseteq (A \cup B) \cap (A \cup C)$. If $x \in (A \cup (B \cap C))$, then either $x \in A$ or $x \in (B \cap C)$. If $x \in A$, then certainly $x \in (A \cup B)$ and $x \in (A \cup C)$, so $x \in ((A \cup B) \cap (A \cup C))$. On the other hand, if $x \in (B \cap C)$, then $x \in B$ and $x \in C$, so $x \in (A \cup B)$ and $x \in (A \cup C)$, so $x \in ((A \cup B) \cap (A \cup C))$.
>
> Hence, $A \cup (B \cap C) \subseteq (A \cup B) \cap (A \cup C)$.
>
> Conversely, if $y \in ((A \cup B) \cap (A \cup C))$, then $y \in (A \cup B)$ and $y \in (A \cup C)$. We consider two cases: $y \in A$ and $y \notin A$. If $y \in A$, then $y \in (A \cup (B \cap C))$, and this part is done. If $y \notin A$, then, since $y \in (A \cup B)$ we must have $y \in B$. Similarly, since $y \in (A \cup C)$ and $y \notin A$, we have $y \in C$. Thus, $y \in (B \cap C)$, and this implies $y \in (A \cup (B \cap C))$. Hence $((A \cup B) \cap (A \cup C) \subseteq A \cup (B \cap C))$. The theorem follows.

Note that this proof is not clear about the facts that it is using. For example, it says "If $y \notin A$, then, since $y \in (A \cup B)$ we must have $y \in B$", but there is no reference to the theorem that supports this claim. Moreover, the repetition of intermediate expressions and all the case analysis make the proof long and verbose. Let us see now how we would prove the property using a calculational approach:

Below, we prove that, for all v, $v{\in}(A{\cup}(B{\cap}C)) \equiv v{\in}((A{\cup}B){\cap}(A{\cup}C))$. By Extensionality (the definition of equality of sets), we can conclude

$$A{\cup}(B{\cap}C) = (A{\cup}B){\cap}(A{\cup}C) \quad .$$

Here is the proof:

$$\begin{array}{ll} & v{\in}(A{\cup}(B{\cap}C)) \\ = & \quad \{ \quad \text{definition of } \cup \quad \} \\ & v{\in}A \vee v{\in}(B{\cap}C) \\ = & \quad \{ \quad \text{definition of } \cap \quad \} \\ & v{\in}A \vee (v{\in}B \wedge v{\in}C) \\ = & \quad \{ \quad \text{distributivity of } \vee \text{ over } \wedge \quad \} \\ & ((v{\in}A \vee v{\in}B) \wedge (v{\in}A \vee v{\in}C)) \\ = & \quad \{ \quad \text{definition of } \cup \text{, twice} \quad \} \\ & v{\in}(A{\cup}B) \wedge v{\in}(A{\cup}C) \\ = & \quad \{ \quad \text{definition of } \cap \quad \} \\ & v{\in}((A{\cup}B){\cap}(A{\cup}C)) \quad . \end{array}$$

In contrast to the conventional proof, this proof is concise, is explicit about all the properties it uses, and eliminates all the case analysis. Moreover, it reveals that the core property of the theorem is that disjunction distributes over conjunction. In fact, this proof uses a strategy that is common in mathematics: to prove something about operators, eliminate them using their definitions (which are usually based on more elementary operators), manipulate the formulae, and reintroduce the original operators.

The calculational approach and equational logic can be extended to many mathematical domains. Nevertheless, we believe that they should be introduced using simple and recreational problems. In particular, logic puzzles where the goal is to solve simultaneous equations on Booleans, can be introduced by analogy with simultaneous equations on numbers. For example, consider the following problem:

> Suppose Ben is twice as old as Anne, but two years ago, Ben was three times as old as Anne. How old are Ben and Anne?

In our experience, most secondary-school students know how to solve this problem. Their first step is to model the problem as the two simultaneous equations

$$b = 2{\times}a \quad \wedge \quad b{-}2 = 3{\times}(a{-}2) \quad ,$$

where a and b denote, respectively, Ben and Anne's ages. Then, and now using the calculational proof format, most of them know how to calculate the correct solution:

$$
\begin{array}{ll}
 & b = 2\times a \ \ \wedge \ \ b-2 = 3\times(a-2) \\
= & \quad \{ \quad \text{replace } b \text{ by } 2\times a \quad \} \\
 & b = 2\times a \ \ \wedge \ \ 2\times a - 2 \ = \ 3\times(a-2) \\
= & \quad \{ \quad \text{arithmetic} \quad \} \\
 & b = 2\times a \ \ \wedge \ \ 4 = a \\
= & \quad \{ \quad \text{replace } a \text{ by } 4 \quad \} \\
 & b = 8 \ \wedge \ 4 = a \ .
\end{array}
$$

Now, consider a different problem:

> In an abridged version of Shakespeare's *Merchant of Venice*, Portia had two caskets: gold and silver. Inside one of these caskets, Portia had put her portrait, and on each was an inscription. Portia explained to her suitor that each inscription could be either true or false but, on the basis of the inscriptions, he was to choose the casket containing the portrait. If he succeeded, he could marry her.
>
> The inscriptions were:
>
> Gold: *The portrait is in this casket.*
> Silver: *If the inscription on the gold casket is true, this inscription is false.*
>
> Which casket contained the portrait? What can we deduce about the inscriptions?

Most students solve this problem by case analysis. However, this problem is similar to the one above. The main difference is the domain: whilst the problem above was about solving simultaneous equations on natural numbers, this problem is about solving simultaneous equations on Booleans. As a result, the strategy is the same. First, we model the problem by writing down the simultaneous equations

$$(pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge (is \ \equiv \ ig \Rightarrow \neg is) \quad ,$$

where the variables mean:

ig the inscription on the gold casket is true
is the inscription on the silver casket is true
pg the portrait is in the gold casket
ps the portrait is in the silver casket .

Then, instead of using the algebra of numbers, we use the algebra of Booleans:

$$
\begin{array}{ll}
 & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge (is \equiv ig \Rightarrow \neg is) \\
= & \quad \{ \quad \text{definition of } \Rightarrow \text{ and associativity} \quad \} \\
 & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge ((is \equiv \neg is) \equiv \neg is \vee ig) \\
= & \quad \{ \quad \text{negation (twice)} \quad \} \\
 & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge \neg(\neg is \vee ig) \\
= & \quad \{ \quad \text{De Morgan} \quad \} \\
 & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge is \wedge \neg ig \\
= & \quad \{ \quad \text{reflexivity, negation, and Leibniz} \quad \} \\
 & (pg \equiv \neg ps) \wedge (\mathsf{false} \equiv pg) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{false}) \\
= & \quad \{ \quad \text{Leibniz and negation} \quad \} \\
 & (\mathsf{true} \equiv ps) \wedge (\mathsf{false} \equiv pg) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{false}) \quad .
\end{array}
$$

This problem is discussed in scenario 4, so we will not discuss any details of the rules involved. We would just like to remark that the property we call "Leibniz" corresponds to "substitution of equals for equals" (we have used it twice in the problem on numbers). Also, observing that the variables pg and ig are equal, we could simplify the solution by naming only one of them. However, since this is a coincidence, we prefer to show the general method of solution. The exercise 4.6.4 in scenario 4, for example, shows a variation of the problem where such a coincidence does not happen.

We believe that logic puzzles such as this one are a good vehicle to teach manipulation without interpretation. In our experience, most people would agree that using case analysis in the problem on Ben and Anne's ages is not a good idea, because case analysis is very specific and does not scale well to more complicated problems; most people would agree that it is more important to teach the students how to solve general systems of simultaneous equations. Moreover, it is important to remark that when the students are solving such problems on numbers, they are not interpreting the formulae. We think the same should be done when solving problems on Booleans; for that

reason, we include three different logic puzzles in our catalogue of scenarios (scenarios 3, 4, and 5) and we show how to solve them calculationally.

Formal manipulation of algorithms We conclude this section by showing how the calculational method can be used to support formal manipulation of algorithms. Suppose that x and y are numbers and we are asked to determine what the following program does[2]:

$$x := x+y \; ; \; y := x-y \; ; \; x := x-y \; .$$

One way of solving the problem is to test the program with some specific values, guess what it is doing to the variables, and verifying if the guess is correct. However, this strategy is not easy to apply in general, since most programs are more complex. So, instead of guessing, let us *calculate* what the program does. In other words, let us calculate its functional specification.

The specification of a program can be seen as a relation between its input and its output. The specification of this program, for example, would relate the initial values of x and y to their final values after running the program. We specify a program S by stating a *precondition* P and a *postcondition* Q and requiring that S be constructed to satisfy

$$\{ P \} \; S \; \{ Q \} \quad .$$

If so, we say that S establishes the postcondition Q under the assumption of precondition P. In other words, the notation $\{ P \} \; S \; \{ Q \}$ means that, for all possible values of the variables in P, S and Q, if, initially, the state of the program variables satisfies the predicate P and the statement S is executed, S is guaranteed to terminate and, on termination of S, the final state will satisfy the predicate Q. We call the triple $\{ P \} \; S \; \{ Q \}$ a *Hoare triple*[3].

We can use this new notation to state more precisely our goal. The goal is to calculate a precondition P in:

$$\{ P \} \quad x := x+y \; ; \; y := x-y \; ; \; x := x-y \; \{ x = X \wedge y = Y \} \; .$$

The postcondition is the most general we can have: it states that x and y will have some values (X and Y) on termination. Now, consider the third assignment of the

[2]In this thesis, we use the Guarded Command Language (GCL) to express algorithms. GCL is a very simple programming language with just four programming constructs — assignment, sequential composition, conditionals, and loops. The GCL was introduced by Dijkstra [Dij75].

[3]Hoare triples were introduced by Sir Tony Hoare in [Hoa69]. The presentation we show here is, essentially, the same as the one in [Bac03, chapter 9], which we recommend.

program, $x := x-y$. After this assignment, the value of x will become $x-y$. So, if the postcondition is to apply to x after the assignment, it should apply to $x-y$ before the assignment. More specifically, $x-y = X$ should be true before the assignment. This rule is called the *assignment axiom* and in its general form can be written as

$$\{\ Q[x := e]\ \}\ x := e\ \{\ Q\ \}\quad ,$$

where $Q[x := e]$ corresponds to the property Q with all occurrences of 'x' replaced by 'e'. Using the assignment axiom, and working backwards from the postcondition, we can now annotate the program as follows. The reader should read it from bottom to top:

$$\{\ (x+y)-((x+y)-y) = X \wedge (x+y)-y = Y\ \}$$
$$x := x+y\ ;$$
$$\{\ x-(x-y) = X \wedge x-y = Y\ \}$$
$$y := x-y\ ;$$
$$\{\ x-y = X \wedge y = Y\ \}$$
$$x := x-y$$
$$\{\ x = X \wedge y = Y\ \}\ .$$

This means that the precondition we are looking for is

$$(x+y)-((x+y)-y) = X \wedge (x+y)-y = Y\quad .$$

Simplifying the arithmetic expressions, we have:

$$\{\ y = X \wedge x = Y\ \}\quad x := x+y\ ;\ y := x-y\ ;\ x := x-y\ \{\ x = X \wedge y = Y\ \}\ .$$

The conclusion is that, provided that the program starts in a state where $y = X$ and $x = Y$, its execution is guaranteed to terminate in a state where $x = X$ and $y = Y$. In other words, the program is swapping the values of the variables x and y. Note that the program does not use any additional variables to swap the values! We discuss this program and its generalisation in scenario 8. In fact, this program can be used with other operators and it is usually presented as a programming trick, with no formal justification (see, for example, [CR06, p. 130], and [PC06, p. 182]). We believe that this reflects the informal style of most software engineering practitioners.

As we have just seen, this calculational approach is useful for verifying the correctness of existing programs. However, the ideal is to avoid guessing and to use the method

to *calculate* programs that satisfy a given specification. In chapter 4, for example, we show how to calculate several algorithms from their formal specifications. In the next section, we briefly discuss the constructive aspect of the method.

2.3 Goal-oriented investigations

Conventional mathematics is characterised by the "theorem-proof" style of reasoning, in which theorems are first formulated and then verified. But as Roland Backhouse explains in [Bac02] , while conventional mathematics is primarily concerned with the modelling and *analysis* of existing natural systems, computing is dominated by a concern for *synthesis*, i.e., with the design and construction of new systems and the accompanying algorithmic techniques. Therefore, computing is characterised by an emphasis on construction and by a goal-oriented style of reasoning. The final example of the previous section is an example of a goal-oriented investigation. Rather than guessing a precondition and then verifying it, we calculate what the precondition should be.

In fact, we can use the same skills to calculate programs. For example, suppose that we are asked to write an assignment X to the variable s such that[4]

$$\{\ s = n^2\ \}\ s\ ,\ n\ :=\ X\ ,\ n{+}1\ \ \{\ s = n^2\ \}\ .$$

We can easily guess that $s\ :=\ (n{+}1)^2$ is suitable. But suppose that we are not allowed to use exponentiation — the computation of s should only involve additions and not multiplications. Which assignment should we write? Well, applying the assignment axiom, we get

$$\{\ X = (n{+}1)^2\ \}\ s\ ,\ n\ :=\ X\ ,\ n{+}1\ \ \{\ s = n^2\ \}\ .$$

Comparing with the specification, the goal is to calculate X so that

$$s = n^2\ \ \Rightarrow\ \ X = (n{+}1)^2\ \ .$$

We assume $s = n^2$ and calculate:

$$\begin{array}{ll} & X = (n{+}1)^2 \\ = & \{\quad \text{arithmetic}\quad \} \end{array}$$

[4]The assignment $s\ ,\ n\ :=\ X\ ,\ n{+}1$ is a simultaneous assignment (also called multiple assignment). In general, a simultaneous assignment $x_0\ ,\ x_1\ ,\ \cdots\ ,\ x_n\ :=\ e_0\ ,\ e_1\ ,\ \cdots\ ,\ e_n$ is executed by evaluating all the expressions $e_0\ ,\ e_1\ ,\ \cdots\ ,\ e_n$ and then, for each i, updating the value of the variable x_i to the value obtained for expression e_i.

$$
\begin{array}{ll}
 & X \;=\; n^2 + 2{\cdot}n + 1 \\
= & \quad \{ \quad s = n^2 \text{ and } 2{\cdot}n = n{+}n \text{ (recall that we cannot use multiplication)} \quad \} \\
 & X = s{+}n{+}n{+}1 \;\; .
\end{array}
$$

We conclude, with no guessing involved, that the required assignment statement is:

$$\{\, s = n^2 \,\}\; s\,,n \;:=\; s{+}n{+}n{+}1\,,n{+}1 \quad \{\, s = n^2 \,\} \;\; .$$

We believe that the emphasis on construction brings reliability and can have a tremendous impact on mathematics. Mathematicians, of course, are aware of the benefits. In [Pol81, chapter 8] and [Pol90, pp. 141–148 (Pappus) and pp. 225–232 (Working backwards)] , for example, Pólya discusses goal-oriented reasoning. He calls it *regressive planning*, or *working backwards*, and he classifies it as a "basically important pattern". Perhaps one reason why the pattern is not so used in conventional mathematics is that formulae are usually interpreted. In contrast, in the calculational method, formulae are manipulated, most of the time, without interpretation. This allows us to use guiding principles and heuristics based on the shape of the formulae. It is common, for example, to have an almost-forced obvious sequence of steps, once we formally express the goal. For example, in scenario 6, we discuss the following problem:

> Let a finite number of points be joined in pairs by any system of curves, including the possibility of loops (for example, joining a point C with itself; see figure 2.1) and of multiple edges (joining the same pair of points). We define the *local degree* of a vertex A, denoted by $d.A$, to be the number of edges incident with the point A, counting loops twice. For example, in figure 2.1,
>
> $$d.A = 6,\; d.B = 3, \text{ and } d.C = 3 \;\; .$$
>
> We want to show that in any network, as outlined above, the number of vertices which have odd local degree is an even number. (Note that in the system shown in figure 2.1, precisely *two* vertices, B and C, have odd local degrees.)
>
> **The Handshaking Lemma** This property is also known as the Handshaking Lemma. As explained in [Hon98, p. 8], if we think of the vertices as people, and the joining of two vertices A and B (say) to mean that A and B shook hands (loops, if any, indicating one shook hands with himself and

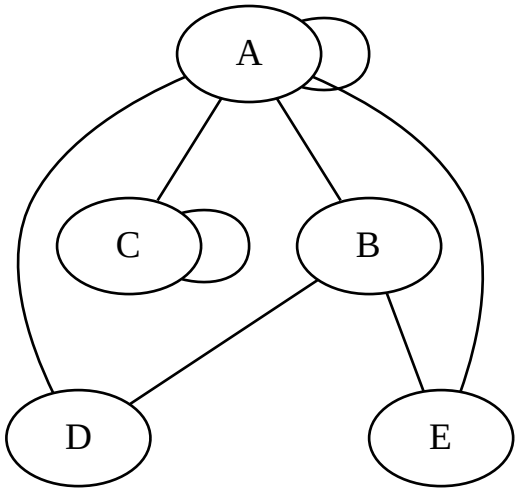


Figure 2.1: System of curves with five points

> counting as two handshakes), the local degree $d.A$ of a vertex A gives the total number of times A shook hands. What we want to show, then, is that the number of people who have shaken hands an odd number of times is even. This application is all the more interesting because it is independent of time — one can state without fear of contradiction that the number of people at the opera next Thursday (or in the whole world from the beginning of time if you like) who will shake hands an odd number of times is even. (One might enjoy verifying this result with a group of friends.)

The goal of the problem is to determine the value of the following expression[5]:

$$even.\langle \Sigma a \ : \ a{\in}V \wedge odd.(d.a) \ : \ 1 \rangle \quad .$$

If the result is true, there is an even number of vertices with odd degree; otherwise, there is an odd number. The problem statement claims that the result is always true.

[5]Quantifiers allow us to denote the operation of applying some binary operator (like addition, multiplication, conjunction, disjunction or equivalence) to an arbitrary bag of values. We use a uniform notation for quantifiers: the Eindhoven quantifier notation (mentioned in section 2.2). There are five components to the notation. The first component is the *quantifier*. In this case, the quantifier is Σ, which denotes summation of an arbitrary number of values. We also use $\equiv$ as a quantifier to denote a continued equivalence of an arbitrary number of values. The second component is the *dummy* variable; in this case, variable a. The third component is the *range* of the dummy; in this case, the range is $a{\in}V \wedge odd.(d.a)$. The range is a Boolean-valued expression that determines the set of values of the dummy for which the expression is true. The fourth component is the *term*. In this case, the term is the natural number 1, meaning that we add 1 for each value a that satisfies the range (in other words, we are adding 1 (counting) for each node a with an odd degree in V). The final component of the notation is the angle brackets; these serve to delimit the scope of the dummy variable. Also, we use an infix dot to denote functional application.

However, the goal we propose is to calculate its value. We know that predicate *even* distributes through addition, that is:

$$even.(m+n) \equiv even.m \equiv even.n \quad .$$

In terms of arbitrary summations, this rule can be expressed as:

$$even.\langle \Sigma a{:}R{:}T \rangle = \langle \equiv a{:}\ R{:}\ even.T \rangle \quad .$$

This means that we can use this property to manipulate our goal:

$$\begin{array}{ll} & even.\langle \Sigma a\ :\ a{\in}V \wedge odd.(d.a)\ :\ 1 \rangle \\ = & \quad \{\quad even \text{ distributes over addition} \quad \} \\ & \langle \equiv a{:}\ a{\in}V \wedge odd.(d.a){:}\ even.1 \rangle \\ = & \quad \{\quad even.1 \text{ is } \mathsf{false} \quad \} \\ & \langle \equiv a{:}\ a{\in}V \wedge odd.(d.a){:}\ \mathsf{false} \rangle \\ = & \quad \{\quad \text{the range can be simplified by using the so-called} \\ & \qquad \text{trading rule (the conjunct } odd.(d.a)\wedge \text{ is ``traded''} \\ & \qquad \text{into an implication } odd.(d.a)\Rightarrow \text{ in the term)} \quad \} \\ & \langle \equiv a{:}\ a{\in}V{:}\ odd.(d.a) \Rightarrow \mathsf{false} \rangle \\ = & \quad \{\quad odd.(d.a) \Rightarrow \mathsf{false} \ \equiv\ even.(d.a) \quad \} \\ & \langle \equiv a{:}\ a{\in}V{:}\ even.(d.a) \rangle \\ = & \quad \{\quad even \text{ distributes over addition} \quad \} \\ & even.\langle \Sigma a : a{\in}V : d.a \rangle \quad . \end{array}$$

This calculation shows that the parity of the number of vertices with odd degree is the same as the parity of the sum of all the degrees. But because each edge has two ends, the sum of all the degrees is simply twice the total number of edges. We thus have:

$$\begin{array}{ll} & even.\langle \Sigma a\ :\ a{\in}V \wedge odd.(d.a)\ :\ 1 \rangle \\ = & \quad \{\quad \text{calculation above} \quad \} \\ & even.\langle \Sigma a : a{\in}V : d.a \rangle \\ = & \quad \{\quad \text{the sum of all the degrees is twice the number of edges;} \\ & \qquad \text{hence, it is an even number} \quad \} \\ & \mathsf{true}\ . \end{array}$$

And so we can conclude that every undirected graph contains an even number of vertices with odd degree.

The fact that *even* distributes over addition allowed to transform the problem into a problem in logic. Once we have done that, we were driven by the goal of simplifying the formulae. This calculation is clearly a process guided by the shape of the formulae. On the other hand, conventional solutions for this problem are not goal-oriented. Consider the following solution, representative of the conventional style, taken from [Hon98, p. 8]:

> The proof in general is simple. We denote by T the total of all the local degrees:
>
> (1) $T = d(A) + d(B) + d(C) + \cdots + d(K)$.
>
> In evaluating T we count the number of edges running into A, the number into B, etc., and add. Because each edge has two ends, T is simply twice the number of edges; hence T is even.
>
> Now the values $d(P)$ on the right-hand side of (1) which are even add up to a sub-total which is also even. The remaining values $d(P)$ each of which is odd, must also add up to an even sub-total (since T is even). This shows that there is an even number of odd $d(P)$'s (it takes an even number of odd numbers to give an even sum). Thus there must be an even number of vertices with odd local degree.

There is nothing wrong with this solution in the sense that it clearly shows why the property holds. However, it is clearly oriented to verification: it starts by introducing the total sum of all the local degrees, observing that its value is even; then it analyses that sum to conclude the property. The question is: how can we teach students to consider the total sum of all the local degrees? In general, how can we teach students to identify seemingly unrelated concepts that will be crucial in the development of their arguments? We do not think we can.

On the other hand, if we look at the goal-oriented proof, we see that the goal is simple to express. Furthermore, with some training, most students would write it correctly and would be able to calculate that the parity of the number of vertices with odd degree is the same as the parity of the sum of all the degrees. And then (and only then) the introduction of the total sum of all the degrees would make sense. In conclusion, we believe it is more valuable to work in a formal and goal-oriented way, since it allows us to discover the crucial properties.

2.4 On concision and avoidance of unnecessary detail

Effective reasoning depends on economy of expression. This is particularly true when reasoning about programs: since formulae can be long, including unnecessary detail can lead to unmanageable complexity.

One way of achieving concision is by avoiding unnecessary case analysis. In general, solutions by case analysis result from failing in the identification of fundamental, structural properties. We have seen an example in page 21, on how to rewrite a proof by mutual inclusion into a proof that was more concise, mainly because it avoided an unnecessary case analysis. In that case, the structural property is that disjunction distributes over conjunction.

Exploiting structural properties leads to shorter proofs. However, it is important to note that concision is not about the number of words or lines of a proof. As an example, consider the following lemma taken from the book [JJ98, p. 39][6]:

> **Lemma 3.1**
> For any fixed $n \geq 1$ we have $a \cong b \pmod{n}$ if and only if $n \backslash (a-b)$.

For the authors, $a \cong b \pmod{n}$ is defined as a and b having the same remainder upon division by n. Their proof is by mutual implication:

> **Proof**
> Putting $a = q \times n + r$ and $b = q' \times n + r'$ as above, we have $a-b = (q-q') \times n + (r-r')$ with $-n < r-r' < n$. If $a \cong b \pmod{n}$ then $r = r'$, so $r-r' = 0$ and $a-b = (q-q') \times n$, which is divisible by n. Conversely, if n divides $a-b$ then it divides $(a-b) - (q-q') \times n = r-r'$; now the only integer strictly between $-n$ and n which is divisible by n is 0, so $r-r' = 0$, giving $r = r'$ and hence $a \cong b \pmod{n}$.

By most conventional standards, this is a concise proof. However, we can avoid the mutual implication and prove the lemma as follows:

> Writing $a = q \times n + r$ and $b = q' \times n + r'$, we have $a-b = (q-q') \times n + (r-r')$, with $-n < r-r' < n$. Hence,

[6] The notations we use are not the same as in the book, but the lemma and the structure are (including the text). The symbol $\cong$ is used for the congruence relation and $n \backslash (a-b)$ means that n divides $a-b$.

$$
\begin{array}{ll}
 & n \backslash (a-b) \\
= & \quad \{ \quad \text{consideration above} \quad \} \\
 & n \backslash ((q-q') \times n + (r-r')) \\
= & \quad \{ \quad \text{division property} \quad \} \\
 & n \backslash (r-r') \\
= & \quad \{ \quad -n < r-r' < n \text{ and the only integer strictly between } -n \text{ and } n \\
 & \qquad \text{which is divisible by } n \text{ is } 0 \quad \} \\
 & r-r' = 0 \ .
\end{array}
$$

Therefore $r = r'$ and, by definition, $a \cong b \pmod{n}$.

The size of the two proofs is similar (the second uses less words, but has more lines). Nevertheless, we consider the second proof more concise; it avoids the two cases associated with the mutual implication, by identifying the two relevant properties (the properties used in the second and third steps). Note how the second part of the first proof mentions all the relevant properties for establishing the equality; the authors could establish the theorem just by replacing

"if n divides $a-b$ then it divides $(a-b) - (q-q') \times n = r-r'$"

by

"n divides $a-b$ *is the same as* n divides $r-r'$" .

This example also shows that an emphasis on equational logic can lead to better arguments.

On naming It is impossible to solve problems without introducing names. However, if we name unnecessary elements or if we make unnecessary distinctions, we add unnecessary detail and complexity to the solution. The two following examples show how the avoidance of unnecessary naming leads to more effective and simple solutions. Both examples are taken from [Bac07]. Let us start with the problem "Goat, Cabbage and Wolf", shown in page 14. The problem is about crossing four individuals without violating the two conditions:

1. the goat should not be left alone with the cabbage;
2. the wolf should not be left alone with the goat.

These two conditions expose a similarity between the wolf and the cabbage: the goat cannot be left with either the wolf or the cabbage. Moreover, there are no restrictions on leaving the wolf alone with the cabbage. This clearly suggests that both the cabbage and the wolf are playing the same role. Why, then, are the "wolf" and the "cabbage" distinguished by giving them different names?

Let us restate the problem[7], 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 (note that in the problem with the four individuals, we would have two solutions, since we have two different choices when choosing the first beta to take across).

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.

Another problem where the decision of naming is even more important is the following:

> **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?

[7] The restatement of the problem and the subsequent two paragraphs are extracted from [Bac07], since the author of this dissertation could not write it any better.

One way of naming the elements of this problem is by distinguishing the individual people, as in, for example, H_1, W_1, H_2, W_2, H_3, and W_3 — where the Hs are husbands, the Ws are wives, and a pair H_n,W_n forms a couple.

Using this notation, one possible transition from the initial state is to take across the couple H_1,W_1. Similarly, we can take across the couple H_2,W_2, or the couple H_3,W_3. This suggests that we could find a notation to represent a couple and, instead of the three different transitions, we would have one single transition. That is indeed what we do. As in [Bac07], we use the letters H, W and C to mean husband, wife and couple, respectively. These are preceded by a number; for example, $2H$ means two husbands, $3C$ means three couples and $1C$,$2H$ means one couple and two husbands. We exploit the notation to distinguish between couples and individuals; for example, $1H$,$1W$ means a husband and wife who do not form a couple, whilst $1C$ means a husband and wife who do form a couple.

The second notation is avoiding unnecessary detail, because there is no need to distinguish the individuals. Note that if we use the first notation, we have a total of 60 possible transitions; on the other hand, if we use the second notation, we reduce this number to a third, i.e., we have a total of 17 possible transitions! It is a massive improvement for such a small problem. We discuss this problem further in the next chapter.

Finally, in chapter 4, we show the gains in our problem-solving skills that can be achieved by the right combination of precision and concision. In particular, the crucial step that allowed the discovery of the two main novel contributions was to rewrite the so-called extended Euclid's algorithm in terms of matrices. The conventional formulation of the algorithm uses four distinct variables; our reformulation encapsulates these variables into one 2×2 matrix. Not only we avoid unnecessary naming, but we also gain from the introduction of matrix arithmetic.

CHAPTER 3

Techniques for Algorithmic Problem Solving

The required techniques of effective reasoning are pretty formal, but as long as programming is done by people that don't master them, the software crisis will remain with us and will be considered an incurable disease. And you know what incurable diseases do: they invite the quacks and charlatans in, who in this case take the form of Software Engineering gurus.

— EDSGER W. DIJKSTRA (2000)

This chapter discusses techniques that can be used to simplify and solve problems. In contrast to the previous chapter, which contains general principles that apply to all problems, the techniques shown in this chapter are more specific. For example, there are techniques that are irrelevant to certain problems, and there are problems that have different solutions, each based on a different technique. Moreover, certain techniques can be combined together to achieve more effective solutions.

As in the previous chapter, we illustrate each technique with simple and accessible examples, and, whenever convenient, we refer to the parts of the thesis where the technique is used.

We start, in section 3.1, by discussing one of the most elementary techniques in problem solving: problem decomposition. In particular, we focus on the role of sequential composition. We complement this section with section 3.2, where we show how a combination of symmetry and problem decomposition can massively simplify some solutions. In section 3.2, we also include a discussion on how algebraic symmetry can be used to guide and simplify calculations. Still connected with problem decomposition is the notion of distributivity, which we discuss in section 3.3. Besides its use to simplify calculations, distributivity can also be used to name the elements of a problem

more effectively.

Perhaps the most important technique in algorithmic problem solving is the use of invariants, which we discuss in section 3.4. We show how we can calculate invariants, and how we can use invariants to calculate algorithms. More specifically, we show the derivation of a non-trivial algorithm that is guided by the notion of invariance.

Essential to the the derivation of algorithms is the ability to prove that an algorithm terminates. In section 3.5, we show how termination is typically proved.

Finally, we conclude the chapter in section 3.6 with a discussion on program inversion. Although a formal approach to program inversion can be used to prove certain arguments more precisely, the technique is not widely used. We believe it deserves to be better known.

3.1 Problem decomposition

Problem decomposition is one of the most elementary problem-solving techniques. It consists in breaking down a problem into smaller and more manageable problems, and finding a way of combining the solutions of the smaller problems to solve the original one. This technique is sometimes called *divide-and-conquer*, but the latter denomination is more used when the problem is broken down recursively.

The simplest way of combining two smaller problems into a larger problem is by sequential composition. For example, constructing a program S satisfying the specification

$$\{\ P\ \}\ S\ \{\ Q\ \}$$

can be done by constructing two (smaller) programs, S_1 and S_2 such that

$$\{\ P\ \}\ S_1\ \{\ R\ \}$$

and

$$\{\ R\ \}\ S_2\ \{\ Q\ \}\quad .$$

Provided that P is guaranteed, program S_1 establishes R; therefore, executing S_2 after S_1 establishes Q. We write $S_1;S_2$ to denote execution of S_1 followed by execution of S_2, and we have

$$\{\ P\ \}\ S_1\,;S_2\ \{\ Q\ \}\quad .$$

Note that if one of the smaller problems is easy to solve, S is essentially reduced to the more difficult problem. This suggests that whenever we have to solve an algorithmic problem, we should always think of states from which the problem is easy to solve. For example, recall the problem of "The Chameleons of Camelot", briefly discussed in chapter 2:

> **The Chameleons of Camelot**
>
> On the island of Camelot there are three different types of chameleons: grey chameleons, brown chameleons, and crimson chameleons. Whenever two chameleons of different colours meet, they both change colour to the third colour.
>
> For which number of grey, brown, and crimson chameleons is it possible to arrange a succession of meetings that results in all the chameleons displaying the same colour?

Assuming that S is the algorithm that organises meetings, the goal of the problem is to find a precondition P such that:

$\{\ P\ \}$

S

$\{\ \textit{All the chameleons display the same colour}\ \}$.

One way of decomposing the problem is to think of states for which the problem is easy to solve. In other words, for which numbers of chameleons is it easy to arrange meetings that satisfy the goal? For example, the simplest states we can think of are the ones where there is only one type of chameleon; in this case the goal is trivially satisfied. Similarly, the problem is easy to solve when the number of chameleons of the three different colours is the same; in this case, we can choose two colours and organise a meeting between all the chameleons of these two colours. More generally, for the states where at least two types of chameleons are equally numbered, we can arrange a meeting between all the chameleons of these two types. As a result, we can decompose the problem as follows:

$\{\ P\ \}$

S_1

; $\{\ \textit{Two classes of chameleons are equally numbered}\ \}$

Two classes of chameleons are equally numbered, so we can arrange a meeting

between all the chameleons of these two classes. As a consequence, their colour changes to the third one.

(If there are only chameleons of one colour, there is nothing left to do.)

{ *All the chameleons display the same colour* } .

We solve this problem in scenario 9. This decomposition is key to the solution, since it helps us to find an important property of the algorithm S_1 that determines P.

An alternative way of decomposing a problem consists in identifying states that are easy to achieve from the initial state. In the next section we will see an example.

The next section contains more examples where problem decomposition is used. More specifically, we show how a combination of problem decomposition with symmetry can massively improve and simplify our solutions.

3.2 Symmetry

Symmetry is a regularity that is possessed by an object and is characterised by the transformations that leave the object unchanged. For example, the sequence

4,3,2,1,2,3,4

is symmetric with respect to the action "swap every k^{th} element to the right of 1 with the k^{th} element to the left of 1". The importance of symmetry is that it gives us *free information* about the structure of a problem. For example, given the action above and the partial sequence 1,2,3,4, we would be able to reconstruct the entire sequence very easily. The relevance to algorithms is clear: if we know that the solution to an algorithmic problem is symmetric with respect to some action, we just need to construct half of the solution; the rest follows from symmetry.

The two following problems are examples of symmetric problems. The first is the problem of the Jealous Couples that we have already seen in page 34. Indeed, any river-crossing problem where the crossing rules are the same for both directions is symmetric. If we reverse a solution that crosses objects from left to right, we get another solution that crosses the same objects from right to left.

The second problem is known as the "nuclear pennies game". It is about moving a checker on a one-dimensional board according to some given rules. The solution we show is taken from [BCF10].

We conclude the section with a discussion on algebraic symmetry.

Jealous Couples For the reader's convenience, we repeat the problem statement (shown in section 2.4, page 34):

> **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?

Recall that we use the letters H, W and C to mean husband, wife and couple, respectively. For example, $1H$,$1W$ means a husband and wife who do not form a couple, whilst $1C$ means a husband and wife who do form a couple.

The solution we show is taken from [Bac07]. The solution to the problem is a sequence of transitions that leads from the initial state (where we assume that the three couples are at the left bank) to the final state (where the three couples are at the right bank).

We denote a state by two sequences separated by bars (the bars represent the river). An example is $1C$,$2H$ $||$ $2W$, 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 $||$ $3C$. This notation allows invalid states 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). More generally, any state where a W appears on the same side as a C or an H, is invalid.

To denote transitions, we use a similar notation: we use the space between the two bars to express who is crossing. An example is $3H$ $|2W|$ $1W$, which denotes the transition of transporting two wives across the river, leaving three husbands at the left bank and one wife at the right bank. An example of an invalid transition is $3H$ $|3W|$, because the boat can only carry at most two people.

We do not need to express the position or direction of the boat, since the boat and the three couples are initially at the same bank, and the boat must alternate between the left bank and the right bank. Note that the alternation of the boat means that the total number of transitions is odd.

Using Hoare triples to denote changes of state, the goal is to construct a sequence of transitions S satisfying

$$\{\ 3C\ ||\ \}\quad S\quad \{\ ||\ 3C\ \}\ .$$

In words, this means that provided that we have three couples at the left bank, performing the sequence of transitions S is guaranteed to terminate in a state where the three couples are at the right bank. A concrete example of this notation in use is:

$$\{\ 3C \mid\mid\ \}$$
$$1C,2H\ |2W|$$
$$\{\ 1C,2H \mid\mid 2W\ \}\ .$$

This example means that beginning in the initial state, letting two wives cross will result in a state where two husbands and one couple are at the left bank and two wives at the right bank.

Using this notation we can easily express our strategy for decomposing the problem. Our strategy can be summarised as exploiting two properties of the problem:

- The solution is symmetric, i.e., given a sequence of transitions that crosses the three couples from left to right, its reverse is a sequence of transitions that crosses the three couples from right to left;
- It is easy to get the wives from one side to the other whilst their husbands remain on one bank[1].

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

$$\{\ 3C \mid\mid\ \} \quad S_1 \quad \{\ 3H \mid\mid 3W\ \}\ ,$$
$$\{\ 3H \mid\mid 3W\ \} \quad S_2 \quad \{\ 3W \mid\mid 3H\ \}\ ,$$
$$\{\ 3W \mid\mid 3H\ \} \quad S_3 \quad \{\ \mid\mid 3C\ \}\ .$$

The sequence S_1 changes the state from the initial 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 final state, where everyone is at the right bank. So, doing S_1 followed by S_2 followed by S_3, which we

[1]The initial decomposition of this problem is an example where we identify states that are easily accessible from the initial state. Compare this with the problem "The Chameleons of Camelot", where we identify states from where we can easily achieve the final state.

denote by S_1 ; S_2 ; S_3, will achieve the goal of changing the state from the initial state (everyone is at the left bank) to the final state (everyone is at the right bank).

We have decomposed the problem into three different problems because we want to exploit symmetry, and because the total number of transitions is odd. Symmetry is captured by making the sequence S_3 the reverse of the sequence S_1. So, if we construct S_1, the sequence S_3 comes for free (this is what we mean by obtaining *free information*). As a result, we only have to tackle the problem of constructing S_1 and S_2.

The sequence S_1 is easy to construct, since it is easy to get the wives from one side to the other whilst their husbands remain on one bank. Here is how it is achieved:

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

We thus have,

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

From the symmetry of the solution, we immediately conclude that the sequence S_3 is the reverse of S_1:

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

(Note how S_3 is obtained by reading S_1 backwards.)

We now construct S_2 by further decomposing the problem. We first observe that S_1 leaves the boat at the right bank, so the starting position of S_2 is the right bank. Symmetrically, S_3 starts from the left bank, so S_2 must leave the boat at the left bank. Moreover, if the solution is to remain symmetric, and because the number of total transitions is odd, the middle transition has to be symmetric. Clearly, the only symmetric transition that is valid is $1C \mid 1C \mid 1C$[2]. Thus, S_2 must surely take the following form:

[2]Note how the notation adopted allows us to easily identify $1C \mid 1C \mid 1C$ as the middle transition. Had we chosen to denote only who crosses (e.g. "1C left" or "1C right"), it would be impossible to identify the middle transition.

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

The task is now to construct the symmetric sequences of transitions T_1 and T_2. Note that we do not know yet if the middle transition is from right to left or from left to right. If it is from right to left, the transition must be preceded by the state $1C \,||\, 2C$ and results in the state $2C \,||\, 1C$. Vice-versa, if the middle transition is from left to right, the transition must be preceded by the state $2C \,||\, 1C$ and results in the state $1C \,||\, 2C$. Using brute-force, we find that T_1 consists of just two actions:

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

Symmetrically, for T_2 we have:

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

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

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

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

The final solution involves eleven crossings, but we have only constructed five. By combining problem decomposition and symmetry, six crossings are given for free! We believe that this is an excellent example of how the free information that results from exploiting symmetry can massively improve our solutions. Moreover, the combination of problem decomposition and symmetry can be used to solve many other symmetric problems. The next paragraphs show another example: the "nuclear pennies game".

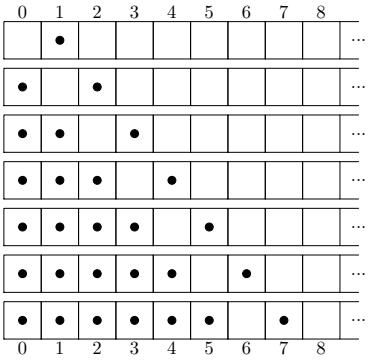
Seven-Trees-in-One and the Nuclear Pennies Game Consider the definition of binary trees — a binary tree is an empty tree or an element together with a pair of binary trees. Let us use symbols $+$ and $\times$ to denote disjoint union and Cartesian product respectively and let $\mathbb{1}$ denote the unit type. The type T of binary trees can be characterised by the type isomorphism $T \cong \mathbb{1}+T\times T$. Surprisingly, it can be shown that there is an isomorphism between seven-tuples of binary trees and binary trees. That is, $T^7 \cong T$. This has been dubbed "seven trees in one" by Blass [Bla95] who attributes the isomorphism to a remark made by Lawvere [Law91].

The isomorphism has been turned into a game with checkers called the "nuclear pennies game" [Pip07]. The game is played on a one-dimensional board of infinite extent. A checker is placed on one of the squares and the goal is to move the checker six places to the right. An atomic move is to replace a checker in a square numbered $n+1$ by two checkers, one in each of the two adjacent squares n and $n+2$, or vice-versa, two checkers, one in square n and one in square $n+2$ for some n, are replaced by a checker in square $n+1$. Note that there can be multiple checkers in a square. The connection with seven-trees-in-one is easy to see if one views a move as replacing $T^n\times T$ by $T^n\times(\mathbb{1}+T\times T)$ or vice-versa (having a checker in square n corresponds to T^n).

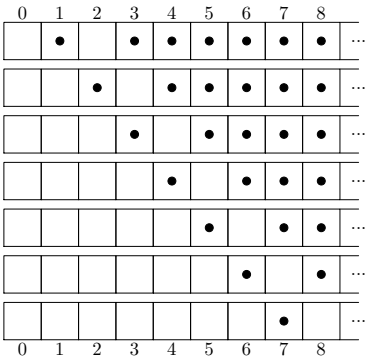
The nuclear-pennies game has an easy solution if one exploits the left-right symmetry of the problem (moving a checker 6 places to the right is symmetric to moving a checker 6 places to the left). The problem is decomposed into first ensuring that there is a

checker in the square 6 places to the right of the starting position and, symmetrically, there is a checker in the square 6 places to the left of the finishing position.

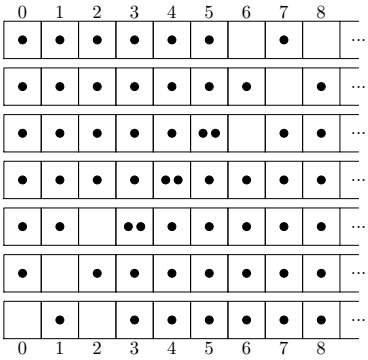
Achieving this first stage is easy. Below we show how it is done. First, six moves are needed to ensure that a checker is added six places to the right of the starting position. (This is shown below using dots to indicate checkers on a square. A blank indicates no checker on the square.)



Symmetrically, working from bottom to top, six moves are needed to ensure that a checker is added six places to the left of the finishing position.



Now the goal is to connect these two intermediate states (the bottom state in the top diagram and the top state in the bottom diagram). An appropriate (symmetrical) sequence of states is as follows. (For the reader's convenience, the last and first states in the above figures are repeated as the top and bottom states in the figure below.)



The first and last moves make the number of checkers in the leftmost and rightmost positions equal. Then a small amount of creativity is needed to identify the two (symmetrical) moves to the (symmetrical) middle state.

Algebraic symmetry We use the term *algebraic symmetry* when symmetry is used in an algebraic context; for example, when an expression consists of subexpressions with a common shape, we say that the expression is (algebraically) symmetric. That is, we can say that the expression

$$p \times q = p \times (x + y)$$

is symmetric, because the shapes of the expressions on both sides of the equality are similar (we have an expression pre-multiplied by p on both sides of the equality). Algebraic symmetry admits a notion of order; we can say, for example, that the expression above is more symmetric than its equivalent

$$p \times q \;=\; p \times x + p \times y \quad ,$$

since both sides of the equality in the first expression are more similar. In other words, in the first expression, both sides of the equality are multiplications that share the first argument (i.e., they share the same shape), whilst in the second expression, the shapes are different: the left side is a multiplication and the right side is an addition.

Sometimes, making an expression *more symmetric* can simplify and guide our calculations. Consider, for example, the problem of scenario 1, where we are ask to prove that the product of four consecutive positive natural numbers cannot be the square of an integer number.

Assuming that n is a positive natural number, the goal of the problem is to construct a calculation of the form

$$
\begin{array}{ll}
 & S.(n(n{+}1)(n{+}2)(n{+}3)) \\
= & \quad \{ \quad \text{justification} \quad \} \\
 & \mathsf{false},
\end{array}
$$

where $S.n$ is defined to be true only when n is the square of an integer.

Having nothing else to play with, let us manipulate the product $n(n{+}1)(n{+}2)(n{+}3)$ by using the property that multiplication distributes over addition:

$$
\begin{array}{ll}
 & S.(n(n{+}1)(n{+}2)(n{+}3)) \\
= & \quad \{ \quad \text{we use distributivity, twice; there are three ways in which} \\
 & \qquad \text{we can develop the product, so we choose to multiply} \\
 & \qquad n \text{ by } n{+}3 \text{ and } n{+}1 \text{ by } n{+}2 \text{ in order to introduce symmetry} \\
 & \qquad \text{—both have the term } n^2{+}3n \quad \} \\
 & S.((n^2{+}3n)(n^2{+}3n+2)) \\
= & \quad \{ \quad \text{we introduce symmetry again; we want to transform the} \\
 & \qquad \text{argument of } S \text{ into an expression that looks like a square} \quad \} \\
 & S.(((n^2{+}3n+1){-}1)((n^2{+}3n+1){+}1)) \\
= & \quad \{ \quad \text{difference of two squares, i.e., } (m{-}1)(m{+}1) = m^2{-}1 \quad \} \\
 & S.((n^2{+}3n+1)^2{-}1) \\
= & \quad \{ \quad \text{there are no two consecutive positive integers} \\
 & \qquad \text{that are both squares} \quad \} \\
 & \mathsf{false} \ .
\end{array}
$$

Note how symmetry guided our calculation! In fact, this problem is *asking* for symmetry, since the goal is to determine if we can express the product $n(n{+}1)(n{+}2)(n{+}3)$ as a product of two equal numbers, i.e., as a product with a symmetric shape.

This thesis contains other examples of where algebraic symmetry helps. In chapter 4, for example, we show how the transformation of the definition of greatest common divisor into a more symmetric definition suggests the structure of Euclid's algorithm.

3.3 Distributivity

Suppose that f is a function and $\oplus$ is a binary operator. We say that f distributes over $\oplus$, if for all a and b, there exists an operator $\otimes$, such that:

(3.3.1) $f.(a \oplus b) = f.a \otimes f.b$.

Distributivity is important for two reasons. First, it can be used to reduce the number of calculations. If we use (3.3.1) from right to left, we clearly reduce the number of function applications. For example, the fact that multiplication distributes over addition

$$a \times (b+c) = a \times b + a \times c$$

can be used to reduce the number of multiplications (an example is the last calculation of the previous section).

The second reason why distributivity is important is that, reading (3.3.1) from left to right, it corresponds to problem decomposition. If we want to apply a function to a "problem" $a \oplus b$, consisting of two smaller "problems" a and b, we apply the function to each "smaller problem" and we combine the solutions using the operator $\otimes$. A simple example of this pattern is the rule that most of us learn in school to calculate the value of products. For example, suppose that we want to compute the value of 13×102. We first observe that 102 is the same as $100+2$; so, using distributivity, we first calculate the values of 13×100 and 13×2 (which are easy to compute), and then we combine the results using addition (and we get the final result of 1326).

Sometimes, the decomposition of the problem may occur in the opposite direction. That is, the problem $a \oplus b$ can be a simplification of the problems a and b. In chapter 4, we show an example of when this occurs: we prove that the so-called Mersenne function distributes over the greatest common divisor:

$$2^{m \text{ gcd } n} - 1 = (2^m - 1) \text{ gcd } (2^n - 1) \quad .$$

In general, it is more difficult to compute the value of the right-hand side than to compute the value of the left-hand side. As an example, this property can be used to simplify the computation of the greatest common divisor of the numbers 1023 and 127:

$$\begin{array}{ll} & 1023 \text{ gcd } 127 \\ = & \{ \quad 1023 = 2^{10} - 1 \text{ and } 127 = 2^7 - 1 \quad \} \end{array}$$

$$
\begin{array}{ll}
 & (2^{10}-1) \text{ gcd } (2^7-1) \\
= & \quad \{ \quad \text{distributivity} \quad \} \\
 & 2^{10 \text{ gcd } 7}-1 \\
= & \quad \{ \quad 10 \text{ gcd } 7 = 1 \quad \} \\
 & 1 \ .
\end{array}
$$

Note that definition (3.3.1) is not conventional, since conventional definitions of distributivity involve only one function and one binary operator. In particular, where we write $\otimes$, most people would write the same operator $\oplus$. However, for us, a property like

$$log.(a \times b) \ = \ log.a + log.b$$

is a distributivity property. Another distributivity property we have seen in section 2.3 is that *even* distributes over addition:

$$even.(a+b) \equiv even.a \equiv even.b \quad .$$

This property is useful, because we can reason about the parity of numbers within the Boolean domain, which is simpler than the domain of numbers. For example, consider the following problem:

Chess moves

In chess, a bishop moves along the diagonal. That is, starting from a position (i, j), a bishop can move a (positive or negative) distance k to the position $(i+k\,,\ j+k)$ or to the position $(i+k\,,\ j-k)$. (This is provided, of course, that the bishop stays within the boundary of the board. See figure 3.1; the bishop is in position $(2, 2)$.)

Show that a move from the position (i, j) to the position $(i+k\,,\ j+k)$ does not change the colour of the square. **Hint:** The following definition can be useful:

$$black.(i, j) \ \equiv \ even.i \ \equiv \ even.j.$$

The goal of this problem is to prove the following two equalities:

(3.3.2) $black.(i, j) \ \equiv \ black.(i+k\,,\ j+k)$

and

(3.3.3) $black.(i, j) \ \equiv \ black.(i+k\,,\ j-k)$.

A calculational proof of (3.3.2) is as follows:

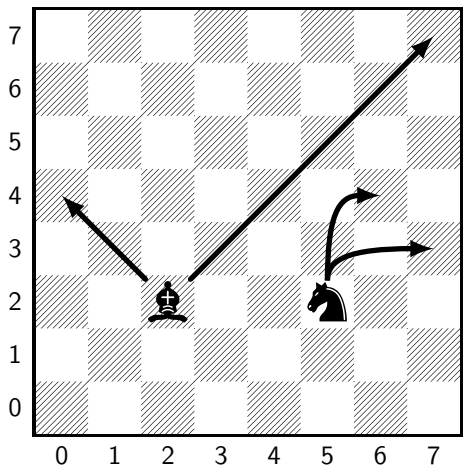


Figure 3.1: Examples of chess moves: bishop and knight

$$
\begin{array}{ll}
 & \mathit{black}.(i{+}k\,,\ j{+}k) \\
= & \quad \{ \quad \text{definition of } \mathit{black} \quad \} \\
 & \mathit{even}.(i{+}k) \equiv \mathit{even}.(j{+}k) \\
= & \quad \{ \quad \mathit{even} \text{ distributes over addition} \quad \} \\
 & \mathit{even}.i \equiv \mathit{even}.k \equiv \mathit{even}.j \equiv \mathit{even}.k \\
= & \quad \{ \quad \text{associativity and symmetry of } \equiv \quad \} \\
 & \mathit{even}.i \equiv \mathit{even}.j \equiv \mathit{even}.k \equiv \mathit{even}.k \\
= & \quad \{ \quad \text{associativity and reflexivity of } \equiv \quad \} \\
 & \mathit{even}.i \equiv \mathit{even}.j \\
= & \quad \{ \quad \text{definition of } \mathit{black} \quad \} \\
 & \mathit{black}.(i,j) \quad .
\end{array}
$$

The proof of (3.3.3) is similar. Note that the solution constitutes a unified interface for reasoning about how the colour of the squares change regardless of the chess piece. For example, we can use the same proof structure to prove that the move of a knight always changes the colour of the square (see figure 3.1; in this case, the key property is that the numbers 1 and 2 have different parities). Furthermore, it is easy to create new exercises just by choosing different moves.

Standard solutions to parity problems are usually done within the familiar domain of numbers. In this particular example, a standard solution would claim that the parities of $i{+}k{+}j{+}k$ and $i{+}j$ are the same. However, reasoning within the Boolean domain

can be more effective: the algebraic manipulations may be less familiar than ordinary arithmetic, but they are easier because the domain is much simpler. We show another example in scenario 5, where we use distributivity to solve a logic puzzle.

Distributivity and naming We conclude this section with the observation that distributivity can also be used to name the elements of a problem more effectively. An example where we have used it implicitly was when we named the elements of the problem "Jealous Couples". Recall that, if we distinguish the individual people as suggested in section 2.4, the initial state would be represented as

$$H_1,W_1,H_2,W_2,H_3,W_3 \;||\quad .$$

From this state, three possible transitions are to the following states:

$$H_2,W_2,H_3,W_3 \;||\; H_1,W_1 \quad ,$$

$$H_1,W_1,H_3,W_3 \;||\; H_2,W_2 \quad \text{, and}$$

$$H_1,W_1,H_2,W_2 \;||\; H_3,W_3 \quad .$$

We have seen that we can reduce these three transitions to one transition by using a notation that names couples, rather than naming individual people. In a way, we are factoring out the initial state and combining the three different states into one single state. To formulate this idea, let us write $s \rightarrow s'$ to denote a transition from state s to state s', and let us use the operator $\otimes$ to indicate a choice between transitions. Then, using i to denote the initial state, and a, b, and c to denote the three states shown above, we have:

$$(i \rightarrow a) \otimes (i \rightarrow b) \otimes (i \rightarrow c) \quad .$$

Now, using the operator $\oplus$ to indicate the unification of names, we can rewrite this expression as:

$$i \rightarrow (a \oplus b \oplus c) \quad ,$$

where $a \oplus b \oplus c$, in this particular case, would be $2C \;||\; 1C$. Clearly, the equality

$$i \rightarrow (a \oplus b \oplus c) = (i \rightarrow a) \otimes (i \rightarrow b) \otimes (i \rightarrow c)$$

has the same shape as definition (3.3.1).

3.4 Invariants

An invariant is a property or expression that remains constant throughout the execution of an algorithm. Because invariants are constant, they are often used to prove that certain states are not achievable. For this reason, they are sometimes called *safety properties*. Scenario 7, for example, is about a problem where an invariant is used to prove that a certain state is unreachable. Invariants can also be used in different ways. In chapter 1 we have shown how to verify a theorem using one of the invariants of Euclid's algorithm. In this section, we show two more examples of how invariants can be used to solve algorithmic problems. The first is a simple, recreational problem that can be used to introduce the notion of invariant. The second example shows the importance of invariants in algorithm construction: we develop an algorithm that constructs a witness for an existential theorem. It is also an example of how we can use algorithms to do mathematics.

Introducing invariants Invariants can be introduced via simple and recreational examples. For instance, consider the following problem, taken from [Bac07]:

> **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?

We are given the initial and final numbers of empty boxes and we are required to find the total number of boxes at the end of the process. This motivates the introduction of variables for these values; we use t to denote the total number of boxes, and e to denote the number of empty boxes. Initially, we know that $t = e = 11$. We want to determine the value of t after the small boxes are placed into the medium boxes.

Since the number of empty boxes that is selected is unknown, let us focus on the atomic action of placing boxes inside an empty box. Whenever we put eight boxes inside an empty box, the total number of boxes increases by eight and the number of empty boxes increases by seven. Therefore, the assignment that models this is:

$$t, e \;:=\; t{+}8, e{+}7 \quad .$$

An invariant of this type of assignment is easy to calculate. We seek some linear combination of t and e that remains constant after execution of the assignment, so we propose to calculate x and y, such that

$$(x{\cdot}t + y{\cdot}e)[t\,,e \;:=\; t{+}8\,,e{+}7] \;=\; x{\cdot}t + y{\cdot}e \quad .$$

In words, we want to calculate x and y, such that the value of $x{\cdot}t + y{\cdot}e$ after execution of the assignment $t\,,e := t{+}8\,,e{+}7$ remains the same. The calculation is straightforward:

$$\begin{array}{ll} & (x{\cdot}t + y{\cdot}e)[t\,,e \;:=\; t{+}8\,,e{+}7] \;=\; x{\cdot}t + y{\cdot}e \\ = & \quad \{ \quad \text{substitution} \quad \} \\ & x{\cdot}(t{+}8) + y{\cdot}(e{+}7) \;=\; x{\cdot}t + y{\cdot}e \\ = & \quad \{ \quad \text{arithmetic} \quad \} \\ & 8{\cdot}x + 7{\cdot}y \;=\; 0 \\ \Leftarrow & \quad \{ \quad \text{arithmetic} \quad \} \\ & x = 7 \;\wedge\; y = -8 \quad . \end{array}$$

Thus, an invariant of the assignment is $7{\cdot}t - 8{\cdot}e$. We know that its initial value is -11 (because $t = e = 11$). Since it is an invariant, its final value has to be -11. This means that on termination, when $e = 102$, we have

$$7{\cdot}t - 8{\cdot}102 \;=\; -11 \quad .$$

Therefore, the final value of t is 115. That is, at the end of the process there are 115 boxes.

This solution is also an example of appropriate naming: we have introduced only two variables, one to express the goal, and the other to model the concrete data given by the problem statement. If we had introduced variables for the numbers of small, medium, and large boxes, the solution would be more complicated.

Calculating an algorithm Invariants are very important for calculating algorithms. The following example shows the derivation of a non-trivial algorithm that constructs a witness for the existential theorem[3]:

[3]We write $a \backslash b$ to denote "a divides b". The derivation we show is essentially the same as in [Dij82b], but, to make it more accessible, we include more details.

Theorem For any odd $p \geq 1$, integer $K \geq 1$, and odd r, a value x exists such that

$$1 \leq x < 2^K \ \wedge \ 2^K \backslash (x^p - r) \ \wedge \ odd.x \quad .$$

The functional specification of the algorithm S that we want to construct is

$$\{ \ odd.p \wedge p \geq 1 \wedge K \geq 1 \wedge odd.r \ \}$$
$$S$$
$$\{ \ 1 \leq x < 2^K \ \wedge \ 2^K \backslash (x^p - r) \ \wedge \ odd.x \ \} \ .$$

Since the statement S is essentially a search process, it is reasonable to assume that it has to be a repetitive statement. This means that we should choose an invariant for S. A common technique to find an invariant from the postcondition is to replace a constant by a variable; here, we replace the constant K by a variable k, and we rewrite the postcondition to:

$$P \ \wedge \ k{=}K \quad ,$$

where

$$P \quad \equiv \quad 1 \leq x < 2^k \ \wedge \ 2^k \backslash (x^p - r) \ \wedge \ odd.x \quad .$$

Now, because we can easily find values for x and k that make P true ($k = x = 1$, for example), we choose P to be the invariant and the negation of $k = K$ to be the guard of the loop. This results in the following:

$$k \, , x \ := \ 1 \, , 1 \ ;$$
$$\{ \ \textbf{Invariant}: \ P \ \}$$
$$\mathsf{do} \ k \neq K \ \rightarrow \ k \, , x \ := \ k{+}1 \, , x{+}X \ \mathsf{od}$$
$$\{ \ P \ \wedge \ k = K \ \}$$

The loop terminates when $k = K$. Moreover, if P is an invariant, it will be true on termination and we can conclude that the loop establishes the theorem. So, the goal is to find a value for X that guarantees the invariance of P. The formal requirement is:

$$P[k \, , x \ := \ k{+}1 \, , x{+}X] \Leftarrow P \quad .$$

Expanding the expressions, the formal requirement is:

$$1 \leq x+X < 2^{k+1} \;\; \wedge \;\; 2^{k+1} \backslash ((x+X)^p - r) \;\; \wedge \;\; odd.(x+X)$$

$\Leftarrow$

$$1 \leq x < 2^k \;\; \wedge \;\; 2^k \backslash (x^p - r) \;\; \wedge \;\; odd.x \;\; .$$

Note that if $X = 0$, the only conjunct that can be violated is $2^{k+1} \backslash (x^p - r)$. So, an acceptable refinement for our loop is:

do $k \neq K \rightarrow$ if $2^{k+1} \backslash (x^p - r) \rightarrow k := k+1$

$\Box \;\; 2^{k+1} \not\backslash (x^p - r) \rightarrow k, x := k+1, x+X$

fi

od

In other words, whenever we have $2^{k+1} \backslash (x^p - r)$, we do not change the value of x. But what if $2^{k+1} \not\backslash (x^p - r)$? In this case, we want to compute an X such that

$$1 \leq x+X < 2^{k+1} \;\; \wedge \;\; 2^{k+1} \backslash ((x+X)^p - r) \;\; \wedge \;\; odd.(x+X)$$

$\Leftarrow$

$$P \;\; \wedge \;\; 2^{k+1} \not\backslash (x^p - r) \;\; .$$

Let us focus first on the middle conjunct:

$$2^{k+1} \backslash ((x+X)^p - r)$$

$=$ { By the Binomial Theorem, we have

$(x+X)^p = \langle \Sigma j : 0 \leq j \leq p : \binom{p}{j} \times x^{p-j} \times X^j \rangle$ }

$$2^k \times 2 \backslash (\langle \Sigma j : 0 \leq j \leq p : \binom{p}{j} \times x^{p-j} \times X^j \rangle - r)$$

$=$ { range disjunction, associativity, and

distributivity (we factor out X) }

$$2^k \times 2 \backslash ((x^p - r) + X \times \langle \Sigma j : 1 \leq j \leq p : \binom{p}{j} \times x^{p-j} \times X^{j-1} \rangle)$$

$\Leftarrow$ { we have that $2^k \backslash (x^p - r)$; assume that $X = 2^k$ to make

both terms divisible by 2^k }

$$2 \backslash ((\tfrac{x^p - r}{2^k}) + \langle \Sigma j : 1 \leq j \leq p : \binom{p}{j} \times x^{p-j} \times 2^{k \times (j-1)} \rangle)$$

$=$ { distributivity }

$$2 \backslash (\tfrac{x^p - r}{2^k}) \;\equiv\; \langle \equiv j : 1 \leq j \leq p : 2 \backslash (\binom{p}{j} \times x^{p-j} \times 2^{k \times (j-1)}) \rangle$$

$$
\begin{array}{ll}
= & \{ \quad 2^{k+1} \backslash (x^p - r), \ \text{i.e.} \ \ 2 \backslash (\frac{x^p - r}{2^k}) \equiv \mathsf{false}; \\
& \quad \text{also, } \langle \forall j : 2 \leq j \leq p : 2 \backslash 2^{k \times (j-1)} \rangle \quad \} \\
& \mathsf{false} \ \equiv \ 2 \setminus p \times x^{p-1} \\
= & \{ \quad \mathit{odd}.p \text{ and } \mathit{odd}.x \quad \} \\
& \mathsf{true} \ .
\end{array}
$$

The conclusion is that

$$2^{k+1} \backslash ((x+2^k)^p - r) \ \Leftarrow \ P \ \wedge \ 2^{k+1} \backslash (x^p - r) \quad .$$

We also have (the proof is easy and left to the reader):

$$
\begin{array}{l}
1 \leq x + 2^k < 2^{k+1} \quad \wedge \quad \mathit{odd}.(x+2^k) \\
\Leftarrow \\
1 \leq x < 2^k \ \wedge \ \mathit{odd}.x \ .
\end{array}
$$

Therefore, we conclude that

$$
\begin{array}{l}
P[k \,,\, x \ := \ k+1 \,,\, x+2^k] \\
\Leftarrow \\
P \ \wedge \ 2^{k+1} \backslash (x^p - r),
\end{array}
$$

and we rewrite the algorithm to:

$$
\begin{array}{l}
\{ \ \mathit{odd}.p \wedge p \geq 1 \wedge K \geq 1 \wedge \mathit{odd}.r \ \} \\
k \,,\, x \ := \ 1 \,,\, 1 \ ; \\
\{ \ \textbf{Invariant}\text{: } P \ \} \\
\mathsf{do} \ k \neq K \rightarrow \quad \mathsf{if} \ \ 2^{k+1} \backslash (x^p - r) \ \rightarrow \ k \ := \ k+1 \\
\qquad\qquad\qquad\qquad \Box \ \ 2^{k+1} \backslash (x^p - r) \ \rightarrow \ k \,,\, x \ := \ k+1 \,,\, x+2^k \\
\qquad\qquad\qquad\qquad \mathsf{fi} \\
\mathsf{od} \\
\{ \ P \ \wedge \ k = K \ \}
\end{array}
$$

The algorithm computes the value of 2^{k+1} twice and the value of 2^k once. We can optimise it by introducing a new variable d to hold the value of 2^k, but the goal of this

derivation is not to get the most efficient algorithm. Instead, we hope to have demonstrated how the notion of invariance can be used to calculate non-trivial algorithms. The only step where we had to do some guessing was the step where we assumed that X was 2^k (third step of the calculation). But we think it was a reasonable guess, completely guided by symmetry.

Finally, because $K \geq 1$, and because we are incrementing k by 1 at each step of the loop, the algorithm will reach a point at which $k = K$. Therefore, the algorithm terminates and the existence theorem is proved. In the next section, we discuss program termination in more detail.

In chapter 4, we show several derivations of algorithms, all of them based on the notion of invariance. Also, scenarios 7, 9, and 11 provide further examples of problems involving invariants.

3.5 Proving program termination

When we construct an algorithm, we have to guarantee that it makes progress towards the desired postcondition. For example, in the last algorithm of the previous section, we included a brief discussion on why the variable k would eventually become K. If we had omitted the termination argument, the theorem would not be established.

The classic method of proving that an algorithm terminates was proposed by Turing in a 1949 paper [MJ84], where he wrote:

> Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.

In this thesis, we use the term *bound function* and we define it to be a natural-number-valued function of the program variables that measures the size of the problem to be solved. A guarantee that the value of such a bound function is always decreased at each iteration is a guarantee that the number of times the loop body is executed is at most the initial value of the bound function.

For example, the function $K-k$ can be used as a bound function of the last algorithm shown in the previous section. Its initial value is $K-1$, and it clearly decreases at each iteration. Because the loop terminates when $k = K$, the function $K-k$ is bounded below.

Another example where it is easy to find a bound function is Euclid's algorithm. Recall

that, in chapter 1, we have used Euclid's algorithm to prove that the greatest common divisor of two consecutive Fibonacci numbers is 1:

$$
\begin{array}{l}
x\,,y \;:=\; \mathsf{fib}.(n{+}1)\,,\mathsf{fib}.n \;\;; \\
\{\;\textbf{Invariant}\colon \quad x \text{ and } y \text{ are two consecutive Fibonacci numbers} \\
\qquad\qquad\qquad \wedge \;\; \mathsf{fib}.(n{+}1)\;\mathsf{gcd}\;\mathsf{fib}.n \;=\; x\;\mathsf{gcd}\;y \;\;\} \\
\mathsf{do}\;\; y<x \;\rightarrow\; x \;:=\; x{-}y \\
\Box \;\; x<y \;\rightarrow\; y \;:=\; y{-}x \\
\mathsf{od} \\
\{\;x=y=1 \;\wedge\; \mathsf{fib}.(n{+}1)\;\mathsf{gcd}\;\mathsf{fib}.n \;=\; 1\;\}
\end{array}
$$

However, we have not proved that the algorithm terminates (we have just stated that it does). To prove that it terminates, we observe that at each step we either decrease x or we decrease y. As a result, the value of $x{+}y$ has to decrease; formally, we have to prove:

$$(x{+}y)[x \;:=\; x{-}y] < C \quad\Leftarrow\quad x{+}y = C \;\wedge\; y<x \quad \text{, and}$$

$$(x{+}y)[y \;:=\; y{-}x] < C \quad\Leftarrow\quad x{+}y = C \;\wedge\; x<y \quad .$$

In words, these requirements mean that if the value of $x{+}y$ before any assignment is C, it has to be less than C after the execution of an assignment. We prove the first and we leave the second for the reader:

$$
\begin{array}{ll}
 & (x{+}y)[x \;:=\; x{-}y] \\
= & \quad \{ \quad \text{substitution} \quad \} \\
 & (x{-}y){+}y \\
= & \quad \{ \quad \text{associativity and symmetry} \quad \} \\
 & (x{+}y){-}y \\
= & \quad \{ \quad x{+}y = C \quad \} \\
 & C{-}y \\
< & \quad \{ \quad 0<y \quad \} \\
 & C \;\;.
\end{array}
$$

Termination is not always as obvious as in the previous examples. For a more challenging problem, consider the following game, taken from [vG90]:

> We are requested to provide an argument for the termination of the following game: a finite bit string (i.e. a string of zeroes and ones) is repeatedly transformed by replacing
>
> a pattern 00 by 01 , or
> a pattern 11 by 10 , wherever in the string and as long as such transformations are possible.

The solution we show here is Van Gasteren's solution. Since the pair of transformations is invariant under an interchange of 0 and 1, only equality and difference of bits matter. Exploiting this observation, we record the succession of neighbour equalities and differences in the bit string as a string of y's and x's, with

y standing for a pair of equal neighbour bits, and
x standing for a pair of different neighbour bits

(which given the first bit precisely determines the bit string).

In this terminology, a transformation changes a y in the "code string" into an x, while leaving all elements to the left of that y unchanged. Thus the code string decreases lexically at each transformation. Since it furthermore is lexically bounded from below — by the string of appropriate length consisting of x's only — the game terminates.

(The shape of the bit string upon termination follows from the observation that the leftmost bit of the bit string does not change in the game and that upon termination the code string consists of x's only.)

The bound function is the lexical ordering of the code string; since the strings are enumerable, the function can be seen as a natural-valued-function that maps a code string to the correspondent natural number (e.g., the lexically smallest code string is mapped to zero).

This argument is also a good example of appropriate naming: we named neither the lengths nor the individual elements of the bit and code strings, and we only had to consider one change of one symbol (namely, of a y into an x).

In scenario 10, we show another non-trivial problem on termination, and we include this game as an exercise.

3.6 Algorithm Inversion

Inverting an algorithm S consists in finding another algorithm, usually denoted by S^{-1}, that when composed with S leaves the program state unchanged. In other words,

executing S^{-1} after S amounts to doing nothing, that is, if we provide to S^{-1} some output of S, it will compute a corresponding input to S.

Some statements are easy to invert. The inverse of skip, for example, is skip itself. Also, the inverse of $x := x-y$ is $x := x+y$. However, other statements are difficult or impossible to invert. For example, we cannot invert $x := 1$ without knowing the value of x before the assignment; we can only invert it if we know the precondition. The inverse of

$$\{\ x = 0\ \}\ \ x\ :=\ 1$$

is

$$\{\ x = 1\ \}\ \ x\ :=\ 0\ \ .$$

Note that the assertion becomes an assignment and the assignment becomes an assertion. This simple example shows that we may be able to compute inverses only when the precondition is given. Therefore, we define the inverse of a statement with respect to a precondition. That is, S^{-1} is the (right) inverse of S with respect to R, if for every Q

$$\{\ R \wedge Q\ \}\ \ S\,;S^{-1}\ \{\ Q\ \}\ \ .$$

An important aspect of the above characterisation is that it distributes through program constructs. This allows us to reduce the inversion of a program into the inversion of its components. For example, the inverse of a sequence of commands is the reverse of the sequence of inverses of the individual commands:

$$(S_0;S_1;\cdots;S_n)^{-1}\ \ =\ \ S_n^{-1};\cdots;S_1^{-1};S_0^{-1}\ \ .$$

To illustrate this rule, suppose that we want to invert the following program, taken from the scenario 8:

$$\{\ \ x = X\ \ \wedge\ \ y = Y\ \ \}$$

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

$$y\ :=\ x-y\ ;$$

$$x\ :=\ x-y$$

$$\{\ \ x = Y\ \ \wedge\ \ y = X\ \ \}\ \ .$$

As seen in section 2.2 (page 25), this program swaps the values of variables x and y without using a temporary variable (we are assuming that overflows do not occur). Using the rule, the inverse of the program is:

$$\{\ x = Y\ \wedge\ y = X\ \}$$
$$(x\ :=\ x{-}y)^{-1};$$
$$(y\ :=\ x{-}y)^{-1};$$
$$(x\ :=\ x{+}y)^{-1}$$
$$\{\ x = X\ \wedge\ y = Y\ \}\ .$$

Note that, as expected, the precondition and the postcondition are swapped. Also, the inverse of $x := x{-}y$ is $x := x{+}y$, and vice-versa. To calculate the inverse of $y := x{-}y$, we first note that it is the same as $y := -(y{-}x)$, which is equivalent to $y := y{-}x\ ;\ y := -y$. The inverse of this sequence is $y := -y\ ;\ y := y{+}x$, which is equivalent to $y := -y{+}x$. Therefore, the assignment $y := x{-}y$ is its own inverse. Applying this to the program, we have

$$\{\ x = Y\ \wedge\ y = X\ \}$$
$$x := x{+}y\ ;$$
$$y\ :=\ x{-}y\ ;$$
$$x := x{-}y$$
$$\{\ x = X\ \wedge\ y = Y\ \}\ .$$

Comparing the original program and its inverse, we conclude that the program is its own inverse. This is not surprising, because swapping the values of two variables twice is the same as doing nothing.

Now, if c_0 and c_1 are constants, the inverse of

(3.6.1) $v\ :=\ c_0\ ;\ S\ \{\ v = c_1\ \}$

is

$$v\ :=\ c_1\ ;\ S^{-1}\ \{\ v = c_0\ \}\ .$$

In (3.6.1), variable v is initialised to a value c_0, S is executed, and upon termination v has the final value c_1. The inverse assigns c_1 to v, undoes what S did, and terminates with $v = c_0$. Note, again, how the assignment and the assertion switch places.

Another important inversion rule concerns alternative commands. Suppose we want to invert the following command:

$\{\ G_0 \vee G_1\ \}$

if $G_0 \rightarrow S_0\ \{\ C_0\ \}$

$\square\ G_1 \rightarrow S_1\ \{\ C_1\ \}$

fi

$\{\ C_0 \vee C_1\ \}$.

Execution must begin with one of the guards true, so the disjunction of the guards has been placed before the statement. Execution terminates with either C_0 or C_1 true, depending on which command is executed, so $C_0 \vee C_1$ is the postcondition. To invert this command we must know whether to perform the inverse of S_0 or to perform the inverse of S_1. Therefore, C_0 and C_1 cannot be true at the same time (i.e., $\neg(C_0 \wedge C_1)$). For symmetry, we also require $\neg(G_0 \wedge G_1)$. Because the command ends in a state satisfying $C_0 \vee C_1$, its inverse must begin in a state satisfying $C_0 \vee C_1$. Also, execution of $G_1 \rightarrow S_1\ \{\ C_1\ \}$ means that beginning with G_1 true, S_1 is executed, and C_1 is established. The inverse must express that beginning with C_1 true, S_1 is undone, and G_1 is established:

$C_1 \rightarrow S_1^{-1}\ \{\ G_1\ \}$.

Note how, when inverting a guarded command with a postcondition, the guard and postcondition switch places. Continuing to read backwards yields the inverse of the alternative command:

$\{\ C_0 \vee C_1\ \}$

if $C_1 \rightarrow S_1^{-1}\ \{\ G_1\ \}$

$\square\ C_0 \rightarrow S_0^{-1}\ \{\ G_0\ \}$

fi

$\{\ G_0 \vee G_1\ \}$.

The final rule we will see is used to invert iterative commands. Suppose we have a loop that executes while G_0 is true:

do $G_0 \rightarrow S_0$ od $\{\ \neg G_0\ \}$.

Clearly, the inverse of this loop has to have $\neg G_0$ as its precondition; also, its postcondition has to be the precondition of this loop. Therefore, based on what we have seen for the alternative command, we add the following assertions:

$$\{\ \neg G_1\ \}\ \mathsf{do}\ G_0 \rightarrow S_0\ \{\ G_1\ \}\ \mathsf{od}\ \{\ \neg G_0\ \}\ .$$

The inverse of this command can now be obtained by reading it backwards, as we have done for the alternative command. Its inverse is:

$$\{\ \neg G_0\ \}\ \mathsf{do}\ G_1 \rightarrow S_0^{-1}\ \{\ G_0\ \}\ \mathsf{od}\ \{\ \neg G_1\ \}\ .$$

Finally, because our syntax allows non-determinacy, it is common to have programs of the following shape:

$$\begin{array}{l}
\{\ G_0 \vee G_1\ \} \\
\mathsf{do}\ \ G_0 \ \rightarrow\ S_0\ \{\ C_0\ \} \\
\Box\ \ G_1 \ \rightarrow\ S_1\ \{\ C_1\ \} \\
\mathsf{od} \\
\{\ C_0 \vee C_1\ \}\ .
\end{array}$$

If both the guards G_0 and G_1 are true, the block operator ($\Box$) ensures that one of the statements S_0 and S_1 is chosen non-deterministically. If both the guards are false, the loop terminates properly (in fact, in that case, it is the same as doing skip)[4].

To invert such a program, we can use the inversion rule for alternative statements together with the inversion rule for iterative statements. Provided that $\neg(C_0 \wedge C_1)$ and $\neg(G_0 \wedge G_1)$, the inverse of the program is:

$$\begin{array}{l}
\{\ C_0 \vee C_1\ \} \\
\mathsf{do}\ \ C_1 \ \rightarrow\ S_1^{-1}\ \{\ G_1\ \} \\
\Box\ \ C_0 \ \rightarrow\ S_0^{-1}\ \{\ G_0\ \} \\
\mathsf{od} \\
\{\ G_0 \vee G_1\ \}\ .
\end{array}$$

In section 4.6, this rule is used to invert Euclid's algorithm. For more details on the inversion rules shown in this section, we recommend the expositions in [Gri81, chapter 21] and [vdS93, chapter 11]. As far as we know, the technique of program inversion first appeared in [Dij82a, pp. 351–354] and, since then, it has been mentioned and used in many places (see, for example, [vdS91, Che90, vW91, MB03]).

[4]Note that if all the guards are false in the alternative statement of the shape if $\cdots$ fi, then proper termination does not occur (in that case, it is the same as doing abort).

Although a formal approach to program inversion can be used to prove certain arguments more precisely, the technique is not widespread. We believe it deserves to be better known. We use it in section 4.6 to make an argument more precise.

CHAPTER 4

A Calculational and Algorithmic Approach to Elementary Number Theory

The elementary theory of numbers should be one the very best subjects for early mathematical instruction. It demands very little previous knowledge, its subject matter is tangible and familiar; the processes of reasoning which it employs are simple, general and few; and it is unique among the mathematical sciences in its appeal to natural human curiosity.

— G. H. HARDY (1928)

4.1 Introduction

This chapter, which is based on [BF08], [BF10], and [Fer10], presents a calculational and algorithmic approach to elementary number theory, a theory concerned with the properties of the integer numbers. In other words, we use the principles and techniques described in previous chapters to reason about numbers and some of their properties (especially divisibility properties). In our view, the algorithmic nature of some number-theoretical concepts and the points highlighted in the opening quote of this chapter justify well why number theory is a good subject to be rewritten with a focus on algorithmic content. We hope to show that our reformulation can be used to convey principles of algorithmic development to mathematics students. Moreover, since number theory forms the mathematical foundations of cryptography, we believe it can also be useful for computing science students.

Our approach is unconventional, mainly because traditional presentations of number

theory have benefited little from the advances that have been made in our understanding of the basic principles of algorithm development. A blatant example is the conventional treatment of Euclid's algorithm to compute the greatest common divisor (gcd) of two positive natural numbers, the oldest nontrivial algorithm that involves iteration and that has not been superseded by algebraic methods. (For a modern paraphrase of Euclid's original statement, see [Knu97, pp. 335–336].) Most books on number theory include Euclid's algorithm, but rarely use the algorithm directly to reason about properties of numbers. In a thesis such as this one, it is of course not the place to rewrite mathematics textbooks. Nevertheless, our goal in this chapter is to demonstrate how a focus on algorithmic method can enrich and reinvigorate the teaching of mathematics. We use Euclid's algorithm to derive both old and well-known, and new and previously unknown, properties of the greatest common divisor and rational numbers. The leitmotiv is the notion of a loop invariant — how it can be used as a verification interface (i.e., how to verify theorems) and as a construction interface (i.e., how to investigate and derive new theorems).

We begin in section 4.2 with the construction of the integer division algorithm, with basic properties of the division relation, and with the construction of Euclid's algorithm. In contrast to standard presentations of the algorithm, which typically assume the existence of the gcd operator with specific algebraic properties, our derivation gives a constructive proof of the existence of an infimum operator in the division ordering of natural numbers.

The focus of section 4.3 is the systematic use of invariant properties of Euclid's algorithm to verify known identities. Section 4.4, on the other hand, shows how to use the algorithm to derive new results related with the greatest common divisor: we calculate sufficient conditions for a natural-valued function[1] to distribute over the greatest common divisor, and we derive an efficient algorithm to enumerate the positive rational numbers in two different ways.

Although the identities in section 4.3 are well-known, we believe that our derivations improve considerably on standard presentations. One example is the proof that the greatest common divisor of two numbers is a linear combination of the numbers; by the simple device of introducing matrix arithmetic into Euclid's algorithm, it suffices to observe that matrix multiplication is associative in order to prove the theorem. This exemplifies the gains in our problem-solving skills that can be achieved by the right combination of precision and concision. The introduction of matrix arithmetic at this early stage was also what enabled us to derive a previously unknown algorithm to

[1] We call a function natural-valued if its range is the set of natural numbers.

enumerate the rationals in so-called Stern-Brocot order (see section 4.4), which is one of the primary novel results (as opposed to method) in this chapter.

One of the main tools provided by number theory is modular arithmetic. Therefore, in section 4.5, we illustrate how the theory of congruences can be made more calculational, we show how we can construct an algorithm to do modular exponentiation, and we construct a simple version of the Chinese remainder theorem in two different ways. This section was jointly written with Arjan Mooij.

The final problem that we solve in this chapter is one that has received a lot of attention: which numbers can be written as the sum of two squares? There is a well-known theorem due to Albert Girard (we call it the two-squares theorem) that answers this question and that is usually verified in elementary number theory books. In section 4.6, we show a new and constructive proof of the two-squares theorem, based on our formulation of Euclid's algorithm expressed in terms of matrices. Rather than simply verifying the result — as it is usually done in the mathematical community — we use Euclid's algorithm as an interface to *investigate* which numbers can be written as sums of two positive squares. The precise formulation of the problem as an algorithmic problem is the key, since it allows us to use algorithmic techniques and to avoid guessing. The notion of invariance, in particular, plays a central role in our development: it is used initially to observe that Euclid's algorithm can actually be used to represent a given number as a sum of two positive squares, and then it is used throughout the argument to prove other relevant properties. We also show how the use of program inversion techniques can make mathematical arguments more precise. The theorem that we derive is more general than the one conjectured by Girard.

We finish the chapter with a brief summary of the work of Stern and Brocot, the 19th century authors after whom the Stern-Brocot tree is named. It is interesting to review their work, particularly that of Brocot, because it is clearly motivated by practical, algorithmic problems. The review of Stern's paper was written by Roland Backhouse, since the author of this dissertation does not read any German. We include it in this thesis, because we believe that the historical perspective enriches the material of section 4.4. It also resolves recent misunderstandings about the origin of the Eisenstein-Stern and Stern-Brocot enumerations of the rationals.

Most of the material shown in this chapter was published before. Sections 4.2, 4.3, 4.4, and the appendix with the summary of the work of Stern and Brocot were published in [BF10]. The only exception is the derivation of the division algorithm shown in section 4.2. Also, section 4.6 was published in [Fer10]. The only section of this chapter that was never published, and is therefore new, is section 4.5.

4.2 Divisibility theory

Division is one of the most important concepts in number theory. This section begins with a derivation of the division algorithm from its formal specification. We also give a short, basic account of the division relation. We observe that division is a partial ordering on the natural numbers and pose the question whether the infimum, in the division ordering, of any pair of numbers exists. The algorithm we know as Euclid's gcd algorithm is then derived in order to give a positive (constructive) answer to this question.

4.2.1 Integer division

Integer division as a Galois connection

The integer division of P by Q, here denoted by $P \div Q$, is usually introduced as the integer x such that

$$P \;=\; x \times Q + r \quad \wedge \quad 0 \leq r < |Q| \quad .$$

(Note that this definition rounds down, rather than rounding towards zero; for example, $(-7) \div 2$ is -4, not -3.) This formulation is usually accompanied by many examples that convey the concept of division, dividend, and remainder. However, we believe that the students do not learn how to reason effectively about division. Properties like the following[2]

(4.2.1) $[\ (a \div b) \div c = a \div (c \times b)\ ]$

are rarely discussed, and even when they are, their justification is typically informal and imprecise. Note, however, that this sort of property is often given as a *rule of thumb* in connection to exercises. Properly understanding them becomes relevant to build the correct underlying mathematical intuitions. Therefore, we propose the introduction of the integer division of P by Q as the Galois connection[3]:

(4.2.2) $\langle \forall k ::\ k \times Q \leq P \;\equiv\; k \leq P \div Q \rangle$.

[2]The square so-called "everywhere" brackets are used to indicate that a Boolean statement is "everywhere" true. That is, the statement has the value true for all instantiations of its free variables. Such statements are often called "facts", or "laws", or "theorems".

When using the everywhere brackets, the domain of the free variables has to be made clear. This is particularly important here because sometimes the domain of a variable is the integers and sometimes it is the natural numbers. Usually, we rely on a convention for naming the variables, but sometimes we preface a law with a reminder of the domain.

[3]We use a systematic notation for quantified expressions: the Eindhoven quantifier notation (mentioned in section 2.2 and already used in page 29). For more details, see [Bac03, chapter 11] and [GS93,

This definition requires that Q is a natural number; other requirements on the variables involved will emerge later. We can use this definition to effectively prove properties of integer division. For instance, replacing k by $P \div Q$, we establish the property:

$$(P \div Q) \times Q \leq P \quad .$$

We can also conclude that $0 \leq P$ is equivalent to $0 \leq P \div Q$, by replacing k by 0. Also, using indirect equality, definition (4.2.2) can be used to prove property (4.2.1) in just three steps:

$$\begin{array}{ll} & k \leq (a \div b) \div c \\ = & \quad \{ \quad \text{definition (4.2.2)} \quad \} \\ & k \times c \leq a \div b \\ = & \quad \{ \quad \text{definition (4.2.2) and associativity} \quad \} \\ & k \times (c \times b) \leq a \\ = & \quad \{ \quad \text{definition (4.2.2)} \quad \} \\ & k \leq a \div (c \times b) \quad . \end{array}$$

Moreover, definition (4.2.2) is a suitable specification for an algorithm that computes $P \div Q$. Note that the goal of such an algorithm is to compute a solution to the equation[4]

$$x :: \ \langle \forall k ::\ k \times Q \leq P \ \equiv \ k \leq x \rangle \quad .$$

If a solution to this equation exists, then it is unique (because the relation $\leq$ is reflexive and anti-symmetric). Furthermore, an important property of the solution x follows from instantiating k with x in (4.2.2):

(4.2.3) $\quad x \times Q \leq P \quad .$

Instantiating k with $x+1$ in (4.2.2), we also have:

(4.2.4) $\quad \neg((x+1) \times Q \leq P) \quad .$

In fact, x is the largest integer that satisfies (4.2.3). Properties (4.2.3) and (4.2.4) are the only ingredients we need to specify the division algorithm:

chapter 8].

Recall that the symbol $\equiv$ denotes Boolean equality. In continued expressions $\equiv$ is read associatively and $=$ is read conjunctionally. For example, $p \equiv q \equiv r$ is evaluated associatively—i.e. as $(p \equiv q) \equiv r$ or $p \equiv (q \equiv r)$, whichever is most convenient—whereas $p = q = r$ is evaluated conjunctionally—i.e. $p = q$ and $q = r$.

[4]The notation $x :: E$ means that x is the unknown and the other free variables are parameters of the equation E.

$$S$$

$$\{\ x{\times}Q \leq P\ \ \wedge\ \ \neg((x{+}1){\times}Q \leq P)\ \}\ .$$

We now apply a common technique in algorithm development: we take the first conjunct as the invariant, since it is easy to initialise ($x\ :=\ 0$), and we take the negation of the second conjunct as the loop guard. The first version of the algorithm becomes[5]:

$$\{\ 0 \leq P\ \}$$

$$x\ :=\ 0\,;$$

$$\{\ \textbf{Invariant:}\quad x{\times}Q \leq P\ \}$$

$$\mathsf{do}\ \ (x{+}1){\times}Q \leq P\ \rightarrow\ x\ :=\ A$$

$$\mathsf{od}$$

$$\{\ x{\times}Q \leq P\ \ \wedge\ \ \neg((x{+}1){\times}Q \leq P)\ \}\ .$$

The precondition $0 \leq P$ is necessary to make the invariant initially valid. Now, calculating the assignment to x, so that the invariant is preserved, is the same as calculating A in a way that the following requirement is satisfied:

$$A{\times}Q \leq P\ \Leftarrow\ x{\times}Q \leq P\ \ \wedge\ \ (x{+}1){\times}Q \leq P\ \ .$$

Clearly, we can choose A to be $x{+}1$ and we get the next version of the algorithm:

$$\{\ 0 \leq P\ \}$$

$$x\ :=\ 0\,;$$

$$\{\ \textbf{Invariant:}\quad x{\times}Q \leq P\ \}$$

$$\mathsf{do}\ \ (x{+}1){\times}Q \leq P\ \rightarrow\ x\ :=\ x{+}1$$

$$\mathsf{od}$$

$$\{\ x{\times}Q \leq P\ \ \wedge\ \ \neg((x{+}1){\times}Q \leq P)\ \}\ .$$

Termination proof To prove that the algorithm terminates, we have to define a *bound function*, which is a natural-number-valued function of the program variables that measures the size of the problem to be solved. A guarantee that the value of such a bound

[5]We use the Guarded Command Language (GCL), a very simple programming language with just four programming constructs — assignment, sequential composition, conditionals, and loops. The GCL was introduced by Dijkstra [Dij75]. The statement do S od is a loop that executes S repeatedly while at least one of S's guards is true. Expressions in curly brackets are assertions.

function is always decreased at each iteration is a guarantee that the number of times the loop body is executed is at most the initial value of the bound function.

In this case, a good candidate is the function $P-x$; we need to verify that

$$\{\ P-x=C\ \}$$
$$x\ :=\ x+1$$
$$\{\ P-x<C\ \}\ .$$

The formal requirement is

$$P-(x+1)<C \Leftarrow P-x=C\ \ ,$$

and the proof is very simple:

$$\begin{array}{ll} & P-(x+1)<C \\ = & \quad \{\quad \text{distributivity and associativity} \quad \} \\ & (P-x)-1<C \\ = & \quad \{\quad P-x=C \quad \} \\ & C-1<C \\ = & \quad \{\quad \text{integer inequality} \quad \} \\ & \texttt{true}\ . \end{array}$$

That the function is bounded below follows from the invariant and from the guard:

$$\begin{array}{ll} & 0\le P-x \\ = & \quad \{\quad \text{cancellation} \quad \} \\ & x\le P \\ = & \quad \{\quad \text{we know from the invariant that } x\times Q\le P; \\ & \qquad \textbf{assuming that } \mathbf{0<Q}\text{, we have } x\le x\times Q; \\ & \qquad \text{because } \le \text{ is transitive, we also have } x\le P \quad \} \\ & \texttt{true}\ . \end{array}$$

Note that the assumption $0<Q$, highlighted in bold in the calculation, emerges naturally from the shape of the invariant. As a result, to guarantee termination, we have to include $0<Q$ as a precondition of the algorithm:

$$
\begin{array}{l}
\{\ 0<Q\ \wedge\ 0\le P\ \}\\
x\ :=\ 0\,;\\
\{\ \textbf{Invariant:}\quad x\times Q\le P\ \}\\
\mathsf{do}\ \ (x{+}1)\times Q\le P\ \rightarrow\ x\ :=\ x{+}1\\
\mathsf{od}\\
\{\ x\times Q\le P\ \ \wedge\ \ \neg((x{+}1)\times Q\le P)\ \}\ .
\end{array}
$$

Refining the guard The current version of the algorithm is computing the value $x{+}1$ twice: once in the guard and once in the loop body. A good improvement would be to remove its computation from the guard. In order to do that, we first observe that

$$
\begin{array}{ll}
 & (x{+}1)\times Q\le P\\
= & \quad \{\quad \text{distributivity}\quad \}\\
 & x\times Q+Q\ \le\ P\\
= & \quad \{\quad \text{cancellation}\quad \}\\
 & Q\ \le\ P-x\times Q\ .
\end{array}
$$

This suggests the introduction of a variable that equals $P-x\times Q$. Calling this variable r and adding the equality

$$r\ =\ P-x\times Q$$

to the invariant, we get the following algorithm:

$$
\begin{array}{l}
\{\ 0<Q\ \wedge\ 0\le P\ \}\\
x,r\ :=\ 0,P\ ;\\
\{\ \textbf{Invariant:}\quad x\times Q\le P\ \wedge\ r\ =\ P-x\times Q\ \}\\
\mathsf{do}\ \ Q\le r\ \rightarrow\ x,r\ :=\ x{+}1,B\\
\mathsf{od}\\
\{\ x\times Q\le P\ \ \wedge\ \ \neg((x{+}1)\times Q\le P)\ \wedge\ r\ =\ P-x\times Q\ \}\ .
\end{array}
$$

Clearly, to satisfy the invariant, the initial value of r must be P. The next step is to calculate the assignment to r, that is, to calculate B in a way that the following holds:

$$\{\ r = P - x\times Q\ \wedge\ Q \le r\ \}$$
$$x, r\ :=\ x{+}1, B$$
$$\{\ r = P - x\times Q\ \}\ .$$

Using the assignment axiom, we determine B as follows:

$$\begin{aligned} & B = P - (x{+}1)\times Q \\ = & \quad \{\quad \text{distributivity} \quad \} \\ & B = P - x\times Q - Q \\ = & \quad \{\quad \text{invariant } r = P - x\times Q \text{ and cancellation} \quad \} \\ & B = r - Q\ . \end{aligned}$$

Therefore, the final version of the algorithm is:

$$\{\ 0 < Q\ \wedge\ 0 \le P\ \}$$
$$x, r\ :=\ 0, P\ ;$$
$$\{\ \textbf{Invariant:}\quad x\times Q \le P\ \wedge\ r = P - x\times Q\ \}$$
$$\mathsf{do}\ \ Q \le r\ \rightarrow\ x, r\ :=\ x{+}1, r{-}Q$$
$$\mathsf{od}$$
$$\{\ x\times Q \le P\ \wedge\ \neg((x{+}1)\times Q \le P)\ \wedge\ r = P - x\times Q\ \}\ .$$

Deriving a recursive algorithm To illustrate the flexibility of the definition of $P \div Q$ as a Galois connection, we now show how to calculate a recursive definition of $P \div Q$ for natural P and positive Q. The first step is to express $P \div Q$ in terms of the operator $\div$, but with the first argument reduced:

$$\begin{aligned} & k \le P \div Q \\ = & \quad \{\quad (4.2.2) \quad \} \\ & k\times Q \le P \\ = & \quad \{\quad \text{cancellation} \quad \} \\ & k\times Q - Q \le P - Q \\ = & \quad \{\quad \text{distributivity, } 0 < Q, \text{ and } (4.2.2); \text{ the construction assumes} \\ & \qquad \text{that the first argument is a natural number, so we need to} \end{aligned}$$

$$\begin{array}{ll} & \quad \{ \quad \text{guarantee that } Q \leq P \quad \} \\ & k-1 \leq (P-Q) \div Q \\ = & \quad \{ \quad \text{cancellation} \quad \} \\ & k \leq (P-Q) \div Q + 1 \ . \end{array}$$

Hence, by indirect equality, the following property holds:

$$P \div Q = (P-Q) \div Q + 1 \Leftarrow Q \leq P \ .$$

We now consider the case when $P < Q$:

$$\begin{array}{ll} & k \leq P \div Q \\ = & \quad \{ \quad (4.2.2) \quad \} \\ & k \times Q \leq P \\ = & \quad \{ \quad \text{transitivity } (k \times Q \leq P \text{ and } P < Q); \text{ cancellation} \quad \} \\ & k \leq 0 \ . \end{array}$$

Again, by indirect equality, we conclude:

$$P \div Q = 0 \Leftarrow P < Q \ .$$

Putting the two cases together, we have the following recursive definition (for natural P and positive Q):

$$\begin{array}{lllll} P \div Q & | & P < Q & = & 0 \\ P \div Q & | & Q \leq P & = & (P-Q) \div Q + 1 \ . \end{array}$$

This function clearly terminates, as it stops when $P < Q$ and P is being reduced at each recursive step.

Some final remarks We would like to stress that the derivation of the division algorithm is an educational example that can be used to teach algorithmic techniques such as loop formation, using the invariant to calculate assignments, and proving progress. It can also be used to teach the technique of introducing extra variables and computations to produce more efficient versions.

As we have seen, the definition based on a Galois connection is so flexible that we can not only use it to prove properties on division, but also to derive two different

algorithms: one iterative and one recursive! We think that this definition should be better known and more used in school mathematics.

Also, the calculational approach allows us to be more constructive because the requirements emerge from the calculations. As an example, the requirement that the divisor Q is positive emerges as a necessary condition in the proof that the bound function is bounded below. Another example is the calculation of the assignment to the variable r; rather than guessing the assignment and making a post-hoc verification, we have calculated it with no guessing involved!

4.2.2 Division relation

The division relation, here denoted by an infix "$\backslash$" symbol, is the relation on integers defined to be the converse of the "is-a-multiple-of" relation[6]:

$$[\ m\backslash n\ \equiv\ \langle \exists k : k\in\mathbb{Z} : n = k\times m\rangle\ ]\quad .$$

In words, an integer m divides an integer n (or n is divisible by m) if there exists some integer k such that $n = k\times m$. In that case, we say that m is a divisor of n and that n is a multiple of m.

The division relation plays a prominent role in number theory. So, we start by presenting some of its basic properties and their relation to addition and multiplication. First, it is reflexive because multiplication has a unit (i.e., $m = 1\times m$) and it is transitive, since multiplication is associative. It is also (almost) preserved by linear combination because multiplication distributes over addition:

$$(4.2.5)\quad [\ k\backslash x \wedge k\backslash y\ \equiv\ k\backslash(x + a\times y) \wedge k\backslash y\ ]\quad .$$

(We leave the reader to verify this law; take care to note the use of the distributivity of multiplication over addition in its proof.) Reflexivity and transitivity make division a *preorder* on the integers. It is not anti-symmetric but the numbers equivalent under the preordering are given by

$$[\ m\backslash n \wedge n\backslash m\ \equiv\ \mathsf{abs}.m = \mathsf{abs}.n\ ]\quad ,$$

where abs is the absolute value function and the infix dot denotes function application. Each equivalence class thus consists of a natural number and its negation. If the division relation is restricted to natural numbers, division becomes anti-symmetric, since

[6] Although the notation $m|n$ is more common, we prefer to use an asymmetric symbol such as the backward slash to denote an asymmetric relation. As the authors of [GKP94, p. 102] point out, vertical bars are overused and $m\backslash n$ gives an impression that m is the denominator of an implied ratio.

abs is the identity function on natural numbers. This means that, restricted to the natural numbers, division is a *partial* order with 0 as the greatest element and 1 as the smallest element.

Infimum in the division ordering

The first question that we consider is whether two arbitrary natural numbers m and n have an infimum in the division ordering. That is, can we solve the following equation[7]?

(4.2.6) $x::\quad \langle \forall k \;::\; k\backslash m \wedge k\backslash n \;\equiv\; k\backslash x \rangle$.

The answer is not immediately obvious because the division ordering is partial. (With respect to a total ordering, the infimum of two numbers is their minimum; it is thus equal to one of them and can be easily computed by a case analysis.)

If a solution to (4.2.6) exists, it is unique (because the division relation on natural numbers is reflexive and anti-symmetric). When it does have a solution, we denote it by $m \triangledown n$. That is, provided it can be established that (4.2.6) has a solution,

(4.2.7) $[\; k\backslash m \wedge k\backslash n \;\equiv\; k \setminus (m \triangledown n) \;]$.

Because conjunction is idempotent,

$$[\; k\backslash m \wedge k\backslash m \;\equiv\; k\backslash m \;]\ .$$

That is, m solves (4.2.6) when m and n are equal. Also, because $[\; k\backslash 0 \;]$,

$$[\; k\backslash m \wedge k\backslash 0 \;\equiv\; k\backslash m \;]\ .$$

That is, m solves (4.2.6) when n is 0. So, $m \triangledown m$ exists as does $m \triangledown 0$, and both equal m:

(4.2.8) $[\; m \triangledown m \;=\; m \triangledown 0 \;=\; m \;]$.

Other properties that are easy to establish by exploiting the algebraic properties of conjunction are, first, $\triangledown$ is symmetric (because conjunction is symmetric)

(4.2.9) $[\; m \triangledown n \;=\; n \triangledown m \;]$,

and, second, $\triangledown$ is associative (because conjunction is associative)

(4.2.10) $[\; (m \triangledown n) \triangledown p = m \triangledown (n \triangledown p) \;]$.

[7] Unless indicated otherwise, the domain of all variables is $\mathbb{N}$, the set of natural numbers. Note that we include 0 in $\mathbb{N}$.

Note that we choose infix notation for $\triangledown$, since it allows us to write $m \triangledown n \triangledown p$ without having to choose between $(m \triangledown n) \triangledown p$ or $m \triangledown (n \triangledown p)$. (See the discussion on infix notation in section 2.2.)

The final property of $\triangledown$ that we deduce from (4.2.7) is obtained by exploiting (4.2.5), with x and y replaced by m and n, respectively :

(4.2.11) $[\ (m + a \times n) \triangledown n \ = \ m \triangledown n\ ]$.

4.2.3 Constructing Euclid's algorithm

At this stage in our analysis, properties (4.2.9), (4.2.10) and (4.2.11) assume that equation (4.2.6) has a solution in the appropriate cases. For instance, (4.2.9) means that, if (4.2.6) has a solution for certain natural numbers m and n, it also has a solution when the values of m and n are interchanged.

In view of properties (4.2.8) and (4.2.9), it remains to show that (4.2.6) has a solution when both m and n are strictly positive and unequal. We do this by providing an algorithm that computes the solution. Equation (4.2.6) does not directly suggest any algorithm, but the germ of an algorithm is suggested by observing that it is equivalent to

(4.2.12) $x, y ::\quad x = y \ \wedge \ \langle \forall k ::\ k \backslash m \wedge k \backslash n \ \equiv \ k \backslash x \wedge k \backslash y \rangle$.

This new shape strongly suggests an algorithm that, initially, establishes the truth of

$$\langle \forall k ::\ k \backslash m \wedge k \backslash n \ \equiv \ k \backslash x \wedge k \backslash y \rangle$$

— which is trivially achieved by the assignment $x,y := m,n$ — and then, reduces x and y in such a way that the property is kept invariant whilst making progress to a state satisfying $x = y$. When such a state is reached, we have found a solution to the equation (4.2.12), and the value of x (or y since they are equal) is a solution of (4.2.6). (Note that this is the same technique we have used in the construction of the division algorithm.) Thus, the structure of the algorithm we are trying to develop is as follows:

$\{\ 0 < m \ \wedge \ 0 < n\ \}$

$x , y \ := \ m , n \ ;$

$\{$ **Invariant**: $\langle \forall k ::\ k \backslash m \wedge k \backslash n \ \equiv \ k \backslash x \wedge k \backslash y \rangle\ \}$

do $x \neq y \ \rightarrow \ x , y \ := \ A , B$

od

$\{\ x = y \ \wedge \ \langle \forall k ::\ k \backslash m \wedge k \backslash n \ \equiv \ k \backslash x \wedge k \backslash y \rangle\ \}$

Now we only have to define A and B in such a way that the assignment in the loop body leads to a state where $x = y$ is satisfied while maintaining the invariant. Exploiting the transitivity of equality, the invariant is maintained by choosing A and B so that

(4.2.13) $\langle \forall k ::\ k \backslash x \wedge k \backslash y \ \equiv \ k \backslash A \wedge k \backslash B \rangle$.

To ensure that we are making progress towards the termination condition, we have to define a *bound function* that depends on the assignments we choose for A and B.

At this point, we need to exploit properties specific to division. (Refer back to section 4.2.2 for a discussion of some of the properties.) Inspecting the shape of (4.2.13), we see that it is similar to the shape of property (4.2.5). This suggests that we can use (4.2.5), and in fact, considering this property, we have the corollary:

(4.2.14) $[\ k \backslash x \wedge k \backslash y \ \equiv \ k \backslash (x-y) \wedge k \backslash y \ ]$.

The relevance of this corollary is that our invariant is preserved by the assignment $x \ := \ x-y$ (leaving the value of y unchanged). (Compare (4.2.14) with (4.2.13).) Note that this also reduces the value of x when y is positive. This suggests that we strengthen the invariant by requiring that x and y remain positive; the assignment $x \ := \ x-y$ is executed when x is greater than y and, symmetrically, the assignment $y \ := \ y-x$ is executed when y is greater than x. As bound function we can take $x+y$. The algorithm becomes

$\{\ 0 < m \ \wedge \ 0 < n \ \}$

$x , y \ := \ m , n \ ;$

$\{$ **Invariant**: $0<x \ \wedge \ 0<y \ \wedge \ \langle \forall k ::\ k \backslash m \wedge k \backslash n \ \equiv \ k \backslash x \wedge k \backslash y \rangle$

Bound function: $x+y$ $\}$

do $x \neq y \ \rightarrow$

if $\ y < x \ \rightarrow \ x \ := \ x-y$

$\square \ \ x < y \ \rightarrow \ y \ := \ y-x$

fi

od

$\{\ 0<x \ \wedge \ 0<y \ \wedge \ x = y \ \wedge \ \langle \forall k ::\ k \backslash m \wedge k \backslash n \ \equiv \ k \backslash x \wedge k \backslash y \rangle \ \}$

(We leave the reader to perform the standard steps used to verify the correctness of the algorithm.) Finally, since

$(x < y \vee y < x) \ \equiv \ x \neq y \ \ ,$

we can safely remove the outer guard and simplify the algorithm, as shown below.

$$\{\ 0<m\ \wedge\ 0<n\ \}$$

$$x,y\ :=\ m,n\ ;$$

$$\{\ \textbf{Invariant}:\ \ 0{<}x\ \ \wedge\ \ 0{<}y\ \ \wedge\ \ \langle\forall k::\ k\backslash m\wedge k\backslash n\ \ \equiv\ \ k\backslash x\wedge k\backslash y\rangle$$

$$\quad\textbf{Bound function}:\ x{+}y\ \}$$

$$\mathsf{do}\ \ y<x\ \rightarrow\ x\ :=\ x{-}y$$

$$\Box\ \ x<y\ \rightarrow\ y\ :=\ y{-}x$$

$$\mathsf{od}$$

$$\{\ 0{<}x\ \ \wedge\ \ 0{<}y\ \ \wedge\ \ x=y\ \ \wedge\ \ \langle\forall k::\ k\backslash m\wedge k\backslash n\ \ \equiv\ \ k\backslash x\wedge k\backslash y\rangle\ \}$$

The algorithm that we have constructed is Euclid's algorithm for computing the greatest common divisor of two positive natural numbers, the oldest nontrivial algorithm that has survived to the present day! (Please note that our formulation of the algorithm differs from most versions found in number-theory books. While they use the property $[\ m\triangledown n\ =\ n\triangledown(m \bmod n)\ ]$, we use (4.2.14), i.e., $[\ m\triangledown n\ =\ (m{-}n)\triangledown n\ ]$. For an encyclopedic account of Euclid's algorithm, we recommend [Knu97, p. 334].)

4.2.4 Greatest common divisor

In section 4.2.2, we described the problem we were tackling as establishing that the infimum of two natural numbers under the division ordering always exists; it was only at the end of the section 4.2.3 that we announced that the algorithm we had derived is an algorithm for determining the greatest common divisor. This was done deliberately in order to avoid the confusion that can — and does — occur when using the words "greatest common divisor". In this section, we clarify the issue in some detail.

Confusion and ambiguity occur when a set can be ordered in two different ways. The natural numbers can be ordered by the usual size ordering (denoted by the symbol $\leq$), but they can also be ordered by the division relation. When the ordering is not made explicit (for instance, when referring to the "least" or "greatest" of a set of numbers), we might normally understand the size ordering, but the division ordering might be meant, depending on the context.

In words, the infimum of two values in a partial ordering — if it exists — is the largest value (with respect to the ordering) that is at most both values (with respect to the ordering). The terminology "greatest lower bound" is often used instead of "infimum".

Of course, "greatest" here is with respect to the partial ordering in question. Thus, the infimum (or greatest lower bound) of two numbers with respect to the division ordering — if it exists — is the largest number with respect to the division ordering that divides both of the numbers. Since, *for strictly positive numbers*, "largest with respect to the division ordering" implies "largest with respect to the size ordering" (equally, the division relation, restricted to strictly positive numbers, is a subset of the $\leq$ relation), the "largest number with respect to the *division* ordering that divides both of the numbers" is the same, *for strictly positive numbers*, as the "largest number with respect to the *size* ordering that divides both of the numbers". Both these expressions may thus be abbreviated to the "greatest common divisor" of the numbers, with no problems caused by the ambiguity in the meaning of "greatest" — *when the numbers are strictly positive*. Ambiguity does occur, however, when the number 0 is included, because 0 is the *largest* number with respect to the division ordering, but the *smallest* number with respect to the size ordering. If "greatest" is taken to mean with respect to the division ordering on numbers, the greatest common divisor of 0 and 0 is simply 0. If, however, "greatest" is taken to mean with respect to the size ordering, there is no greatest common divisor of 0 and 0. This would mean that the gcd operator is no longer idempotent, since $0 \bigtriangledown 0$ is undefined, and it is no longer associative, since, for positive m, $(m \bigtriangledown 0) \bigtriangledown 0$ is well-defined whilst $m \bigtriangledown (0 \bigtriangledown 0)$ is not.

Concrete evidence of the confusion in the standard mathematics literature is easy to find. We looked up the definition of greatest common divisor in three commonly used undergraduate mathematics texts, and found three non-equivalent definitions. The first [Hir95, p. 30] defines "greatest" to mean with respect to the divides relation (as, in our view, it should be defined); the second [Bur05, p. 21, def. 2.2] defines "greatest" to mean with respect to the $\leq$ relation (and requires that at least one of the numbers be non-zero). The third text [Fra98, p. 78] excludes zero altogether, defining the greatest common divisor of strictly positive numbers as the generator of all linear combinations of the given numbers; the accompanying explanation (in words) of the terminology replaces "greatest" by "largest" but does not clarify with respect to which ordering the "largest" is to be determined.

Now that we know that $\bigtriangledown$ is the greatest common divisor, we could change the operator to *gcd*, i.e., replace $m \bigtriangledown n$ by $m \; gcd \; n$. However, we stick to the "$\bigtriangledown$" notation because it makes the formulae shorter, and, so, easier to read. We also use "$\bigtriangleup$" to denote the least common multiple operator. To remember which is which, just remember that infima (*lower* bounds) are indicated by *downward*-pointing symbols (eg. $\downarrow$ for minimum, and $\vee$ for disjunction) and suprema (*upper* bounds) by *upward*-pointing symbols.

4.3 Euclid's algorithm as a verification interface

In this section we show how algorithms and the notion of invariance can be used to prove theorems. In particular, we show that the exploitation of Euclid's algorithm makes proofs related with the greatest common divisor simple and more systematic than the traditional ones.

There is a clear pattern in all our calculations: every time we need to prove a new theorem involving ∇, we construct an invariant that is valid initially (with $x, y := m, n$) and that corresponds to the theorem to be proved upon termination (with $x = y = m \nabla n$). Alternatively, we can construct an invariant that is valid on termination (with $x = y = m \nabla n$) and whose initial value corresponds to the theorem to be proved. The invariant in section 4.3.3 is such an example. Then, it remains to prove that the chosen invariant is valid after each iteration of the repeatable statement.

We start with a minor change in the invariant that allows us to prove some well-known properties. Then, we explore how the shape of the theorems to be proved determine the shape of the invariant. We also show how to prove a geometrical property of ∇.

4.3.1 Exploring the invariant

The invariant that we use in section 4.2.3 rests on the validity of the theorem

$$[\ k \backslash m \ \wedge\ k \backslash n \quad \equiv \quad k \backslash (m-n) \ \wedge\ k \backslash n\ ]\ .$$

But, as Van Gasteren observed in [vG90, Chapter 11], we can use the more general and equally valid theorem

$$[\ k \setminus (c \times m) \ \wedge\ k \setminus (c \times n) \quad \equiv \quad k \setminus (c \times (m-n)) \ \wedge\ k \setminus (c \times n)\ ]$$

to conclude that the following property is an invariant of Euclid's algorithm:

$$\langle \forall k, c ::\ k \setminus (c \times m) \ \wedge\ k \setminus (c \times n) \quad \equiv \quad k \setminus (c \times x) \ \wedge\ k \setminus (c \times y) \rangle\ .$$

In particular, the property is true on termination of the algorithm, at which point x and y both equal $m \nabla n$. That is, for all m and n, such that $0 < m$ and $0 < n$,

$$[\ k \setminus (c \times m) \ \wedge\ k \setminus (c \times n) \quad \equiv \quad k \setminus (c \times (m \nabla n))\ ]\ . \tag{4.3.1}$$

In addition, theorem (4.3.1) holds when $m < 0$, since

$$[\ (-m) \nabla n = m \nabla n\ ] \quad \wedge \quad [\ k \setminus (c \times (-m)) \quad \equiv \quad k \setminus (c \times m)\ ]\ ,$$

and it holds when m equals 0, since $[\ k\backslash 0\ ]$. Hence, using the symmetry between m and n we conclude that (4.3.1) is indeed valid for all integers m and n. (In Van Gasteren's presentation, this theorem only holds for all $(m,n) \neq (0,0)$.)

Theorem (4.3.1) can be used to prove a number of properties of the greatest common divisor. If, for instance, we replace k by m, we have

$$[\ m \setminus (c\times n) \ \equiv\ m \setminus (c \times (m\bigtriangledown n))\ ]\ ,$$

and, as a consequence, we also have

(4.3.2) $$[\ (m \setminus (c\times n) \ \equiv\ m\backslash c) \ \Leftarrow\ m\bigtriangledown n = 1\ ]\ .$$

More commonly, (4.3.2) is formulated as the weaker

$$[\ m\backslash c \ \Leftarrow\ m\bigtriangledown n = 1 \ \wedge\ m\backslash(c\times n)\ ]\ ,$$

and is known as Euclid's Lemma. Another significant property is

(4.3.3) $$[\ k \setminus (c \times (m\bigtriangledown n)) \ \equiv\ k \setminus ((c\times m)\bigtriangledown(c\times n))\ ]\ ,$$

which can be proved as:

$$\begin{array}{ll} & k \setminus (c \times (m\bigtriangledown n)) \\ = & \quad \{\quad (4.3.1)\ \} \\ & k \setminus (c\times m)\ \wedge\ k \setminus (c\times n) \\ = & \quad \{\quad (4.2.7)\ \} \\ & k \setminus ((c\times m)\bigtriangledown(c\times n))\ . \end{array}$$

From (4.3.3) we conclude

(4.3.4) $$[\ (c\times m)\bigtriangledown(c\times n) = c \times (m\bigtriangledown n)\ ]\ .$$

Property (4.3.4) states that multiplication by a natural number distributes over $\bigtriangledown$. It is an important property that can be used to simplify arguments where both multiplication and the greatest common divisor are involved. An example is Van Gasteren's proof of the theorem

(4.3.5) $$[\ (m\times p)\bigtriangledown n = m\bigtriangledown n \ \Leftarrow\ p\bigtriangledown n = 1\ ]\ ,$$

which is as follows:

$$
\begin{array}{ll}
 & m \nabla n \\
= & \quad \{ \quad p \nabla n = 1 \text{ and } 1 \text{ is the unit of multiplication} \quad \} \\
 & (m \times (p \nabla n)) \nabla n \\
= & \quad \{ \quad (4.3.4) \quad \} \\
 & (m \times p) \nabla (m \times n) \nabla n \\
= & \quad \{ \quad (m \times n) \nabla n = n \quad \} \\
 & (m \times p) \nabla n \ .
\end{array}
$$

4.3.2 ∇ on the left side

In the previous sections, we have derived a number of properties of the ∇ operator. However, where the divides relation is involved, the operator always occurs on the right side of the relation. (For examples, see (4.2.7) and (4.3.3).) Now we consider properties where the operator is on the left side of a divides relation. Our goal is to show that

(4.3.6) $[\ (m \nabla n) \setminus k \ \equiv \ \langle \exists a, b :: \ k \ = \ m \times a + n \times b \rangle \ ]$,

where the range of a and b is the integers.

Of course, if (4.3.6) is indeed true, then it is also true when k equals $m \nabla n$. That is, a consequence of (4.3.6) is

(4.3.7) $[\ \langle \exists a, b :: \ m \nabla n \ = \ m \times a + n \times b \rangle \ ]$.

In words, $m \nabla n$ is a linear combination of m and n. For example,

$$3 \nabla 5 \ = \ 1 \ = \ 3 \times 2 - 5 \times 1 \ = \ 5 \times 2 - 3 \times 3 \ .$$

Vice-versa, if (4.3.7) is indeed true then (4.3.6) is a consequence. (The crucial fact is that multiplication distributes through addition.) It thus suffices to prove (4.3.7).

We can establish (4.3.7) by constructing such a linear combination for given values of m and n.

When n is 0, we have

$$m \nabla 0 \ = \ m \ = \ m \times 1 + 0 \times 1 \ .$$

(The multiple of 0 is arbitrarily chosen to be 1.)

When both m and n are non-zero, we need to augment Euclid's algorithm with a computation of the coefficients. The most effective way to establish the property is to establish that x and y are linear combinations of m and n is an invariant of the algorithm; this is best expressed using matrix arithmetic.

In the algorithm below, the assignments to x and y have been replaced by equivalent assignments to the vector $(x\ y)$. Also, an additional variable **C**, whose value is a 2×2 matrix of integers has been introduced into the program. Specifically, **I**, **A** and **B** are 2×2 matrices; **I** is the identity matrix $\left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right)$, **A** is the matrix $\left(\begin{smallmatrix}1&0\\-1&1\end{smallmatrix}\right)$ and **B** is the matrix $\left(\begin{smallmatrix}1&-1\\0&1\end{smallmatrix}\right)$. (The assignment $(x\ y) := (x\ y)\times\mathbf{A}$ is equivalent to $x, y := x-y, y$, as can be easily checked.)

$\{\ 0<m\ \wedge\ 0<n\ \}$

$(x\ y), \mathbf{C}\ :=\ (m\ n), \mathbf{I}\ ;$

$\{$ **Invariant**: $(x\ y)\ =\ (m\ n)\times\mathbf{C}\ \}$

do $y<x\ \rightarrow\ (x\ y), \mathbf{C}\ :=\ (x\ y)\times\mathbf{A}\ ,\ \mathbf{C}\times\mathbf{A}$

$\square\ \ x<y\ \rightarrow\ (x\ y), \mathbf{C}\ :=\ (x\ y)\times\mathbf{B}\ ,\ \mathbf{C}\times\mathbf{B}$

od

$\{\ (x\ y)\ =\ (m\triangledown n\ \ m\triangledown n)\ =\ (m\ n)\times\mathbf{C}\ \}$

The invariant shows only the relation between the vectors $(x\ y)$ and $(m\ n)$; in words, $(x\ y)$ is a multiple of $(m\ n)$.

It is straightforward to verify that the invariant is established by the initialising assignment, and maintained by the loop body. Crucial to the proof that it is maintained by the loop body is that multiplication (here of matrices) is associative. Had we expressed the assignments to **C** in terms of its four elements, verifying that the invariant is maintained by the loop body would have amounted to giving in detail the proof that matrix multiplication is associative. This is a pointless duplication of effort, avoiding which fully justifies the excursion into matrix arithmetic.

(An exercise for the reader is to express the property that m and n are linear combinations of x and y. The solution involves observing that **A** and **B** are invertible. This will be exploited in section 4.4.2.)

This algorithm is commonly called Extended Euclid's Algorithm.

4.3.3 A geometrical property

In this section, we prove that in a Cartesian coordinate system, $m \triangledown n$ can be interpreted as the number of points with integral coordinates on the straight line joining the points $(0,0)$ and (m,n), excluding $(0,0)$. Formally, with dummies s and t ranging over integers, we prove:

(4.3.8) $[\ \langle \Sigma s,t \ :\ m\times t = n\times s \ \wedge\ s \le m \ \wedge\ t \le n \ \wedge\ (0<s \vee 0<t)\ :\ 1\rangle \ =\ m \triangledown n\ ]$.

We begin by observing that (4.3.8) holds when $m = 0$ or when $n = 0$ (we leave the proof to the reader). When $0 < m$ and $0 < n$, we can simplify the range of (4.3.8). First, we observe that

$$(0 < s \le m \ \equiv\ 0 < t \le n) \Leftarrow m\times t = n\times s \ ,$$

since

$$\begin{array}{ll} & 0 < t \le n \\ = & \{\ \ 0 < m\ \ \} \\ & 0 < m\times t \le m\times n \\ = & \{\ \ m\times t = n\times s\ \ \} \\ & 0 < n\times s \le m\times n \\ = & \{\ \ 0 < n, \text{cancellation}\ \ \} \\ & 0 < s \le m\ . \end{array}$$

As a result, (4.3.8) can be written as

(4.3.9) $[\ \langle \Sigma s,t \ :\ m\times t = n\times s \ \wedge\ 0 < t \le n \ :\ 1\rangle = m \triangledown n\ ]$.

In order to use Euclid's algorithm, we need to find an invariant that allows us to conclude (4.3.9). If we use as invariant

(4.3.10) $\langle \Sigma s,t \ :\ x\times t = y\times s \ \wedge\ 0 < t \le y \ :\ 1\rangle = x \triangledown y$,

its initial value is the property that we want to prove:

$$\langle \Sigma s,t \ :\ m\times t = n\times s \ \wedge\ 0 < t \le n \ :\ 1\rangle = m \triangledown n\ .$$

Its value upon termination is

$$\langle \Sigma s,t \ :\ (m \triangledown n)\times t = (m \triangledown n)\times s \ \wedge\ 0 < t \le m \triangledown n \ :\ 1\rangle = (m \triangledown n) \triangledown (m \triangledown n)\ ,$$

which is equivalent (by cancellation of multiplication and idempotence of $\triangledown$) to

$$\langle \Sigma s,t \ : \ t = s \ \wedge \ 0 < t \leq m \triangledown n \ : \ 1 \rangle = m \triangledown n \quad .$$

It is easy to see that the invariant reduces to true on termination (because the sum on the left equals $m \triangledown n$), making its initial value also true.

It is also easy to see that the right-hand side of the invariant is unnecessary as it is the same initially and on termination. This motivates the generalisation of the concept "invariant". "Invariants" in the literature are always Boolean-valued functions of the program variables, but we see no reason why "invariants" shouldn't be of any type: for us, an *invariant* of a loop is simply a function of the program variables whose value is unchanged by execution of the loop body[8]. In this case, the value is a natural number. Therefore, we can simplify (4.3.10) and use as invariant

$$\langle \Sigma s,t \ : \ x \times t = y \times s \ \wedge \ 0 < t \leq y \ : \ 1 \rangle \quad . \tag{4.3.11}$$

Its value on termination is

$$\langle \Sigma s,t \ : \ (m \triangledown n) \times t = (m \triangledown n) \times s \ \wedge \ 0 < t \leq m \triangledown n \ : \ 1 \rangle \quad ,$$

which is equivalent to

$$\langle \Sigma s,t \ : \ t = s \ \wedge \ 0 < t \leq m \triangledown n \ : \ 1 \rangle \quad .$$

As said above, this sum equals $m \triangledown n$.

Now, since the invariant (4.3.11) equals the left-hand side of (4.3.9) for the initial values of x and y, we only have to check if it remains constant after each iteration. This means that we have to prove (for $y < x \wedge 0 < y$):

$$\begin{aligned} & \langle \Sigma s,t \ : \ x \times t = y \times s \ \wedge \ 0 < t \leq y \ : \ 1 \rangle \\ = \ & \langle \Sigma s,t \ : \ (x - y) \times t = y \times s \ \wedge \ 0 < t \leq y \ : \ 1 \rangle \ , \end{aligned}$$

which can be rewritten, for positive x and y, as:

$$\begin{aligned} & \langle \Sigma s,t \ : \ (x + y) \times t = y \times s \ \wedge \ 0 < t \leq y \ : \ 1 \rangle \\ = \ & \langle \Sigma s,t \ : \ x \times t = y \times s \ \wedge \ 0 < t \leq y \ : \ 1 \rangle \ . \end{aligned}$$

[8] Some caution is needed here because our more general use of the word "invariant" does not completely coincide with its standard usage for Boolean-valued functions. The standard meaning of an invariant of a statement S is a Boolean-valued function of the program variables which, in the case that the function evaluates to true, remains true after execution of S. Our usage requires that, if the function evaluates to false before execution of S, it continues to evaluate to false after executing S.

The proof is as follows:

$$
\begin{array}{ll}
 & \langle \Sigma s,t \ : \ (x+y)\times t = y\times s \ \wedge \ 0<t\leq y \ : \ 1\rangle \\
= & \quad \{ \quad \text{distributivity and cancellation} \quad \} \\
 & \langle \Sigma s,t \ : \ x\times t = y\times (s-t) \ \wedge \ 0<t\leq y \ : \ 1\rangle \\
= & \quad \{ \quad \text{range translation: } s := s+t \quad \} \\
 & \langle \Sigma s,t \ : \ x\times t = y\times s \ \wedge \ 0<t\leq y \ : \ 1\rangle \ .
\end{array}
$$

Note that the simplification done in (4.3.9) allows us to apply the range translation rule in the last step without having to relate the range of variable s with the possible values for variable t.

4.4 Euclid's algorithm as a construction interface

In this section we show how to use Euclid's algorithm to derive new theorems related with the greatest common divisor. We start by calculating reasonable sufficient conditions for a natural-valued function to distribute over the greatest common divisor. We also derive an efficient algorithm for enumerating the positive rational numbers in two different ways.

4.4.1 Distributivity properties

In addition to multiplication by a natural number, there are other functions that distribute over ∇. The goal of this subsection is to determine sufficient conditions for a natural-valued function f to distribute over ∇, i.e., for the following property to hold:

(4.4.1) $[\ f.(m \nabla n) = f.m \nabla f.n \ ]$.

For simplicity, we restrict all variables to natural numbers. This implies that the domain of f is also restricted to the natural numbers.

We explore (4.4.1) by identifying invariants of Euclid's algorithm involving the function f. To determine an appropriate loop invariant, we take the right-hand side of (4.4.1) and calculate:

$$
\begin{array}{ll}
 & f.m \nabla f.n \\
= & \quad \{ \quad \text{the initial values of } x \text{ and } y \text{ are } m \text{ and } n, \text{ respectively} \quad \}
\end{array}
$$

$$
\begin{array}{ll}
& f.x \mathbin{\triangledown} f.y \\
= & \quad \{ \quad \text{suppose that } f.x \mathbin{\triangledown} f.y \text{ is invariant;} \\
& \qquad \text{on termination: } x = m \mathbin{\triangledown} n \ \wedge \ y = m \mathbin{\triangledown} n \quad \} \\
& f.(m \mathbin{\triangledown} n) \mathbin{\triangledown} f.(m \mathbin{\triangledown} n) \\
= & \quad \{ \quad \mathbin{\triangledown} \text{ is idempotent} \quad \} \\
& f.(m \mathbin{\triangledown} n) \ .
\end{array}
$$

Property (4.4.1) is thus established under the assumption that $f.x \mathbin{\triangledown} f.y$ is an invariant of the loop body. (Please note that this invariant is of the more general form introduced in section 4.3.3.)

The next step is to determine what condition on f guarantees that $f.x \mathbin{\triangledown} f.y$ is indeed invariant. Noting the symmetry in the loop body between x and y, the condition is easily calculated to be

$$[\ f.(x-y) \mathbin{\triangledown} f.y \ = \ f.x \mathbin{\triangledown} f.y \ \Leftarrow \ 0 < y < x \ ] \ .$$

Equivalently, by the rule of range translation ($x \ := \ x+y$), the condition can be written as

(4.4.2) $$[\ f.x \mathbin{\triangledown} f.y \ = \ f.(x+y) \mathbin{\triangledown} f.y \ \Leftarrow \ 0 < x \ \wedge \ 0 < y \ ] \ .$$

Formally, this means that

$$\text{“ } f \text{ distributes over } \mathbin{\triangledown} \text{ ”} \ \Leftarrow \ (4.4.2) \ .$$

Incidentally, the converse of this property is also valid:

$$(4.4.2) \ \Leftarrow \ \text{“ } f \text{ distributes over } \mathbin{\triangledown} \text{ ”} \ .$$

The simple calculation proceeds as follows:

$$
\begin{array}{ll}
& f.(x+y) \mathbin{\triangledown} f.y \\
= & \quad \{ \quad f \text{ distributes over } \mathbin{\triangledown} \quad \} \\
& f.((x+y) \mathbin{\triangledown} y) \\
= & \quad \{ \quad (4.2.11) \quad \} \\
& f.(x \mathbin{\triangledown} y) \\
= & \quad \{ \quad f \text{ distributes over } \mathbin{\triangledown} \quad \} \\
& f.x \mathbin{\triangledown} f.y \ .
\end{array}
$$

By mutual implication we conclude that

$$\text{“ } f \text{ distributes over } \triangledown \text{ ”} \quad \equiv \quad (4.4.2) \quad .$$

We have now reached a point where we can determine if a function distributes over $\triangledown$. However, since (4.4.2) still has two occurrences of $\triangledown$, we want to refine it into simpler properties. Towards that end we turn our attention to the condition

$$f.x \triangledown f.y = f.(x+y) \triangledown f.y \quad ,$$

and we explore simple ways of guaranteeing that it is everywhere true. For instance, it is immediately obvious that any function that distributes over addition distributes over $\triangledown$. (Note that multiplication by a natural number is such a function.) The proof is very simple:

$$\begin{array}{ll} & f.(x+y) \triangledown f.y \\ = & \quad \{ \quad f \text{ distributes over addition} \quad \} \\ & (f.x+f.y) \triangledown f.y \\ = & \quad \{ \quad (4.2.11) \quad \} \\ & f.x \triangledown f.y \quad . \end{array}$$

In view of properties (4.2.11) and (4.3.5), we formulate the following lemma, which is a more general requirement:

Lemma 4.4.3 All functions f that satisfy

$$\langle \forall x, y :: \quad \langle \exists a, b \; : \; a \triangledown f.y = 1 \; : \; f.(x+y) \; = \; a \times f.x \; + \; b \times f.y \rangle \rangle$$

distribute over $\triangledown$.

Proof

$$\begin{array}{ll} & f.(x+y) \triangledown f.y \\ = & \quad \{ \quad f.(x+y) \; = \; a \times f.x \; + \; b \times f.y \quad \} \\ & (a \times f.x \; + \; b \times f.y) \triangledown f.y \\ = & \quad \{ \quad (4.2.11) \quad \} \\ & (a \times f.x) \triangledown f.y \\ = & \quad \{ \quad a \triangledown f.y = 1 \quad \text{and} \quad (4.3.5) \quad \} \\ & f.x \triangledown f.y \quad . \end{array}$$

Note that since the discussion above is based on Euclid's algorithm, lemma 4.4.3 only applies to positive arguments. We now investigate the case where m or n is 0. We have, for $m = 0$:

$$
\begin{array}{ll}
& f.(0\triangledown n) \;=\; f.0 \triangledown f.n \\
= & \quad \{ \quad [\; 0\triangledown m = m \;] \quad \} \\
& f.n \;=\; f.0 \triangledown f.n \\
= & \quad \{ \quad [\; a\backslash b \;\equiv\; a = b\triangledown a \;] \quad \} \\
& f.n \setminus f.0 \\
\Leftarrow & \quad \{ \quad \text{obvious possibilities that make the expression valid} \\
& \qquad \text{are } f.0 = 0,\ f.n = 1, \text{ or } f.n = f.0\text{; the first is the} \\
& \qquad \text{interesting case} \quad \} \\
& f.0 = 0 \;\; .
\end{array}
$$

Hence, using the symmetry between m and n we have, for $m = 0$ or $n = 0$:

$$(4.4.4)\quad f.(m\triangledown n) \;=\; f.m \triangledown f.n \quad\Leftarrow\quad f.0 = 0 \;\wedge\; (m = 0 \vee n = 0) \quad .$$

The conclusion is that we can use (4.4.4) and lemma 4.4.3 to prove that a natural-valued function with domain $\mathbb{N}$ distributes over $\triangledown$. We were unable to prove that the condition in lemma 4.4.3 is necessary for a function to distribute over $\triangledown$, but we do not know any function distributing over $\triangledown$ that does not satisfy the condition.

Example 0: the Fibonacci function

In [Dij90], Edsger Dijkstra proves that the Fibonacci function distributes over $\triangledown$. He does not use lemma 4.4.3 explicitly, but he constructs the property

$$(4.4.5)\quad \mathsf{fib}.(x+y) \;=\; \mathsf{fib}.(y-1) \times \mathsf{fib}.x \;+\; \mathsf{fib}.(x+1) \times \mathsf{fib}.y \quad ,$$

and then, using the lemma

$$\mathsf{fib}.y \triangledown \mathsf{fib}.(y-1) \;=\; 1 \quad ,$$

he concludes the proof. His calculation is the same as that in the proof of lemma 4.4.3 but for particular values of a and b and with f replaced by fib. Incidentally, if we don't want to construct property (4.4.5) we can easily verify it using induction — more details are given in [GKP94].

An interesting application of this distributivity property is to prove that for any positive k, every kth number in the Fibonacci sequence is a multiple of the kth number in the Fibonacci sequence. More formally, the goal is to prove

$$\mathsf{fib}.(n\times k) \text{ is a multiple of } \mathsf{fib}.k \quad ,$$

for positive k and natural n. A concise proof is:

$$\begin{array}{ll} & \mathsf{fib}.(n\times k) \text{ is a multiple of } \mathsf{fib}.k \\ = & \quad \{ \quad \text{definition} \quad \} \\ & \mathsf{fib}.k \setminus \mathsf{fib}.(n\times k) \\ = & \quad \{ \quad [\ a\setminus b \ \equiv\ a\bigtriangledown b = a\ ]\,, \\ & \qquad \text{with } a := \mathsf{fib}.k \text{ and } b := \mathsf{fib}.(n\times k) \quad \} \\ & \mathsf{fib}.k \bigtriangledown \mathsf{fib}.(n\times k) \ =\ \mathsf{fib}.k \\ = & \quad \{ \quad \mathsf{fib} \text{ distributes over } \bigtriangledown \quad \} \\ & \mathsf{fib}.(k\bigtriangledown(n\times k)) = \mathsf{fib}.k \\ = & \quad \{ \quad k\bigtriangledown(n\times k) = k \text{ and reflexivity} \quad \} \\ & \mathsf{true} \ . \end{array}$$

Example 1: the Mersenne function

We now prove that, for all integers k and m such that $0 < k^m$, the function f defined as

$$f.m \ =\ k^m - 1$$

distributes over $\bigtriangledown$.

First, we observe that $f.0 = 0$. (Recall the discussion of (4.4.4).) Next, we use lemma 4.4.3. This means that we need to find integers a and b, such that

$$k^{m+n}-1 \ =\ a\times(k^m-1) + b\times(k^n-1) \quad \wedge \quad a\bigtriangledown(k^n-1) = 1 \quad .$$

The most obvious instantiations for a are 1, k^n and k^n-2. (That two consecutive numbers are coprime follows from (4.2.11).) Choosing $a = 1$, we calculate b:

$$\begin{array}{ll} & k^{m+n}-1 \ =\ (k^m-1) + b\times(k^n-1) \\ = & \quad \{ \quad \text{arithmetic} \quad \} \\ & k^{m+n}-k^m \ =\ b\times(k^n-1) \end{array}$$

$$
\begin{array}{ll}
= & \{ \quad \text{multiplication distributes over addition} \quad \} \\
& k^m \times (k^n - 1) \;=\; b \times (k^n - 1) \\
\Leftarrow & \{ \quad \text{Leibniz} \quad \} \\
& k^m = b \;\; .
\end{array}
$$

We thus have

$$k^{m+n} - 1 \;=\; 1 \times (k^m - 1) + k^m \times (k^n - 1) \;\;\wedge\;\; 1 \triangledown (k^n - 1) = 1 \quad ,$$

and we use lemma 4.4.3 to conclude that f distributes over $\triangledown$:

$$[\;\; (k^m - 1) \triangledown (k^n - 1) \;\;=\;\; k^{(m \triangledown n)} - 1 \;\;] \quad .$$

In particular, the Mersenne function, which maps m to $2^m - 1$, distributes over $\triangledown$:

$$[\;\; (2^m - 1) \triangledown (2^n - 1) \;\;=\;\; 2^{(m \triangledown n)} - 1 \;\;] \quad . \tag{4.4.6}$$

A corollary of (4.4.6) is the property

$$[\;\; (2^m - 1) \triangledown (2^n - 1) \;=\; 1 \;\;\equiv\;\; m \triangledown n = 1 \;\;] \quad .$$

In words, two numbers $2^m - 1$ and $2^n - 1$ are coprime is the same as exponents m and n are coprime. (See page 48 for an example where this property is used.)

4.4.2 Enumerating the rationals

A standard theorem of mathematics is that the rationals are "denumerable", i.e. they can be put in one-to-one correspondence with the natural numbers. Another way of saying this is that it is possible to enumerate the rationals so that each appears exactly once.

Recently, there has been a spate of interest in the construction of bijections between the natural numbers and the (positive) rationals (see [GLB06, KRSS03, CW00] and [AZ04, pp. 94–97]). Gibbons *et al* [GLB06] describe as "startling" the observation that the rationals can be efficiently enumerated[9] by "deforesting" the so-called "Calkin-Wilf" [CW00] tree of rationals. However, they claim that it is "not at all obvious" how to "deforest" the Stern-Brocot tree of rationals.

In this section, we derive an efficient algorithm for enumerating the rationals according to both orderings. The algorithm is based on a bijection between the rationals and

[9] By an *efficient enumeration* we mean a method of generating each rational without duplication with constant cost per rational in terms of arbitrary-precision simple arithmetic operations.

invertible 2×2 matrices. The key to the algorithm's derivation is the reformulation of Euclid's algorithm in terms of matrices (see section 4.3.2). The enumeration is efficient in the sense that it has the same time and space complexity as the algorithm credited to Moshe Newman in [KRSS03], albeit with a constant-fold increase in the number of variables and number of arithmetic operations needed at each iteration.

Note that, in our view, it is misleading to use the name "Calkin-Wilf tree of rationals" because Stern [Ste58] had already documented essentially the same structural characterisation of the rationals almost 150 years earlier than Calkin and Wilf. For more explanation, see the appendix (section 4.8) in which we review in some detail the relevant sections of Stern's paper. Stern attributes the structure to Eisenstein, so henceforth we refer to the "Eisenstein-Stern" tree of rationals where recent publications (including our own [BF08]) would refer to the "Calkin-Wilf tree of rationals". For a comprehensive account of properties of the Stern-Brocot tree, including further relationships with Euclid's algorithm, see [GKP94, pp. 116–118].

Euclid's algorithm

A positive rational in so-called "lowest form" is an ordered pair of positive, coprime integers. Every rational $\frac{m}{n}$ has unique lowest-form representation $\frac{m/(m\triangledown n)}{n/(m\triangledown n)}$. For example, $\frac{2}{3}$ is a rational in lowest form, whereas $\frac{4}{6}$ is the same rational, but not in lowest form.

Because computing the lowest-form representation involves computing greatest common divisors, it seems sensible to investigate Euclid's algorithm to see whether it gives insight into how to enumerate the rationals. Indeed it does.

Beginning with an arbitrary pair of positive integers m and n, the algorithm presented in section 4.3.2 calculates an invertible matrix $\mathbf{C}$ such that

$$(m\triangledown n \;\; m\triangledown n) \quad = \quad (m \;\; n) \times \mathbf{C} \;\; .$$

It follows that

$$(1 \;\; 1) \times \mathbf{C}^{-1} \quad = \quad (m/(m\triangledown n) \;\; n/(m\triangledown n)) \;\; . \tag{4.4.7}$$

Because the algorithm is deterministic, positive integers m and n uniquely define the matrix $\mathbf{C}$. That is, there is a function from pairs of positive integers to finite products of the matrices $\mathbf{A}$ and $\mathbf{B}$. Recall that $\mathbf{A}$ is the matrix $\begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}$ and $\mathbf{B}$ is the matrix $\begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}$.

Also, because the matrices $\mathbf{A}$ and $\mathbf{B}$ are constant and invertible, $\mathbf{C}^{-1}$ is a finite product of the matrices $\mathbf{A}^{-1}$ and $\mathbf{B}^{-1}$ and (4.4.7) uniquely defines a rational $\frac{m}{n}$ (in lowest form). We may therefore conclude that there is a bijection between the rationals and the finite

products of the matrices $\mathbf{A}^{-1}$ and $\mathbf{B}^{-1}$ provided that we can show that all such products are different.

The finite products of matrices $\mathbf{A}^{-1}$ and $\mathbf{B}^{-1}$ form a binary tree with root the identity matrix (the empty product). Renaming $\mathbf{A}^{-1}$ as $\mathbf{L}$ and $\mathbf{B}^{-1}$ as $\mathbf{R}$, the tree can be displayed with "L" indicating a left branch and "R" indicating a right branch. Figure 4.1 displays the first few levels of the tree.

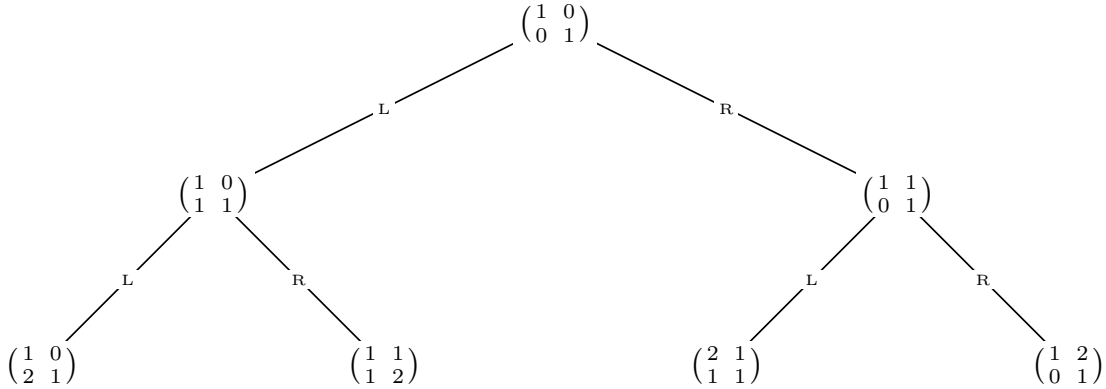


Figure 4.1: Tree of Products of **L** and **R**

That all matrices in the tree are different is proved by showing that the tree is a binary search tree (as formalised shortly). The key element of the proof[10] is that the determinants of **A** and **B** are both equal to 1 and, hence, the determinant of any finite product of **L**s and **R**s is also 1.

Formally, we define the relation $\prec$ on matrices that are finite products of **L**s and **R**s by

$$\begin{pmatrix}a&c\\b&d\end{pmatrix} \prec \begin{pmatrix}a'&c'\\b'&d'\end{pmatrix} \quad \equiv \quad \frac{a+c}{b+d} < \frac{a'+c'}{b'+d'} \ .$$

(Note that the denominator in these fractions is strictly positive; this fact is easily proved by induction.) We prove that, for all such matrices **X**, **Y** and **Z**,

(4.4.8) $\mathbf{X}\times\mathbf{L}\times\mathbf{Y} \ \prec \ \mathbf{X} \ \prec \ \mathbf{X}\times\mathbf{R}\times\mathbf{Z}$.

It immediately follows that there are no duplicates in the tree of matrices because the relation $\prec$ is clearly transitive and a subset of the inequality relation. (Property (4.4.8) formalises precisely what we mean by the tree of matrices forming a binary search tree: the entries are properly ordered by the relation $\prec$, with matrices in the left branch being "less than" the root matrix which is "less than" matrices in the right branch.)

[10] The proof is an adaptation of the proof in [GKP94, p. 117] that the rationals in the Stern-Brocot tree are all different. Our use of determinants corresponds to their use of "the fundamental fact" (4.31). Note that the definitions of **L** and **R** are swapped around in [GKP94].)

In order to show that

(4.4.9) $\mathbf{X}\times\mathbf{L}\times\mathbf{Y} \prec \mathbf{X}$,

suppose $\mathbf{X} = \left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right)$ and $\mathbf{Y} = \left(\begin{smallmatrix} a' & c' \\ b' & d' \end{smallmatrix}\right)$. Then, since $\mathbf{L} = \left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right)$, (4.4.9) is easily calculated to be

$$\frac{(a{+}c){\times}a' + (c{\times}b') + (a{+}c){\times}c' + (c{\times}d')}{(b{+}d){\times}a' + (d{\times}b') + (b{+}d){\times}c' + (d{\times}d')} \;<\; \frac{a{+}c}{b{+}d} \quad .$$

That this is true is also a simple, albeit longer, calculation (which exploits the cancellation properties of multiplication and addition); as observed earlier, the key property is that the determinant of $\mathbf{X}$ is 1, i.e. $a{\times}d - b{\times}c = 1$. The calculation is:

$\frac{(a+c)\times a' + (c\times b') + (a+c)\times c' + (c\times d')}{(b+d)\times a' + (d\times b') + (b+d)\times c' + (d\times d')} < \frac{a+c}{b+d}$

$=$ { arithmetic }

$\frac{(a+c)\times(a'+c') + c\times(b'+d')}{(b+d)\times(a'+c') + d\times(b'+d')} < \frac{a+c}{b+d}$

$=$ { denominators are different from zero }

$(a+c)\times(a'+c')\times(b+d) + c\times(b'+d')\times(b+d)$

$<$

$(a+c)\times(b+d)\times(a'+c') + (a+c)\times d\times(b'+d')$

$=$ { cancellation (twice) }

$c\times(b+d) < (a+c)\times d$

$=$ { distributivity and cancellation }

$c\times b < a\times d$

$=$ { $a\times d - b\times c = 1$ }

true .

The proof that $\mathbf{X} \prec \mathbf{X}\times\mathbf{R}\times\mathbf{Z}$ is similar.

Of course, we can also express Euclid's algorithm in terms of transpose matrices. Instead of writing assignments to the vector $(x\ y)$, we can write assignments to its transpose $\left(\begin{smallmatrix} x \\ y \end{smallmatrix}\right)$. Noting that $\mathbf{A}$ and $\mathbf{B}$ are each other's transposition, the assignment

$$(x\ y)\,,\mathbf{C} \quad := \quad (x\ y)\times\mathbf{A}\ ,\ \mathbf{C}{\times}\mathbf{A}$$

in the body of Euclid's algorithm becomes

$$\begin{pmatrix} x \\ y \end{pmatrix}\ ,\ \mathbf{C} \quad := \quad \mathbf{B}\times\begin{pmatrix} x \\ y \end{pmatrix}\ ,\ \mathbf{B}{\times}\mathbf{C}\ .$$

Similarly, the assignment

$$(x\ y)\,,\mathbf{C} \quad := \quad (x\ y)\times\mathbf{B}\ ,\ \mathbf{C}\times\mathbf{B}$$

becomes

$$\begin{pmatrix} x \\ y \end{pmatrix}\ ,\ \mathbf{C} \quad := \quad \mathbf{A}\times\begin{pmatrix} x \\ y \end{pmatrix}\ ,\ \mathbf{A}\times\mathbf{C}\ .$$

On termination, the matrix $\mathbf{C}$ computed by the revised algorithm will of course be different; the pair $\begin{pmatrix} m/(m\triangledown n) \\ n/(m\triangledown n) \end{pmatrix}$ is recovered from it by the identity

$$\mathbf{C}^{-1}\times\begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad = \quad \begin{pmatrix} m/(m\triangledown n) \\ n/(m\triangledown n) \end{pmatrix}\ .$$

In this way, we get a second bijection between the rationals and the finite products of the matrices $\mathbf{A}^{-1}$ and $\mathbf{B}^{-1}$. This is the basis for our second method of enumerating the rationals.

In summary, we have:

Theorem 4.4.10 Define the matrices $\mathbf{L}$ and $\mathbf{R}$ by

$$\mathbf{L} \ = \ \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{R} \ = \ \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}\ .$$

Then the following algorithm computes a bijection between the (positive) rationals and the finite products of $\mathbf{L}$ and $\mathbf{R}$. Specifically, the bijection is given by the function that maps the rational $\frac{m}{n}$ to the matrix $\mathbf{D}$ constructed by the algorithm together with the function from a finite product, $\mathbf{D}$, of $\mathbf{L}$s and $\mathbf{R}$s to $(1\ 1)\times\mathbf{D}$. (The comments added to the algorithm supply the information needed to verify this assertion.)

$\{\ 0<m\ \wedge\ 0<n\ \}$

$(x\ y)\,,\mathbf{D} \quad := \quad (m\ n)\,,\mathbf{I}\ ;$

$\{$ **Invariant**: $(m\ n) \quad = \quad (x\ y)\times\mathbf{D}\ \}$

do $y<x \ \rightarrow\ (x\ y)\,,\mathbf{D} \quad := \quad (x\ y)\times\mathbf{L}^{-1}\ ,\ \mathbf{L}\times\mathbf{D}$

$\Box\quad x<y \ \rightarrow\ (x\ y)\,,\mathbf{D} \quad := \quad (x\ y)\times\mathbf{R}^{-1}\ ,\ \mathbf{R}\times\mathbf{D}$

od

$\{\ (x\ y) \ = \ (m\triangledown n\ \ m\triangledown n) \quad \wedge \quad (\ m/(m\triangledown n)\ \ n/(m\triangledown n)\ ) \ = \ (1\ 1)\times\mathbf{D}\ \}$

Similarly, by applying the rules of matrix transposition to all expressions in the above, Euclid's algorithm constructs a second bijection between the rationals and finite products of the matrices $\mathbf{L}$ and $\mathbf{R}$. Specifically, the bijection is given by the function that maps the rational $\frac{m}{n}$ to the matrix $\mathbf{D}$ constructed by the revised algorithm together with the function from finite products, $\mathbf{D}$, of $\mathbf{L}$s and $\mathbf{R}$s to $\mathbf{D}\times\begin{pmatrix} 1 \\ 1 \end{pmatrix}$. $\Box$

Enumerating products of L and R

The problem of enumerating the rationals has been transformed to the problem of enumerating all finite products of the matrices **L** and **R**. As observed earlier, the matrices are naturally visualised as a tree —recall figure 4.1— with left branching corresponding to multiplying (on the right) by **L** and right branching to multiplying (on the right) by **R**.

By premultiplying each matrix in the tree by $(1\ 1)$, we get a tree of rationals. (Premultiplying by $(1\ 1)$ is accomplished by adding the elements in each column.) This tree is sometimes called the Calkin-Wilf tree [GLB06, AZ04, CW00]; we call it the *Eisenstein-Stern tree* of rationals. (See the appendix for an explanation.) The first four levels of the tree are shown in figure 4.2. In this figure, the vector $(x\ y)$ has been displayed as $\frac{y}{x}$. (Note the order of x and y. This is to aid comparison with existing literature.)

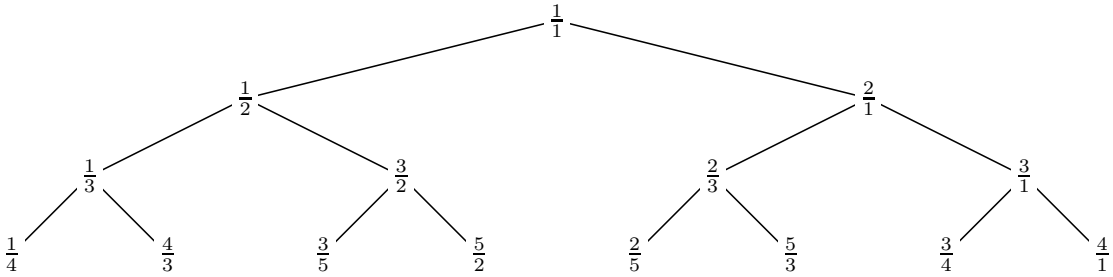


Figure 4.2: Eisenstein-Stern Tree of Rationals (aka Calkin-Wilf Tree)

By postmultiplying each matrix in the tree by $\begin{pmatrix}1\\1\end{pmatrix}$, we also get a tree of rationals. (Postmultiplying by $\begin{pmatrix}1\\1\end{pmatrix}$ is accomplished by adding the elements in each row.) This tree is called the Stern-Brocot tree [GKP94, pp. 116–118]. See figure 4.3. In this figure, the vector $\begin{pmatrix}x\\y\end{pmatrix}$ has been displayed as $\frac{x}{y}$.

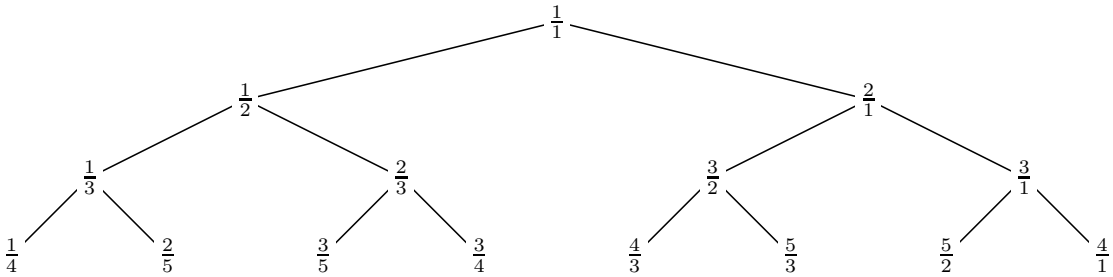


Figure 4.3: Stern-Brocot Tree of Rationals

Of course, if we can find an efficient way of enumerating the matrices in figure 4.1, we immediately get an enumeration of the rationals as displayed in the Eisenstein-Stern tree and as displayed in the Stern-Brocot tree — as each matrix is enumerated, simply premultiply by $(1\ 1)$ or postmultiply by $\begin{pmatrix}1\\1\end{pmatrix}$. Formally, the matrices are enumerated by

enumerating all strings of Ls and Rs in lexicographic order, beginning with the empty string; each string is mapped to a matrix by the homomorphism that maps "L" to **L**, "R" to **R**, and string concatenation to matrix product. It is easy to enumerate all such strings; as we see shortly, converting strings to matrices is also not difficult, for the simple reason that **L** and **R** are invertible.

The enumeration proceeds level-by-level. Beginning with the unit matrix (level 0), the matrices on each level are enumerated from left to right. There are 2^k matrices on level k, the first of which is $\mathbf{L}^k$. The problem is to determine for a given matrix, which is the matrix "adjacent" to it. That is, given a matrix **D**, which is a finite product of **L** and **R**, and is different from $\mathbf{R}^k$ for all k, what is the matrix that is to the immediate right of **D** in figure 4.1?

Consider the lexicographic ordering on strings of Ls and Rs of the same length. The string immediately following a string s (that is not the last) is found by identifying the rightmost L in s. Supposing s is the string $t\mathrm{LR}^j$, where R^j is a string of j Rs, its successor is $t\mathrm{RL}^j$.

It's now easy to see how to transform the matrix identified by s to its successor matrix. Simply postmultiply by $\mathbf{R}^{-j} \times \mathbf{L}^{-1} \times \mathbf{R} \times \mathbf{L}^j$. This is because, for all **T** and j,

$$(\mathbf{T} \times \mathbf{L} \times \mathbf{R}^j) \times (\mathbf{R}^{-j} \times \mathbf{L}^{-1} \times \mathbf{R} \times \mathbf{L}^j) \quad = \quad \mathbf{T} \times \mathbf{R} \times \mathbf{L}^j \ .$$

Also, it is easy to calculate $\mathbf{R}^{-j} \times \mathbf{L}^{-1} \times \mathbf{R} \times \mathbf{L}^j$. Specifically,

$$\mathbf{R}^{-j} \times \mathbf{L}^{-1} \times \mathbf{R} \times \mathbf{L}^j \quad = \quad \begin{pmatrix} 2j+1 & 1 \\ -1 & 0 \end{pmatrix} \ .$$

(We omit the details. Briefly, by induction, $\mathbf{L}^j$ equals $\left(\begin{smallmatrix} 1 & 0 \\ j & 1 \end{smallmatrix}\right)$. Also, **R** is the transpose of **L**.)

The final task is to determine, given a matrix **D**, which is a finite product of **L**s and **R**s, and is different from $\mathbf{R}^k$ for all k, the unique value j such that $\mathbf{D} = \mathbf{T} \times \mathbf{L} \times \mathbf{R}^j$ for some **T**. This can be determined by examining Euclid's algorithm once more.

The matrix form of Euclid's algorithm discussed in theorem 4.4.10 computes a matrix **D** given a pair of positive numbers m and n; it maintains the invariant

$$(m \ n) \quad = \quad (x \ y) \times \mathbf{D} \ .$$

D is initially the identity matrix and x and y are initialised to m and n, respectively; immediately following the initialisation process, **D** is repeatedly premultiplied by **R** so long as x is less than y. Simultaneously, y is reduced by x. The number of times that **D** is premultiplied by **R** is thus the greatest number j such that $j \times m$ is less than n, which

is $\lfloor \frac{n-1}{m} \rfloor$. Now suppose the input values m and n are coprime. Then, on termination of the algorithm, $(1\ 1) \times \mathbf{D}$ equals $(\ m\ \ n\ )$. That is, if

$$\mathbf{D} \ = \ \begin{pmatrix} \mathbf{D}_{00} & \mathbf{D}_{01} \\ \mathbf{D}_{10} & \mathbf{D}_{11} \end{pmatrix} \ ,$$

then,

$$\left\lfloor \frac{n-1}{m} \right\rfloor \ = \ \left\lfloor \frac{\mathbf{D}_{01} + \mathbf{D}_{11} - 1}{\mathbf{D}_{00} + \mathbf{D}_{10}} \right\rfloor \ .$$

It remains to decide how to keep track of the levels in the tree. For this purpose, it is not necessary to maintain a counter. It suffices to observe that $\mathbf{D}$ is a power of $\mathbf{R}$ exactly when the rationals in the Eisenstein-Stern, or Stern-Brocot, tree are integers, and this integer is the number of the next level in the tree (where the root is on level 0). So, it is easy to test whether the last matrix on the current level has been reached. Equally, the first matrix on the next level is easily calculated. For reasons we discuss in the next section, we choose to test whether the rational in the Eisenstein-Stern tree is an integer; that is, we evaluate the Boolean $\mathbf{D}_{00} + \mathbf{D}_{10} = 1$. In this way, we get the following (non-terminating) program which computes the successive values of $\mathbf{D}$.

$\mathbf{D} := \mathbf{I}$;
do $\mathbf{D}_{00} + \mathbf{D}_{10} = 1 \ \rightarrow \ \mathbf{D} := \begin{pmatrix} 1 & 0 \\ \mathbf{D}_{01}+\mathbf{D}_{11} & 1 \end{pmatrix}$
□ $\mathbf{D}_{00} + \mathbf{D}_{10} \neq 1 \ \rightarrow \ j := \left\lfloor \frac{\mathbf{D}_{01} + \mathbf{D}_{11} - 1}{\mathbf{D}_{00} + \mathbf{D}_{10}} \right\rfloor$;
$\mathbf{D} := \mathbf{D} \times \begin{pmatrix} 2j+1 & 1 \\ -1 & 0 \end{pmatrix}$
od

A minor simplification of this algorithm is that the "$-\,1$" in the assignment to j can be omitted. This is because $\lfloor \frac{n-1}{m} \rfloor$ and $\lfloor \frac{n}{m} \rfloor$ are equal when m and n are coprime and m is different from 1. We return to this shortly.

The enumerations

As remarked earlier, we immediately get an enumeration of the rationals as displayed in the Eisenstein-Stern tree and as displayed in the Stern-Brocot tree — as each matrix is enumerated, simply premultiply by $(1\ 1)$ or postmultiply by $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$, respectively.

In the case of enumerating the Eisenstein-Stern tree, several optimisations are possible. First, it is immediate from our derivation that the value assigned to the local variable j is a function of $(1\ 1) \times \mathbf{D}$. In turn, the matrix $\begin{pmatrix} 2j+1 & 1 \\ -1 & 0 \end{pmatrix}$ is also a function of $(1\ 1) \times \mathbf{D}$.

Let us name the function J, so that the assignment becomes

$$\mathbf{D} \;:=\; \mathbf{D} \times J.((1\ 1) \times \mathbf{D}) \ .$$

Then, the Eisenstein-Stern enumeration iteratively evaluates

$$(1\ 1) \times (\mathbf{D} \times J.((1\ 1) \times \mathbf{D})) \ .$$

Matrix multiplication is associative; so this is

$$((1\ 1) \times \mathbf{D}) \times J.((1\ 1) \times \mathbf{D}) \ ,$$

which is also a function of $(1\ 1) \times \mathbf{D}$. Moreover —in anticipation of the current discussion— we have been careful to ensure that the test for a change in the level in the tree is also a function of $(1\ 1) \times \mathbf{D}$. Combined together, this means that, in order to enumerate the rationals in Eisenstein-Stern order, it is not necessary to compute $\mathbf{D}$ at each iteration, but only $(1\ 1) \times \mathbf{D}$. Naming the two components of this vector m and n, and simplifying the matrix multiplications, we get[11]

$$\begin{array}{lllll} m,n & := & 1,1 \ ; & & \\ \mathsf{do} & m=1 & \rightarrow & m,n & := & n+1\,,m \\ \Box & m\neq 1 & \rightarrow & m,n & := & (2\left\lfloor \dfrac{n-1}{m} \right\rfloor + 1) \times m - n \ , \ m \\ \mathsf{od} & & & & & \end{array}$$

At this point, a further simplification is also possible. We remarked earlier that $\lfloor \frac{n-1}{m} \rfloor$ equals $\lfloor \frac{n}{m} \rfloor$ when m and n are coprime and m is different from 1. By good fortune, it is also the case that $(2 \lfloor \frac{n}{m} \rfloor + 1) \times m - n$ simplifies to $n+1$ when m is equal to 1. That is, the elimination of "-1" in the evaluation of the floor function leads to the elimination of the entire case analysis! This is the algorithm attributed to Newman in [KRSS03].

$$\begin{array}{lllll} m,n & := & 1,1 \ ; & & \\ \mathsf{do} & m,n & := & (2\left\lfloor \dfrac{n}{m} \right\rfloor + 1) \times m - n \ , \ m \\ \mathsf{od} & & & & \end{array}$$

Discussion

Our construction of an algorithm for enumerating the rationals in Stern-Brocot order was motivated by reading two publications, [GKP94, pp. 116–118] and [GLB06]. Gib-

[11] Recall that, to comply with existing literature, the enumerated rational is $\frac{n}{m}$ and not $\frac{m}{n}$.

bons, Lester and Bird [GLB06] show how to enumerate the elements of the Eisenstein-Stern tree, but claim that "it is not at all obvious how to do this for the Stern-Brocot tree". Specifically, they say[12]:

> However, there is an even better compensation for the loss of the ordering property in moving from the Stern-Brocot to the Calkin-Wilf tree: it becomes possible to deforest the tree altogether, and generate the rationals directly, maintaining no additional state beyond the 'current' rational. This startling observation is due to Moshe Newman (Newman, 2003). In contrast, it is not at all obvious how to do this for the Stern-Brocot tree; the best we can do seems to be to deforest the tree as far as its levels, but this still entails additional state of increasing size.

In this section, we have shown that it is possible to enumerate the rationals in Stern-Brocot order without incurring "additional state of increasing size". More importantly, we have presented *one* enumeration algorithm with *two* specialisations, one being the "Calkin-Wilf" enumeration they present, and the other being the Stern-Brocot enumeration that they described as being "not at all obvious".

The optimisation of Eisenstein-Stern enumeration which leads to Newman's algorithm is not possible for Stern-Brocot enumeration. Nevertheless, the complexity of Stern-Brocot enumeration is the same as the complexity of Newman's algorithm, both in time and space. The only disadvantage of Stern-Brocot enumeration is that four variables are needed in place of two; the advantage is the (well-known) advantage of the Stern-Brocot tree over the Eisenstein-Stern tree — the rationals on a given level are in ascending order.

Gibbons, Lester and Bird's goal seems to have been to show how the functional programming language Haskell implements the various constructions – the construction of the tree structures and Newman's algorithm. In doing so, they repeat the existing mathematical presentations of the algorithms as given in [GKP94, CW00, KRSS03]. The ingredients for an efficient enumeration of the Stern-Brocot tree are all present in these publications, but the recipe is missing!

The fact that expressing the rationals in "lowest form" is essential to the avoidance of duplication in any enumeration immediately suggests the relevance of Euclid's algorithm. The key to our exposition is that Euclid's algorithm can be expressed in terms of matrix multiplications, where —significantly— the underlying matrices are invertible. Transposition and inversion of the matrices capture the symmetry properties in a pre-

[12] Recall that they attribute the tree to Calkin and Wilf rather than Eisenstein and Stern.

cise, calculational framework. As a result, the bijection between the rationals and the tree elements is immediate and we do not need to give separate, inductive proofs for both tree structures. Also, the determination of the next element in an enumeration of the tree elements has been reduced to one unifying construction.

4.5 The theory of congruences

This section shows a calculational approach to the theory of congruences, an elegant theory that can be used to solve problems on divisibility. The theory was introduced by Carl Friedrich Gauss in his seminal book "Disquisitiones Arithmeticae" [Gau01], in 1801. In section I of his book, he gave the following definition:

> If a number a divides the difference of the numbers b and c, b and c are said to be congruent relative to a; if not, b and c are noncongruent. The number a is called the modulus. If the numbers b and c are congruent, each of them is called a residue of the other. If they are noncongruent they are called nonresidues.

He also wrote in a footnote that "the modulus must obviously be taken absolutely, i.e. without sign". Putting all this into a definition, we have Definition 4.5.1.

Definition 4.5.1 Let n be a natural number. We say that two integers a and b are congruent modulo n, and we write

$$a \cong b \pmod{n} \quad ,$$

when n divides the difference $a-b$. Formally, we write

$$a \cong b \pmod{n} \equiv n \backslash (a-b) \quad .$$

When n does not divide $a-b$ we say that a is noncongruent to b modulo n and we write $a \ncong b \pmod{n}$.

□

Our notation is slightly different from the conventional one, which was originally created by Gauss. Where we write $a \cong b \pmod{n}$, Gauss would have written $a \equiv b \ (mod\ n)$. He justified his choice in a footnote:

> We have adopted this symbol because of the analogy between equality and congruence. For the same reason Legendre, in the treatise which we shall

> often have occasion to cite, used the same sign for equality and congruence. To avoid ambiguity we have made a distinction.

Since we are using $\equiv$ to denote Boolean equality, Gauss's argument can also be applied to our case. Therefore, we have chosen the symbol $\cong$ because it still reflects the analogy with equality.

4.5.1 Basic properties of congruences

The first three properties that we show follow directly from the definition 4.5.1 by considering particular values for the modulus n.

Theorem 4.5.2

1. $[\ a \cong b \pmod 0 \equiv a = b\ ]$
2. $[\ a \cong b \pmod 1\ ]$
3. Two integers are congruent modulo 2 when they have the same parity, i.e. when they are both even or both odd. More formally,

$$[\ a \cong b \pmod 2 \equiv even.a \equiv even.b\ ] \quad .$$

Proof

1. $a \cong b \pmod 0$

$=$ { definition }

$0 \backslash (a-b)$

$=$ { 0 only divides 0 }

$a - b = 0$

$=$ { cancellation }

$a = b$.

2. $a \cong b \pmod 1$

$=$ { definition }

$1 \backslash (a-b)$

$=$ { $[1 \backslash n]$ }

`true` .

3. The following proof is based on the definition of the predicate *even* and on its distributivity properties:

$$
\begin{array}{ll}
 & a \cong b \pmod 2 \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & 2 \backslash (a - b) \\
= & \quad \{ \quad \text{definition of } even \quad \} \\
 & even.(a - b) \\
= & \quad \{ \quad \text{even distributes over addition} \quad \} \\
 & even.a \equiv even.b \quad .
\end{array}
$$

□

(In some number theory books the modulus is considered to be a positive natural number [Bur05], but we see no reason for excluding the case when the modulus is 0.)

Now, given an integer a, let $a \div n$ and $a \bmod n$ be, respectively, its quotient and remainder[13] upon division by n, so that

$$a = (a \div n) \times n + (a \bmod n) \quad ,$$

with

$$0 \leq a \bmod n < n \quad .$$

Then we can conclude that

$$a \cong (a \bmod n) \pmod n \quad ,$$

as the following calculation shows:

$$
\begin{array}{ll}
 & a \cong (a \bmod n) \pmod n \\
= & \quad \{ \quad \text{definition and } a \bmod n = a - (a \div n) \times n \quad \} \\
 & n \backslash (a - a + (a \div n) \times n) \\
= & \quad \{ \quad \text{arithmetic and } [\, n \backslash (k \times n) \,] \quad \} \\
 & \mathsf{true} \quad .
\end{array}
$$

[13] To be consistent with the existing literature, we overload the name "mod". We write $a \bmod n$ to denote the remainder upon division by n and we write $a \cong b \pmod n$ to denote that a and b are congruent modulo n. To help distinguish them, we write the first in a sans-serif font.

Remark In the book [Bur05, p. 64], this last property is presented as:

> Given an integer a, let q and r be its quotient and remainder upon division by n, so that
>
> $$a = q \times n + r \qquad 0 \leq r < n$$
>
> Then, by definition of congruence, $a \equiv r \pmod{n}$.

This is not totally correct, because the conclusion does not follow directly from the definition of congruence — we also use the property that a number n divides any multiple of n.

We have the feeling that most mathematicians would argue that this remark is so obvious that there is no need to include it. Nevertheless, the author is omitting a relevant property; one possible reason is the way in which he records his proofs. Had he used our calculational format and he would be forced to use the theorem $[\, n \backslash (k \times n) \,]$ to derive true in the last step.
(End of Remark)

Because in the integer division by n the remainder r satisfies the condition $0 \leq r < n$, any integer is congruent modulo n to exactly one natural less than n. In particular;

$$[\; a \cong 0 \pmod{n} \equiv n \backslash a \;] \quad ,$$

which follows directly from the definition of congruence. Usually, the set of n integers $\{r : 0 \leq r < n : r\}$ is called the set of least nonnegative residues modulo n.

The following theorem provides a useful characterisation of congruence modulo n in terms of remainder upon division by n.

Theorem 4.5.3 $\quad a \cong b \pmod{n} \quad \equiv \quad a \bmod n = b \bmod n$

Proof

$$\begin{array}{ll} & a \bmod n = b \bmod n \\ = & \quad \{ \quad [\; p \bmod q \;=\; p - (p \div q) \times q \;] \quad \} \\ & a - (a \div n) \times n \;=\; b - (b \div n) \times n \\ = & \quad \{ \quad \text{cancellation} \quad \} \\ & a - b = ((a \div n) - (b \div n)) \times n \end{array}$$

$$
\begin{array}{ll}
= & \{ \quad \text{division} \quad \} \\
& n \backslash (a-b) \\
= & \{ \quad \text{definition} \quad \} \\
& a \cong b \pmod{n} \ .
\end{array}
$$

□

We shall use this alternative definition whenever it is more convenient. The following three properties are an example. As we said before, there is an analogy between congruence and equality, because some elementary properties of equality carry over to congruences. For instance, the following theorem states that congruence is an equivalence relation, i.e. it is reflexive, symmetric, and transitive.

Theorem 4.5.4

1. Reflexivity: $[\, a \cong a \pmod{n} \,]$
2. Symmetry: $[\, a \cong b \pmod{n} \equiv b \cong a \pmod{n} \,]$
3. Transitivity: $[\ a \cong c \pmod{n} \ \Leftarrow \ a \cong b \pmod{n} \ \wedge \ b \cong c \pmod{n} \ ]$

Proof

1.
$$
\begin{array}{ll}
& a \cong a \pmod{n} \\
= & \{ \quad \text{theorem 4.5.3} \quad \} \\
& a \bmod n = a \bmod n \\
= & \{ \quad \text{reflexivity of equality} \quad \} \\
& \mathsf{true} \ .
\end{array}
$$

2.
$$
\begin{array}{ll}
& a \cong b \pmod{n} \\
= & \{ \quad \text{theorem 4.5.3} \quad \} \\
& a \bmod n = b \bmod n \\
= & \{ \quad \text{symmetry of equality} \quad \} \\
& b \bmod n = a \bmod n \\
= & \{ \quad \text{theorem 4.5.3} \quad \} \\
& b \cong a \pmod{n} \ .
\end{array}
$$

3. $a \cong b \pmod{n} \;\wedge\; b \cong c \pmod{n}$

$=$ { theorem 4.5.3 }

$a \bmod n = b \bmod n \;\wedge\; b \bmod n = c \bmod n$

$\Rightarrow$ { transitivity of equality }

$a \bmod n = c \bmod n$

$=$ { theorem 4.5.3 }

$a \cong c \pmod{n}$.

□

The three properties presented in the previous theorem could be proved using the definition of congruence and the properties of division, but the proofs would become slightly more complicated.

Note that since transitivity holds, we can write *continued congruences* to denote that all the numbers involved are congruent. For example, we may write expressions like the following, whenever it is convenient:

$$32 \cong -9 \cong 73 \cong 114 \pmod{41} \quad .$$

Also, whenever the modulus is clear, we may use the calculational proof format. For example, if the modulus is 41, we may write:

$$\begin{array}{ll} & 32 \\ \cong & \{ \quad 32 \cong -9 \quad \} \\ & -9 \\ \cong & \{ \quad -9 \cong 114 \quad \} \\ & 114 \quad . \end{array}$$

Now, another important property related with equality is the so-called Leibniz rule:

$$[\; f.a = f.b \Leftarrow a = b \;] \quad .$$

The correspondent rule in congruences would be

(4.5.5) $[\; f.a \cong f.b \pmod{n} \Leftarrow a \cong b \pmod{n} \;] \quad ,$

but this is not true in general. If, for instance, we define $exp.n$ to be the number of times that 2 divides n, then we have

$$4 \cong 2 \pmod{2} \quad ,$$

but

$$exp.4 \not\cong exp.2 \pmod 2 \quad .$$

We can, however, calculate a condition on f under which (4.5.5) holds:

$$\begin{array}{ll} & f.a \cong f.b \pmod n \\ = & \quad \{ \quad \text{definition} \quad \} \\ & n \backslash (f.a - f.b) \\ = & \quad \{ \quad \bullet \quad f.a - f.b = k \times (a-b) \text{ , for some } k \quad \} \\ & n \backslash (k \times (a-b)) \\ = & \quad \{ \quad a \cong b \pmod n \quad \} \\ & \texttt{true} \ . \end{array}$$

So, if a function f satisfies

(4.5.6) $\langle \forall a, b :: \langle \exists k :: \ f.a - f.b = k \times (a-b) \rangle \rangle$,

then (4.5.5) holds. The following theorem shows two examples of functions that satisfy (4.5.5).

Theorem 4.5.7

1. $[\, a+c \cong b+c \pmod n \ \Leftarrow \ a \cong b \pmod n \,]$
2. $[\, a \times c \cong b \times c \pmod n \ \Leftarrow \ a \cong b \pmod n \,]$

Proof

1. The function involved in this property is $f.n = n+c$. Since $[\, f.a - f.b = a-b \,]$, (4.5.6) holds, and consequently, (4.5.5) holds.
2. The function involved in this property is $f.n = n \times c$. Since $[\, f.a - f.b = c \times (a-b) \,]$, (4.5.6) holds, and consequently, (4.5.5) holds.

□

The following four properties are more flexible and complement the analogy with equality:

Theorem 4.5.8

1. $[\ a{+}c \cong b{+}d \pmod{n} \ \Leftarrow\ a{\cong}b \pmod{n} \wedge c{\cong}d \pmod{n}\ ]$
2. $[\ a{\times}c \cong b{\times}d \pmod{n} \ \Leftarrow\ a{\cong}b \pmod{n} \wedge c{\cong}d \pmod{n}\ ]$
3. $[\, a^k{\cong}b^k \pmod{n} \ \Leftarrow\ a{\cong}b \pmod{n} \wedge 0 < k\,]$
4. If $P.x = \langle \Sigma k : 0 \le k < m : c_k \times x^k \rangle$ then

$$[\, P.a \ \cong\ P.b \pmod{n} \ \Leftarrow\ a \cong b \pmod{n}\,] \quad .$$

Proof

1. $a{+}c \cong b{+}d \pmod{n}$

$=$ { definition }

$n \backslash ((a{+}c){-}(b{+}d))$

$=$ { associativity and symmetry }

$n \backslash ((a{-}b){+}(c{-}d))$

$=$ { context: $a{\cong}b \pmod{n}$ and $c{\cong}d \pmod{n}$, which correspond to $n\backslash(a{-}b)$ and $n\backslash(c{-}d)$, respectively }

`true` .

2. We first observe that if we have $a{\cong}b \pmod{n}$ and $c{\cong}d \pmod{n}$, then there exist integers i and j such that

$$a{-}b = i{\times}n \quad \text{, and}$$

$$c{-}d = j{\times}n \quad .$$

It follows that

$$a{\times}c = (b + i{\times}n){\times}(d + j{\times}n) \quad ,$$

and the proof is:

$a{\times}c \cong b{\times}d \pmod{n}$

$=$ { definition }

$n \backslash (a{\times}c - b{\times}d)$

$=$ { $a{\times}c = (b + i{\times}n){\times}(d + j{\times}n)$ and distributivity }

$$
\begin{array}{ll}
 & n \backslash (b \times d + n \times (i \times d + b \times j + i \times j \times n) - b \times d) \\
= & \quad \{ \quad \text{arithmetic} \quad \} \\
 & n \backslash (n \times (i \times d + b \times j + i \times j \times n)) \\
= & \quad \{ \quad [\, n \backslash (k \times n) \,] \quad \} \\
 & \texttt{true} \ .
\end{array}
$$

3. We can prove it by induction on k. If $k = 1$, it is clearly true. For $k > 1$, we assume the theorem is true (it is our induction hypothesis) and we prove the following:

$$a^{k+1} \cong b^{k+1} \pmod{n} \Leftarrow a \cong b \pmod{n} \quad .$$

The proof is:

$$
\begin{array}{ll}
 & a^{k+1} \cong b^{k+1} \\
= & \quad \{ \quad \text{arithmetic} \quad \} \\
 & a \times a^k \cong b \times b^k \\
\Leftarrow & \quad \{ \quad [\ a \times c \cong b \times d \pmod{n} \Leftarrow a \cong b \pmod{n} \wedge c \cong d \pmod{n} \ ] \quad \} \\
 & a \cong b \wedge a^k \cong b^k \\
= & \quad \{ \quad \text{assume } a \cong b \text{ and use the induction hypothesis} \quad \} \\
 & \texttt{true} \ .
\end{array}
$$

Therefore, we establish the theorem for all positive k.

4.
$$
\begin{array}{ll}
 & P.a \cong P.b \pmod{n} \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & \langle \Sigma k : 0 \le k < m : c_k \times a^k \rangle \cong \langle \Sigma k : 0 \le k < m : c_k \times b^k \rangle \pmod{n} \\
\Leftarrow & \quad \{ \quad \text{general form of} \\
 & \qquad [\ a + c \cong b + d \pmod{n} \Leftarrow a \cong b \pmod{n} \wedge c \cong d \pmod{n} \ ] \quad \} \\
 & \langle \forall k : 0 \le k < m : c_k \times a^k \cong c_k \times b^k \pmod{n} \rangle \\
\Leftarrow & \quad \{ \quad [\, a \times c \cong b \times c \pmod{n} \Leftarrow a \cong b \pmod{n} \,] \text{ and monotonicity} \quad \} \\
 & \langle \forall k : 0 \le k < m : a^k \cong b^k \pmod{n} \rangle \\
\Leftarrow & \quad \{ \quad [\, a^k \cong b^k \pmod{n} \Leftarrow a \cong b \pmod{n} \wedge 0 < k \,] \quad \} \\
 & a \cong b \pmod{n} \ .
\end{array}
$$

□

Congruences in practice The properties we have seen so far can be used to help us with certain computations. For example, suppose that we want to prove that 41 divides $2^{20}-1$. We can do it as follows (all congruences are modulo 41):

$$
\begin{array}{ll}
 & 2^{20}-1 \\
= & \quad \{ \quad 2^{20} = (2^5)^4 \quad \} \\
 & (2^5)^4-1 \\
\cong & \quad \{ \quad 2^5 = 32 \text{ and } 32 \cong -9 \quad \} \\
 & (-9)^4-1 \\
= & \quad \{ \quad (-9)^4 = 81^2 \quad \} \\
 & 81^2-1 \\
\cong & \quad \{ \quad 81 \cong -1 \text{ and } (-1)^2 = 1 \quad \} \\
 & 0 \ .
\end{array}
$$

Thus, $41 \setminus 2^{20}-1$. Note how the proof format easily allows us to use equality and congruence without ambiguity. For another example, suppose that we are asked to compute the remainder obtained upon dividing the sum

$$1!+2!+3!+4!+\cdots+99!+100!$$

by 12. A crucial observation is that $4! = 24 \cong 0 \pmod{12}$. Therefore, for $4 \leq k$, we have:

$$k! \ \cong \ 4!\times 5\times 6\times \cdots \times k \cong 0\times 5\times 6\times \cdots \times k \ \cong \ 0 \pmod{12} \quad .$$

Therefore,

$$
\begin{array}{ll}
 & 1!+2!+3!+4!+\cdots+99!+100! \\
\cong & \quad \{ \quad \text{observation above} \quad \} \\
 & 1!+2!+3!+0+\cdots+0+0 \\
= & \quad \{ \quad \text{arithmetic} \quad \} \\
 & 9 \ .
\end{array}
$$

Finally, we can use property 4 of theorem 4.5.8 to prove the known *rule of thumb* that a number n (written in base 10) is divisible by 3 when the sum of its digits is divisible

by 3. Assume that n has m digits, and that c_k is the k^{th} decimal digit of n (from right). Then, if

$$P.x = \langle \Sigma k : 0 \leq k < m : c_k \times x^k \rangle \quad ,$$

we have

$$n = P.10 \quad .$$

Moreover, assuming that all congruences are modulo 3, we have:

$$\begin{array}{ll} & P.10 \\ \cong & \{ \quad 10 \cong 1 \text{ (mod 3) and property 4 of theorem 4.5.8} \quad \} \\ & P.1 \\ = & \{ \quad \text{arithmetic} \quad \} \\ & \langle \Sigma k : 0 \leq k < m : c_k \rangle \quad . \end{array}$$

The conclusion is, as expected,

$$n \cong \langle \Sigma k : 0 \leq k < m : c_k \rangle \text{ (mod 3)} \quad .$$

Cancellation properties We have seen before that

$$[\, a \times c \;\cong\; b \times c \;\text{ (mod } n) \;\Leftarrow\; a \cong b \text{ (mod } n) \,] \quad .$$

The converse, however, fails to hold. Nevertheless, we can use the following theorem, which can be seen as a cancellation property:

Theorem 4.5.9 For non-zero c and n, we have

$$[\, a \cong b \text{ (mod } \frac{n}{c \bigtriangledown n}) \equiv a \times c \;\cong\; b \times c \;\text{ (mod } n) \,] \quad .$$

Proof

$$\begin{array}{ll} & a \cong b \text{ (mod } \frac{n}{c \bigtriangledown n}) \\ = & \{ \quad \text{definition and } [\; n \backslash a \equiv n \backslash |a| \;] \quad \} \\ & \frac{n}{c \bigtriangledown n} \backslash |a - b| \\ = & \{ \quad \text{definition of division} \quad \} \end{array}$$

$$
\begin{array}{ll}
 & \langle \exists k :: |a-b| = k \times \dfrac{n}{c \bigtriangledown n} \rangle \\
= & \quad \{ \quad \text{cancellation } (c \bigtriangledown n \neq 0) \quad \} \\
 & \langle \exists k :: |a-b| \times c \bigtriangledown n = k \times n \rangle \\
= & \quad \{ \quad \text{multiplication by a natural number} \\
 & \qquad \text{distributes over the greatest common divisor} \quad \} \\
 & \langle \exists k :: (|a-b| \times c) \bigtriangledown (|a-b| \times n) = k \times n \rangle \\
= & \quad \{ \quad \text{definition of division} \quad \} \\
 & n \backslash (|a-b| \times c) \bigtriangledown (|a-b| \times n) \\
= & \quad \{ \quad n \text{ divides } |a-b| \times n \quad \} \\
 & n \backslash (|a-b| \times c) \\
= & \quad \{ \quad \text{distributivity, definition, and } [\ n \backslash a \equiv n \backslash |a|\ ] \quad \} \\
 & a \times c \cong b \times c \pmod{n} \ .
\end{array}
$$

□

A corollary of this theorem is:

Corollary 4.5.10 $[\ a \cong b \pmod{n} \equiv a \times c \cong b \times c \pmod{n} \Leftarrow c \bigtriangledown n = 1\ ]$

Proof

$$
\begin{array}{ll}
 & a \times c \cong b \times c \pmod{n} \\
= & \quad \{ \quad \text{Theorem 4.5.9} \quad \} \\
 & a \cong b \pmod{\dfrac{n}{c \bigtriangledown n}} \\
= & \quad \{ \quad \text{assumption } c \bigtriangledown n = 1 \quad \} \\
 & a \cong b \pmod{n} \ .
\end{array}
$$

□

There is a special case of this corollary that can be useful:

$$[\ a \cong b \pmod{n} \ \equiv \ a \times c \ \cong \ b \times c \pmod{n} \quad \Leftarrow \quad n \text{ is prime} \wedge \neg (p \backslash c)\ ] \ .$$

This last theorem was the first where we have manipulated the modulus part. There is, however, another theorem that involves the modulus part:

Theorem 4.5.11 $[\ a \cong b \pmod m \wedge a \cong b \pmod n \ \equiv\ a \cong b \pmod{m \triangle n}\ ]$

Proof

$$
\begin{array}{ll}
 & a \cong b \pmod m \wedge a \cong b \pmod n \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & m \backslash (a-b) \wedge n \backslash (a-b) \\
= & \quad \{ \quad \text{definition of least common multiple} \quad \} \\
 & (m \triangle n) \backslash (a-b) \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & a \cong b \pmod{m \triangle n} \ .
\end{array}
$$

□

The special case $m \perp n$ of this law is important, because $m \triangle n = m \times n$ when $m \perp n$. Therefore, we will state it explicitly:

$$[\ (a \cong b \pmod m \ \wedge\ a \cong b \pmod n \ \equiv\ a \cong b \pmod{m \times n}) \quad \Leftarrow \quad m \perp n\ ]\ .$$

4.5.2 Modular exponentiation

A problem that arises frequently is to calculate the smallest y such that

$$y \cong a^k \pmod n \ ,$$

that is, to calculate a y such that

$$y = a^k \bmod n \ .$$

A straightforward algorithm is to multiply a by itself k times and then reduce it modulo n, but it is possible to do it much more efficiently. First, we formally specify the algorithm:

$$
\begin{array}{l}
\{\ a \in \mathbb{Z} \ \wedge\ k \in \mathbb{N}\ \} \\
\quad \text{Compute a value } y \\
\{\ y = a^k \bmod n\ \}\ .
\end{array}
$$

A common technique that we use when exponentiation is involved is to write the exponent as a sum of binary powers. In this way, we may reduce the number of multiplications. Noting that k can be written as

$$\langle \Sigma i : 0 \le i < log.k : k_i \times 2^i \rangle \ ,$$

where k_i corresponds to the i^{th} bit (from right) in the binary representation of k, we can rewrite the postcondition shown above:

$$
\begin{array}{ll}
 & y = a^k \bmod n \\
= & \quad \{ \quad \text{binary expansion of } k \quad \} \\
 & y = a^{\langle \Sigma i \,:\, 0 \le i < log.k \,:\, k_i \times 2^i \rangle} \bmod n \\
= & \quad \{ \quad \text{exponentiation distributes over addition} \quad \} \\
 & y = \langle \Pi i \,:\, 0 \le i < log.k \,:\, a^{k_i \times 2^i} \rangle \bmod n \\
= & \quad \{ \quad \text{the values } k_i \text{ are bits, so we use them as the} \\
 & \qquad \text{outermost exponents} \quad \} \\
 & y = \langle \Pi i \,:\, 0 \le i < log.k \,:\, (a^{2^i})^{k_i} \rangle \bmod n \\
= & \quad \{ \quad \bmod \text{ distributes over multiplication;} \\
 & \qquad \text{the values } k_i \text{ are bits} \quad \} \\
 & y = \langle \Pi i \,:\, 0 \le i < log.k \,:\, (a^{2^i} \bmod n)^{k_i} \rangle \bmod n \quad .
\end{array}
$$

Using this new postcondition, we can rewrite the formal specification as

$$
\begin{array}{l}
\{ \;\; a \in \mathbb{Z} \;\wedge\; k \in \mathbb{N} \;\; \} \\
\quad \text{Compute a value } y \\
\{ \;\; y = \langle \Pi i \,:\, 0 \le i < log.k \,:\, (a^{2^i} \bmod n)^{k_i} \rangle \bmod n \;\; \} \quad .
\end{array}
$$

Replacing the constant $log.k$ by a variable n, we get the first version of the algorithm by using the same technique as before (one of the conjuncts is chosen for the invariant and the negation of the other for the guard):

$$
\begin{array}{l}
\{ \;\; a \in \mathbb{Z} \;\wedge\; k \in \mathbb{N} \;\; \} \\
y, n \;:=\; 1, 0 \\
\{ \;\; \textbf{Invariant: } P \;\; \}; \\
\mathsf{do}\; n \neq log.k \;\rightarrow\; y, n \;:=\; Y, n{+}1 \;\mathsf{od} \\
\{ \;\; P \;\wedge\; n = log.k \;\; \},
\end{array}
$$

where

$$
P \;\equiv\; y = \langle \Pi i : 0 \le i < n : (a^{2^i} \bmod n)^{k_i} \rangle \bmod n \quad .
$$

Now, we can calculate Y in a way that P is preserved. Using the assignment axiom, we calculate as follows:

$$
\begin{array}{ll}
& P[y\,,n \;:=\; Y\,,n{+}1] \\
= & \quad \{ \quad \text{substitution} \quad \} \\
& Y = \langle \Pi i \,:\, 0 \le i < n{+}1 \,:\, (a^{2^i} \bmod n)^{k_i} \rangle \bmod n \\
= & \quad \{ \quad \text{range splitting, distributivity, and invariant} \quad \} \\
& Y \;=\; y \times (a^{2^n} \bmod n)^{k_n} \bmod n \\
= & \quad \{ \quad k_n \text{ is either 0 or 1} \quad \} \\
& Y = y \times (a^{2^n} \bmod n)^{k_n} \quad .
\end{array}
$$

The algorithm becomes:

$$
\begin{array}{l}
\{ \;\; a \in \mathbb{Z} \;\wedge\; k \in \mathbb{N} \;\; \} \\
y\,,n \;:=\; 1\,,0 \\
\{ \;\; \textbf{Invariant: } P \;\; \}; \\
\mathsf{do}\; n \neq log.k \;\rightarrow\; y\,,n \;:=\; y \times (a^{2^n} \bmod n)^{k_n}\,,n{+}1\; \mathsf{od} \\
\{ \;\; P \;\wedge\; n = log.k \;\; \}.
\end{array}
$$

Although conditionally correct, we can optimise this algorithm by strengthening the invariant with the equality

$$z = a^{2^n} \bmod n \quad .$$

(We have used this technique in the derivation of the division algorithm — see section 4.2.1.) The shape of the assignment in the loop becomes

$$y\,,n\,,z \;:=\; y \times z^{k_n}\,,n{+}1\,,Z \quad ,$$

and we calculate Z as follows:

$$
\begin{array}{ll}
& (z = a^{2^n} \bmod n)[n\,,z \;:=\; n{+}1\,,Z] \\
= & \quad \{ \quad \text{substitution} \quad \} \\
& Z = a^{2^{n+1}} \bmod n \\
= & \quad \{ \quad \text{arithmetic} \quad \} \\
& Z = (a^{2^n})^2 \bmod n
\end{array}
$$

$$
\begin{array}{ll}
= & \{ \quad \text{invariant} \ \} \\
& Z = (z \times z) \bmod n \ .
\end{array}
$$

The optimised version of the algorithm is:

$$
\begin{array}{l}
\{ \ a \in \mathbb{Z} \ \wedge \ k \in \mathbb{N} \ \} \\
y \, , n \, , z \ := \ 1 \, , 0 \, , a \\
\{ \ \textbf{Invariant:} \ P \ \wedge \ z = a^{2^n} \bmod n \ \}; \\
\mathsf{do} \ n \neq log.k \ \rightarrow \ y \, , n \, , z \ := \ y \times z^{k_n} \, , n{+}1 \, , (z \times z) \bmod n \ \mathsf{od} \\
\{ \ P \ \wedge \ n = log.k \ \}.
\end{array}
$$

It is not difficult to prove termination (a bound function is $log.k - n$). Note that, because k_n is a bit (i.e. either 0 or 1), the algorithm can be rewritten as:

$$
\begin{array}{l}
\{ \ a \in \mathbb{Z} \ \wedge \ k \in \mathbb{N} \ \} \\
y \, , n \, , z \ := \ 1 \, , 0 \, , a \\
\{ \ \textbf{Invariant:} \ P \ \wedge \ z = a^{2^n} \bmod n \ \}; \\
\mathsf{do} \ n \neq log.k \ \rightarrow \\
\qquad \mathsf{if} \ \ k_n = 1 \ \rightarrow \ y \ := \ y \times z^{k_n} \ \ \mathsf{fi} \ ; \\
\qquad n \, , z \ := \ n{+}1 \, , (z \times z) \bmod n \\
\mathsf{od} \\
\{ \ P \ \wedge \ n = log.k \ \}.
\end{array}
$$

4.5.3 On a simple version of the Chinese remainder theorem

The goal of this section is to present the design of an algorithm that computes a solution x for the following simultaneous congruence equations

$$x \cong a \ (\text{mod } m) \ \wedge \ x \cong b \ (\text{mod } n) \quad .$$

We construct the algorithm in two different ways. The first is extracted from an existence proof, whilst the second starts with a functional specification and is based on conventional methods for deriving algorithms.

This section was jointly written with Arjan Mooij.

Studying the existence of solutions

We start our analysis by investigating when a solution to our problem exists. The formulation is straightforward and the calculation proceeds as follows:

$$
\begin{array}{ll}
 & \langle \exists x :: x \cong a \ (\mathrm{mod}\ m) \ \wedge \ x \cong b \ (\mathrm{mod}\ n) \rangle \\
= & \quad \{ \quad \text{definition of congruence (twice)} \quad \} \\
 & \langle \exists x :: m \backslash (x-a) \ \wedge \ n \backslash (x-b) \rangle \\
= & \quad \{ \quad \text{definition of division (twice), cancellation,} \\
 & \qquad \text{distributivity and nesting} \quad \} \\
 & \langle \exists x, i, j :: \ x = a + i \times m \ \wedge \ x = b + j \times n \rangle \\
= & \quad \{ \quad \text{trading and one-point rule} \quad \} \\
 & \langle \exists i, j :: \ a + i \times m = b + j \times n \rangle \\
= & \quad \{ \quad \text{cancellation} \quad \} \\
 & \langle \exists i, j :: \ a - b = j \times n - i \times m \rangle \\
= & \quad \{ \quad \text{property of the greatest common divisor,} \\
 & \qquad \text{here denoted by the infix operator } \triangledown \text{; more details below} \quad \} \\
 & \langle \exists k :: \ a - b = k \times (m \triangledown n) \rangle \\
= & \quad \{ \quad \text{definition of mod} \quad \} \\
 & a \cong b \ (\mathrm{mod}\ m \triangledown n) \ .
\end{array}
$$

The fifth step of the above calculation is the only one that is not well motivated. In fact, we could have stopped after the third step and conclude that there is a solution to our problem if we can write $a-b$ as a linear combination of m and n ($j \times n - i \times m$). However, we know that we can write any linear combination of m and n as a multiple of $m \triangledown n$. This leads to a shorter condition based on the operator mod. The validity of the step is established by mutual implication, as follows:

$$
\begin{array}{ll}
 & \langle \exists i, j :: \ a - b = j \times n - i \times m \rangle \\
= & \quad \{ \quad m \triangledown n \backslash m \ \wedge \ m \triangledown n \backslash n \text{, arithmetic} \quad \} \\
 & \langle \exists i, j :: \ a - b = (\dfrac{j \times n}{m \triangledown n} - \dfrac{i \times m}{m \triangledown n}) \times m \triangledown n \rangle \\
\Rightarrow & \quad \{ \quad k := \dfrac{j \times n}{m \triangledown n} - \dfrac{i \times m}{m \triangledown n} \quad \} \\
 & \langle \exists k :: \ a - b = k \times (m \triangledown n) \rangle \ .
\end{array}
$$

The other direction is based on the extended Euclid's algorithm:

$$
\begin{array}{ll}
 & \langle \exists k :: \ a-b = k \times (m \bigtriangledown n) \rangle \\
= & \quad \{ \quad \text{Euclid's algorithm} \quad \} \\
 & \langle \exists x, y, k \ : \ m \bigtriangledown n \ = \ x \times m + y \times n \ : \ a-b = k \times (x \times m + y \times n) \rangle \\
\Rightarrow & \quad \{ \quad \text{distributivity, associativity, and } i \ := \ -k \times x, j := k \times y \quad \} \\
 & \langle \exists i, j :: \ a-b \ = \ j \times n - i \times m \rangle \ .
\end{array}
$$

Extracting an algorithm from the existence proof

The existence proof is very instructive and can be used directly to build an algorithm. Reading it backwards, we first construct a k such that

$$k = \frac{a-b}{m \bigtriangledown n} \ .$$

Afterwards, we can use the extended Euclid's algorithm to compute a linear combination $i \times m + j \times n$ (in fact, Euclid's algorithm can be used to compute $m \bigtriangledown n$ and the linear combination in one go). Finally, we can find a solution by computing $a - i \times k \times m$ (or $b + j \times k \times n$). The three-step algorithm is:

$$
\begin{array}{l}
k \ := \ \dfrac{a-b}{m \bigtriangledown n} \ ; \\
i \, , j \ : \ m \bigtriangledown n \ = \ i \times m + j \times n \ ; \\
x \ := \ a - i \times k \times m
\end{array}
$$

We now show how to avoid the computation of k:

$$
\begin{array}{ll}
 & a - i \times k \times m \\
= & \quad \{ \quad \text{value of } k \quad \} \\
 & a - i \times \dfrac{a-b}{m \bigtriangledown n} \times m \\
= & \quad \{ \quad \text{distributivity} \quad \} \\
 & a - a \times \dfrac{i \times m}{m \bigtriangledown n} + b \times \dfrac{i \times m}{m \bigtriangledown n} \\
= & \quad \{ \quad \text{From Euclid's algorithm: } \dfrac{i \times m}{m \bigtriangledown n} = 1 - \dfrac{j \times n}{m \bigtriangledown n} \quad \} \\
 & a \times \dfrac{j \times n}{m \bigtriangledown n} + b \times \dfrac{i \times m}{m \bigtriangledown n} \\
= & \quad \{ \quad \text{arithmetic} \quad \} \\
 & \dfrac{a \times j \times n + b \times i \times m}{m \bigtriangledown n} \ .
\end{array}
$$

It is interesting to observe that if $m \bigtriangledown n = 1$, our algorithm computes the solution

$$a \times j \times n + b \times i \times m \quad .$$

Constructing an algorithm from the functional specification

The functional specification of the algorithm that we want to construct is

$$\{ \ 0 < m \ \wedge \ 0 < n \ \wedge \ \langle \exists x :: x \cong a \ (\mathrm{mod}\ m) \ \wedge \ x \cong b \ (\mathrm{mod}\ n) \rangle \ \}$$

$$S$$

$$\{ \ x \cong a \ (\mathrm{mod}\ m) \ \wedge \ x \cong b \ (\mathrm{mod}\ n) \ \} \ .$$

Turning our attention to the postcondition, a common technique is suggested by its shape. First, we rewrite it to the equivalent expression:

$$x \cong a \ (\mathrm{mod}\ m) \ \wedge \ y \cong b \ (\mathrm{mod}\ n) \wedge x = y \quad .$$

Now, with the postcondition rewritten in this new shape, we can take the first two conjuncts to be the invariant and the negation of the third conjunct to be the guard of a repetition statement. The new invariant is very easy to initialise (the statement $x , y \ := \ a , b$ will do) and we get the next version:

$$\{ \ 0 < m \ \wedge \ 0 < n \ \wedge \ \langle \exists x :: x \cong a \ (\mathrm{mod}\ m) \ \wedge \ x \cong b \ (\mathrm{mod}\ n) \rangle \ \}$$

$$x , y \ := \ a , b \ ;$$

$$\{ \ \textbf{Invariant}: \quad x \cong a \ (\mathrm{mod}\ m) \ \wedge \ y \cong b \ (\mathrm{mod}\ n) \ \}$$

$$\mathsf{do} \ \ x \neq y \rightarrow S$$

$$\mathsf{od}$$

$$\{ \ x \cong a \ (\mathrm{mod}\ m) \ \wedge \ y \cong b \ (\mathrm{mod}\ n) \wedge x = y \ \} \ .$$

Refining the guard and the loop statement Clearly, the goal of the loop statement is to change variables x and y without violating the invariant, and in a way such that x and y are equal on termination. Common strategies for changing integer variables so that they become equal are to increase the smallest one or to decrease the largest one, both iteratively. Adopting one of these strategies means that we should rewrite the guard to know at each point what is the smallest or the largest variable, and that we should investigate properties of the "mod" operator involving addition, so that the invariant is not violated.

Rewriting the guard causes no problem, since the following is valid:

$$x \neq y \quad \equiv \quad x < y \ \vee \ y < x \quad .$$

Regarding properties of the "mod" operator involving addition, the following law is extremely useful:

(4.5.12) $[\, a + k \times m \ \cong \ b \pmod{m} \ \Leftarrow \ a \cong b \pmod{m} \,]$.

(This law follows from property 1 of theorem 4.5.8.) In fact, from the above observations and from this law, we get the next version of the program:

$\{ \ 0 < m \ \wedge \ 0 < n \wedge \langle \exists x :: x \cong a \pmod{m} \ \wedge \ x \cong b \pmod{n} \rangle \ \}$

$x \,, y \ := \ a \,, b \ ;$

$\{$ **Invariant**: $\quad x \cong a \pmod{m} \ \wedge \ y \cong b \pmod{n} \ \}$

do $x < y \ \rightarrow \ x \ := \ x + m$

$\Box \ y < x \ \rightarrow \ y \ := \ y + n$

od

$\{ \ x \cong a \pmod{m} \ \wedge \ y \cong b \pmod{n} \wedge x = y \ \}$.

Is is not difficult to see that the invariant is preserved by the loop statement (just use (4.5.12) with $k \ := \ 1$). This version is correct, but we still have to prove termination.

Proof of termination To prove termination, we have to find an appropriate bound function. If we can prove the existence of a number z such that the following property

(4.5.13) $x \leq z \ \wedge \ y \leq z$

is a loop invariant, then we can choose as bound function the function bf defined as

$$bf.(x, y) = (z - x) + (z - y) \quad .$$

The function bf decreases because x and y are iteratively increased and if (4.5.13) holds, then it is natural-valued, i.e., it is bounded below.

So, now we just have to find an appropriate z such that (4.5.13) is a loop invariant. Since the solution we are building is obtained by increasing x or y, then we can assume that a solution z at least a and b exists, i.e.:

$$\langle \exists z \ : \ a \leq z \ \wedge \ b \leq z \ : \ z \cong a \pmod{m} \ \wedge \ z \cong b \pmod{n} \rangle \quad .$$

Using this fact as precondition, we now try to prove the invariant (4.5.13). To prove it, we have to prove that the following holds:

$$x+m \leq z \;\Leftarrow\; x \leq z \qquad \text{and}$$

$$y+n \leq z \;\Leftarrow\; y \leq z \quad .$$

For the first conjunct, the relevant properties that are in the context are:

(4.5.14) $x \cong a \pmod{m}$,

(4.5.15) $z \cong a \pmod{m}$, and

(4.5.16) $x < y \leq z$.

From (4.5.14) and (4.5.15) we can easily prove that $z \cong x \pmod{m}$, i.e., there is an integer j such that $z = x + j \times m$. From (4.5.16), we conclude that $0 < j$. We use these two facts to prove the first conjunct:

$$\begin{array}{ll} & x+m \leq z \\ = & \quad \{ \quad z = x + j \times m \quad \} \\ & x+m \leq x + j \times m \\ = & \quad \{ \quad 0 < j \quad \} \\ & \mathsf{true} \ . \end{array}$$

The proof of the other conjunct is very similar and the fully annotated program is:

$$\begin{array}{l} \{ \; 0 < m \;\wedge\; 0 < n \wedge \langle \exists z : a \leq z \;\wedge\; b \leq z : z \cong a \pmod{m} \;\wedge\; z \cong b \pmod{n} \rangle \; \} \\ x \,, y \;:=\; a \,, b \;; \\ \{ \; \textbf{Invariant:} \quad x \cong a \pmod{m} \;\wedge\; y \cong b \pmod{n} \wedge x \leq z \;\wedge\; y \leq z \; \} \\ \mathsf{do} \; x < y \;\rightarrow\; x \;:=\; x+m \\ \Box \;\; y < x \;\rightarrow\; y \;:=\; y+n \\ \mathsf{od} \\ \{ \; x \cong a \pmod{m} \;\wedge\; y \cong b \pmod{n} \wedge x = y \; \} \ . \end{array}$$

Computing the set of all solutions

We have presented when a solution to the problem exists and how to compute it. However, that solution is not unique. In this section we show how to compute the set of all solutions.

For any a,b,m and n, we assume that

$$\langle \exists i,j:: \ a + i\times m \ = \ b + j\times n \rangle \quad .$$

Let us calculate the set of all solutions of this equation:

$$\begin{array}{ll}
 & \{f,g:\ a + f\times m \ = \ b + g\times n:\ a + f\times m\} \\
= & \quad \{ \quad \text{cancellation} \quad \} \\
 & \{f,g:\ a-b \ = \ g\times n - f\times m:\ a + f\times m\} \\
= & \quad \{ \quad \text{use assumption} \quad \} \\
 & \{f,g:\ j\times n - i\times m \ = \ g\times n - f\times m:\ a + f\times m\} \\
= & \quad \{ \quad \text{cancellation and distributivity} \quad \} \\
 & \{f,g:\ (j-g)\times n = (i-f)\times m:\ a + f\times m\} \\
= & \quad \{ \quad \text{dummy renaming: } f \ := \ i-f \text{ and } g \ := \ j-g \quad \} \\
 & \{f,g:\ g\times n = f\times m:\ (a + i\times m) - f\times m\} \\
= & \quad \{ \quad \text{one-point rule and definition of division} \quad \} \\
 & \{z:\ m\backslash z \wedge n\backslash z:\ (a + i\times m) - z\} \\
= & \quad \{ \quad \text{definition of the least common multiple} \quad \} \\
 & \{z:\ (m \mathbin{\triangle} n)\backslash z:\ (a + i\times m) - z\} \\
= & \quad \{ \quad \text{definition of division and one-point rule} \quad \} \\
 & \{k::\ (a + i\times m) - k\times (m \mathbin{\triangle} n)\} \quad .
\end{array}$$

So, given a solution, the set of all solutions are the values that are equal to it modulo $m \mathbin{\triangle} n$. That is,

$$\{s:\ \ s \ = \ a + i\times m \ (\mathsf{mod} \ (m \mathbin{\triangle} n)):\ \ s\} \quad ,$$

where $a + i\times m$ denotes a solution.

4.6 Designing an algorithmic proof of the two-squares theorem

Which numbers can be written as sums of two squares? According to Dickson [Dic99, p. 225], this classic question in number theory was first discussed by Diophantus, but it is usually associated with Fermat, who stated in 1659 that he possessed an irrefutable proof that every prime of the form $4k + 1$ can be written as the sum of two squares. (He first communicated the result to Mersenne, in a letter dated December 25, 1640; for this reason, this result is sometimes called *Fermat's Christmas Theorem*. Incidentally, Dickson names this result after Albert Girard, who, in 1632, was the first to state it. We follow Dickson's convention and we also refer to the two-squares theorem as Girard's result.) However, as with many other of his results, Fermat did not record his proof. The first recorded proof of Girard's result is due to Euler who proved it in 1749, "after he had struggled, off and on, for *seven years* to find a proof" [Bel08, p. 69]. Euler communicated his five-step argument in a letter to Goldbach dated 6 May 1747, but the fifth step was only made precise in a second letter written in 1749. In 1801, Gauss proved for the first time that such prime numbers are *uniquely* represented as the sum of two positive integers [Gau01, Art. 182].

This classic theorem attracted the attention of many mathematicians. Since Euler's proof by the method of infinite descent, Lagrange proved it using quadratic forms (subsequently, Gauss simplified Lagrange's proof in [Gau01, Art. 182]); Dedekind used Gaussian integers; Serret and Hermite used continued fractions [Her48, Ser48]; Brillhart improved Hermite's argument using Euclid's algorithm [Bri72]; Smith used continuants [CELV99]; more recently, Zagier [Zag90] published a one-sentence proof based on an involution of a particular finite set (see also [AZ04, chapter 4] and [Dij93] for a detailed explanation of the proof); and Wagon [Wag90] gave a self-contained proof based on Euclid's algorithm and on [Bri72].

Like Brillhart and Wagon, we present a proof that is also based on Euclid's algorithm, but, rather than simply verifying Girard's result, we use the algorithm as an interface to *investigate* which numbers can be written as sums of two positive squares[14]. The precise formulation of the problem as an algorithmic problem is the key, since it allows us to use algorithmic techniques and to avoid guessing. The notion of invariance, in particular, plays a central role in our development: it is used initially to observe that

[14]Every square number m^2 can be written as m^2+0^2. However, this type of solution is not considered in this section, since our formulation of Euclid's algorithm deals only with positive numbers. Therefore, our construction aims to express a number as the sum of two *positive* squares.

Euclid's algorithm can actually be used to represent a given number as a sum of two positive squares, and then it is used throughout the argument to prove other relevant properties. We also show how the use of program inversion techniques can make mathematical arguments more precise. As we will see, the end result is also more general than the one conjectured by Girard.

In the next section we show how our reformulation of Euclid's algorithm in terms of matrices (see section 4.3.2) can be used to prove the theorem. At the end of the section, we describe how the argument is organised.

4.6.1 Euclid's algorithm

Recall that in section 4.3.2, we have reformulated the so-called Extended Euclid's algorithm in terms of matrices:

$$\{\ 0<m\ \wedge\ 0<n\ \}$$

$$(x\ y)\,,\mathbf{C}\ :=\ (m\ n)\,,\mathbf{I}\ ;$$

$$\{\ \textbf{Invariant:}\quad (x\ y)\ =\ (m\ n)\times\mathbf{C}\ \}$$

$$\textsf{do}\ \ y<x\quad\rightarrow\quad (x\ y)\,,\mathbf{C}\ :=\ (x\ y)\times\mathbf{A}\ ,\ \mathbf{C}\times\mathbf{A}$$

$$\Box\ \ x<y\quad\rightarrow\quad (x\ y)\,,\mathbf{C}\ :=\ (x\ y)\times\mathbf{B}\ ,\ \mathbf{C}\times\mathbf{B}$$

$$\textsf{od}$$

$$\{\ (x\ y)\ =\ (m\triangledown n\ \ m\triangledown n)\ =\ (m\ n)\times\mathbf{C}\ \}$$

Specifically, $\mathbf{I}$, $\mathbf{A}$, and $\mathbf{B}$ are 2×2 matrices; $\mathbf{I}$ is the identity matrix $\left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right)$, $\mathbf{A}$ is the matrix $\left(\begin{smallmatrix}1&0\\-1&1\end{smallmatrix}\right)$, and $\mathbf{B}$ is the matrix $\left(\begin{smallmatrix}1&-1\\0&1\end{smallmatrix}\right)$.

A key insight exploited in section 4.4.2 is that matrices $\mathbf{A}$ and $\mathbf{B}$ are invertible, which allows us to rewrite the invariant as $(x\ y)\times\mathbf{C}^{-1}=(m\ n)$, where the matrix $\mathbf{C}^{-1}$ is a finite product of the matrices $\mathbf{A}^{-1}$ and $\mathbf{B}^{-1}$, which are, respectively, $\left(\begin{smallmatrix}1&0\\1&1\end{smallmatrix}\right)$ and $\left(\begin{smallmatrix}1&1\\0&1\end{smallmatrix}\right)$. In fact, we have seen that we can change the above algorithm to compute the matrix $\mathbf{C}^{-1}$ instead; renaming $\mathbf{C}^{-1}$ to $\mathbf{D}$, $\mathbf{A}^{-1}$ to $\mathbf{L}$, and $\mathbf{B}^{-1}$ to $\mathbf{R}$, we rewrite it as follows:

$$\{\ 0<m\ \wedge\ 0<n\ \}$$

$$(x\ y)\,,\mathbf{D}\ :=\ (m\ n)\,,\mathbf{I}\ ;$$

$$\{\ \textbf{Invariant:}\quad (x\ y)\times\mathbf{D}\ =\ (m\ n)\ \}$$

$$\textsf{do}\ \ y<x\quad\rightarrow\quad (x\ y)\,,\mathbf{D}\ :=\ (x\ y)\times\mathbf{L}^{-1}\ ,\ \mathbf{L}\times\mathbf{D}$$

$$\Box\ \ x<y\quad\rightarrow\quad (x\ y)\,,\mathbf{D}\ :=\ (x\ y)\times\mathbf{R}^{-1}\ ,\ \mathbf{R}\times\mathbf{D}$$

od

$$\{\ (x\ y)\ =\ (m\triangledown n\ \ m\triangledown n)\qquad\wedge\qquad (m\triangledown n\ \ m\triangledown n)\times\mathbf{D}\ =\ (m\ n)\ \}$$

It is this form of the algorithm that is the starting point for our investigation. Note that if $\mathbf{D}=\left(\begin{smallmatrix}a & b\\ c & d\end{smallmatrix}\right)$, the invariant is equivalent to

$$(m\ n)\ \ =\ \ (x\ y)\times\mathbf{D}\ \ =\ \ (x{\times}a+y{\times}c\ \ \ x{\times}b+y{\times}d)\ \ ,$$

which means that if, at any point in the execution of the algorithm, $(x\ y)$ equals $(a\ c)$, we can conclude that m is a sum of two positive squares, that is:

$$(m\ n)\ \ =\ \ (a\ c)\times\mathbf{D}\ \ =\ \ (a{\times}a+c{\times}c\ \ \ a{\times}b+c{\times}d)\ \ .$$

Symmetrically, if, at any point in the execution of the algorithm, $(x\ y)$ equals $(b\ d)$, we can conclude that n is a sum of two positive squares.

It may help to visualise an execution trace of the algorithm. Table 4.1 depicts the execution trace when the arguments are $m=17$ and $n=4$. Each row of the table shows the state-space and the value of the invariant after each iteration of the algorithm. The first two columns show the values of the variables $(x\ y)$ and $\mathbf{D}$, respectively. The third column shows how the invariant is satisfied, according to the values of the first two columns. The first row corresponds to the initial state and the last row corresponds to the final state.

$(x\ y)$	$\mathbf{D}$, the same as $\left(\begin{smallmatrix}a & b\\ c & d\end{smallmatrix}\right)$	**Invariant:** $(m\ n)=(x{\times}a+y{\times}c\ \ \ x{\times}b+y{\times}d)$
$(17\ 4)$	$\left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right)=\mathbf{I}$	$(17\ 4)=(17{\times}1+4{\times}0\ \ \ 17{\times}0+4{\times}1)$
$(13\ 4)$	$\left(\begin{smallmatrix}1 & 0\\ 1 & 1\end{smallmatrix}\right)=\mathbf{L}$	$(17\ 4)=(13{\times}1+4{\times}1\ \ \ 13{\times}0+4{\times}1)$
$(9\ 4)$	$\left(\begin{smallmatrix}1 & 0\\ 2 & 1\end{smallmatrix}\right)=\mathbf{LL}$	$(17\ 4)=(9{\times}1+4{\times}2\ \ \ 9{\times}0+4{\times}1)$
$(5\ 4)$	$\left(\begin{smallmatrix}1 & 0\\ 3 & 1\end{smallmatrix}\right)=\mathbf{LLL}$	$(17\ 4)=(5{\times}1+4{\times}3\ \ \ 5{\times}0+4{\times}1)$
$(\mathbf{1}\ \mathbf{4})$	$\left(\begin{smallmatrix}\mathbf{1} & 0\\ \mathbf{4} & 1\end{smallmatrix}\right)=\mathbf{LLLL}$	$(17\ 4)=(\mathbf{1}{\times}\mathbf{1}+\mathbf{4}{\times}\mathbf{4}\ \ \ 1{\times}0+4{\times}1)$
$(1\ 3)$	$\left(\begin{smallmatrix}5 & 1\\ 4 & 1\end{smallmatrix}\right)=\mathbf{RLLLL}$	$(17\ 4)=(1{\times}5+3{\times}4\ \ \ 1{\times}1+3{\times}1)$
$(1\ 2)$	$\left(\begin{smallmatrix}9 & 2\\ 4 & 1\end{smallmatrix}\right)=\mathbf{RRLLLL}$	$(17\ 4)=(1{\times}9+2{\times}4\ \ \ 1{\times}2+2{\times}1)$
$(1\ 1)$	$\left(\begin{smallmatrix}13 & 3\\ 4 & 1\end{smallmatrix}\right)=\mathbf{RRRLLLL}$	$(17\ 4)=(1{\times}13+1{\times}4\ \ \ 1{\times}3+1{\times}1)$

Table 4.1: Execution trace of Euclid's algorithm for arguments $m=17$ and $n=4$

As we can see in table 4.1, there is a point at which $x=a=1$ and $y=c=4$; it follows directly from the invariant that 17 can be expressed as the sum of two positive squares

$(17 = 1^2+4^2)$.

One question that now arises is what is so special about the numbers 17 and 4 that made the vectors $(x\ y)$ and $(a\ c)$ to be equal. (Had we used as arguments the numbers 17 and 5, for example, x would never equal a.) Put more generally, how can we characterise the arguments that make the vectors $(x\ y)$ and $(a\ c)$ to be equal at some point in the execution of the algorithm?

A closer inspection of the values shown in table 4.1 can help us answering the general question. If we ignore the first row, we see that the sequence of successive values of the vector $(x\ y)$ is the reverse of the sequence of successive values of $(a\ c)$. Also, because the length of these sequences is the same and odd, there is a middle point at which $(x\ y)=(a\ c)$. So, one way of proving that at some point in the execution of the algorithm the vectors $(x\ y)$ and $(a\ c)$ are equal is to prove that the sequences of successive values of the vectors $(x\ y)$ and $(a\ c)$, with the exception of the initial values, are reverses of each other and that both sequences have odd length. (In the example above, the length is 7.)

Taking this analysis into account, the question can be reformulated as: for which arguments m and n does Euclid's algorithm produce odd-length sequences of successive values of the vectors $(x\ y)$ and $(a\ c)$ that are reverses of each other?

Our answer to this question is divided in three parts. First, in section 4.6.2, we invert Euclid's algorithm to prove that the operations performed on the vector $(x\ y)$ are the same as those performed on the vector $(a\ c)$ when running the algorithm backwards. Second, in section 4.6.3, we determine necessary and sufficient conditions on the arguments m and n to make the initial value of the vector $(x\ y)$ equal the final value of the vector $(a\ c)$. These two parts together characterise the arguments for which the sequences of vectors are each other's reverses. Finally, in section 4.6.4, we show that if the sequences are the reverses of each other, they must have odd length.

Note that our investigation aims at expressing the argument m as a sum of two positive squares — that is why we focus on vectors $(x\ y)$ and $(a\ c)$. This means that, given a value m, we want to characterise which values n can be chosen to be passed along with m as arguments of the algorithm (we perform this characterisation in section 4.6.3).

For brevity, and whenever the context is unambiguous, we shall refer to "the sequences" to mean "the sequences of successive values of the vectors $(x\ y)$ and $(a\ c)$" and to "the sequences are reversed" to mean "the sequences are the reverses of each other". Also, we assume throughout that $\mathbf{D}=\begin{pmatrix} a & b \\ c & d \end{pmatrix}$.

4.6.2 Inverting Euclid's algorithm

In section 3.6, we have seen that, if c_0 and c_1 are constants, the inverse of

(4.6.1) $v := c_0 \ ; \ S \ \{ v = c_1 \}$

is

$$v := c_1 \ ; \ S^{-1} \ \{ v = c_0 \} \ .$$

Since Euclid's algorithm is an instance of (4.6.1) — instantiate v with the variables $(x \ y)$ and $\mathbf{D}$, and consider S to be the loop — its inverse is:

$$(x \ y) := (m \triangledown n \ \ m \triangledown n) \ ;$$
$$\text{initialise } \mathbf{D} \text{ such that } (m \triangledown n \ \ m \triangledown n) \times \mathbf{D} \ = \ (m \ n) \, ;$$
$$S^{-1}$$
$$\{ \ (x \ y) \ = \ (m \ n) \quad \wedge \quad \mathbf{D} \ = \ \mathbf{I} \ \} \ .$$

That is, provided that we initialise $(x \ y)$ to $(m \triangledown n \ \ m \triangledown n)$ and the matrix $\mathbf{D}$ in a way that satisfies $(m \triangledown n \ \ m \triangledown n) \times \mathbf{D} \ = \ (m \ n)$, undoing S terminates in a state where $(x \ y)$ and $\mathbf{D}$ equal their initial values in Euclid's algorithm. But we have to guarantee that there is only one way of initialising $\mathbf{D}$. This is indeed the case, since

$$(m \triangledown n \ \ m \triangledown n) \times \mathbf{D} \ = \ (m \ n)$$
$$=$$
$$(1 \ 1) \times \mathbf{D} \ = \ (m/(m \triangledown n) \ \ n/(m \triangledown n)),$$

where $(m/(m \triangledown n) \ \ n/(m \triangledown n))$ can be seen as a positive rational number in so-called lowest-form representation. We know from section 4.4.2 that there is a bijection between finite products of the matrices $\mathbf{L}$ and $\mathbf{R}$ and the positive rationals. Therefore, $\mathbf{D}$ (which is a finite product of $\mathbf{L}$s and $\mathbf{R}$s) is uniquely defined (more specifically, it represents the path from the origin to the rational $\frac{n/(m \triangledown n)}{m/(m \triangledown n)}$ in the Stern-Eisenstein tree of rationals).

Now, since the alternative statement in the loop of Euclid's algorithm is deterministic ($y < x$ and $x < y$ are mutually exclusive), we can use the inversion rule for deterministic alternative statements together with the inversion rule for iterative statements. Recall, from section 3.6, that the inverse of a program of the form

$\{\ G_0 \vee G_1\ \}$

do $G_0 \ \rightarrow\ S_0\ \{\ C_0\ \}$

$\square\ \ G_1 \ \rightarrow\ S_1\ \{\ C_1\ \}$

od

$\{\ C_0 \vee C_1\ \}$,

is

$\{\ C_0 \vee C_1\ \}$

do $C_1 \ \rightarrow\ S_1^{-1}\ \{\ G_1\ \}$

$\square\ \ C_0 \ \rightarrow\ S_0^{-1}\ \{\ G_0\ \}$

od

$\{\ G_0 \vee G_1\ \}$.

(We require $\neg(C_0 \wedge C_1)$ and $\neg(G_0 \wedge G_1)$.) We now have to insert appropriate assertions in Euclid's algorithm so that the rules presented above can be used. Recall that, as explained in section 4.6.1, we want to ignore the initial values (in effect, this corresponds to ignoring the first row of table 4.1). This motivates moving the first step out of the loop body. Assuming that $n < m$, we can rewrite the algorithm as follows (note the new annotations and recall that $\mathbf{D} = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$):

$\{\ 0 < n\ <\ m\ \}$

$(x\ y)\,,\mathbf{D}\ :=\ (m{-}n\ \ n)\,,\mathbf{L}\ ;$

$\{$ **Invariant**: $\ (x\ y) \times \mathbf{D}\ =\ (m\ n)\ \}$

$\quad\{\ y < x\ \vee\ x < y\ \}$

$\quad$do $\ y < x\ \ \rightarrow\ \ (x\ y)\,,\mathbf{D}\ :=\ (x\ y) \times \mathbf{L}^{-1}\ ,\ \mathbf{L}{\times}\mathbf{D}\ \ \{\ a < c\ \}$

$\quad\square\ \ x < y\ \ \rightarrow\ \ (x\ y)\,,\mathbf{D}\ :=\ (x\ y) \times \mathbf{R}^{-1}\ ,\ \mathbf{R}{\times}\mathbf{D}\ \ \{\ c < a\ \}$

$\quad$od

$\quad\{\ a < c\ \vee\ c < a\ \}$

$\{\ (x\ y)\ =\ (m{\triangledown}n\ \ m{\triangledown}n)\qquad\wedge\qquad(m{\triangledown}n\ \ m{\triangledown}n) \times \mathbf{D}\ =\ (m\ n)\ \}$

For the rest of the argument, whenever we refer to Euclid's algorithm, the intended reference is to this algorithm. The removal of the first step out of the loop body forces $n < m$ and $1 < m$, but it allows us to include assertions after each assignment, making

the inversion of the loop body a straightforward application of the rules mentioned above. (The new assertions after the assignments follow from the facts that premultiplying a matrix by **L** corresponds to adding the first row to the second, and premultiplying a matrix by **R** corresponds to adding the second row to the first.) Note that we have indented the loop to stress that the new disjunctive assertions, $y < x \;\vee\; x < y$ and $a < c \;\vee\; c < a$, are, respectively, the loop's precondition and postcondition. Because the assignments in the loop body are easily inverted, the inverse of Euclid's algorithm becomes:

$\{\; 0 < n \;<\; m \;\}$

$(x \;\; y) \;:=\; (m \triangledown n \;\;\; m \triangledown n)\,;$

initialise **D** such that $(m \triangledown n \;\;\; m \triangledown n) \times \mathbf{D} \;=\; (m \;\; n)\,;$

$\{$ **Invariant**: $\;(x \;\; y) \times \mathbf{D} \;\;=\;\; (m \;\; n) \;\}$

$\quad\{\; a < c \;\vee\; c < a \;\}$

$\quad$ do $\;a < c \;\;\rightarrow\;\; (x \;\; y)\,, \mathbf{D} \;:=\; (x \;\; y) \times \mathbf{L}\;,\; \mathbf{L}^{-1} \times \mathbf{D} \;\;\{\, y < x \,\}$

$\quad\Box\;\; c < a \;\;\rightarrow\;\; (x \;\; y)\,, \mathbf{D} \;:=\; (x \;\; y) \times \mathbf{R}\;,\; \mathbf{R}^{-1} \times \mathbf{D} \;\;\{\, x < y \,\}$

$\quad$ od

$\quad\{\; y < x \;\vee\; x < y \;\}$

$\{\; (x \;\; y) \;=\; (m - n \;\;\; n) \wedge \mathbf{D} = \mathbf{L} \;\}$

Comparing the two algorithms, we see that the assignments to $(x \;\; y)$ and to $(a \;\; c)$ are interchanged: in the original algorithm we have

$$\begin{array}{lll} & y < x & \rightarrow \;\; (x \;\; y) \;:=\; (x - y \;\;\; y) \\ \Box & x < y & \rightarrow \;\; (x \;\; y) \;:=\; (x \;\;\; y - x)\;, \end{array}$$

and in the inverted algorithm we have

$$\begin{array}{lll} & a < c & \rightarrow \;\; (a \;\; c) \;:=\; (a \;\;\; c - a) \\ \Box & c < a & \rightarrow \;\; (a \;\; c) \;:=\; (a - c \;\;\; c)\;. \end{array}$$

(We leave the reader to check the matrix arithmetic.) In other words, the inverse of Euclid's algorithm is Euclid's algorithm itself, but on different variables: the inverted version computes the greatest common divisor using the variables a and c. This means that to make the sequences of successive values of the vectors $(x \;\; y)$ and $(a \;\; c)$ the reverse of each other, we only need to guarantee that the initial value of $(x \;\; y)$ in the

non-inverted algorithm is the same as the initial value of $(a\ c)$ in the inverted one. In other words, we need to guarantee that in Euclid's algorithm, the initial value of $(x\ y)$ is the same as the final value of $(a\ c)$.

The initial assignments of the inverted algorithm may seem strange at first sight, but the important fact to retain is that if we compose both algorithms, the program state remains unchanged. The inversion of the algorithm serves only as a formal proof that the process applied to $(x\ y)$ in one direction is the same as the one applied to $(a\ c)$ in the opposite direction. In the remainder of our investigation, we base our discussion on Euclid's algorithm, i.e., on the non-inverted version.

4.6.3 Reversed sequences of vectors

Given the result of the previous section, saying that the sequences of vectors $(x\ y)$ and $(a\ c)$ are reversed is equivalent to saying that the initial value of $(a\ c)$ is equal to the final value of $(x\ y)$ and the initial value of $(x\ y)$ is equal to the final value of $(a\ c)$.

Looking at the algorithm, we see that the initial value of $(a\ c)$ is $(1\ 1)$ and the final value of $(x\ y)$ is $(m \triangledown n\ \ m \triangledown n)$. So, for the sequences to be reversed, $m \triangledown n$ has to be 1, i.e., m and n have to be coprime. We thus assume henceforth that $m \triangledown n = 1$.

Also, the initial value of $(x\ y)$ is $(m-n\ \ n)$. So, because $m \triangledown n = 1$, we have the following equality:

"The sequences are reversed"

$=$

"The final value of $(a\ c)$ is $(m-n\ \ n)$" .

We can rewrite this equality in terms of matrix $\mathbf{D}$:

"The sequences are reversed"

$=$

"The final value of $\mathbf{D}$ is $\begin{pmatrix} m-n & b \\ n & d \end{pmatrix}$ for some b and d" .

Now, because $\mathbf{D}$ is the product of matrices whose determinant equals 1, its determinant also equals 1; this allows us to calculate b and d:

$det.\mathbf{D} = 1$

$=$ $\quad \{ \quad \mathbf{D}$ has the shape $\begin{pmatrix} m-n & b \\ n & d \end{pmatrix} \quad \}$

$$
\begin{array}{ll}
 & (m-n)\times d - n\times b \;=\; 1 \\
= & \quad \{ \quad \text{arithmetic} \quad \} \\
 & m\times d - n\times(d+b) \;=\; 1 \\
= & \quad \{ \quad \text{we have assumed that } m \bigtriangledown n = 1\text{, so, on termination,} \\
 & \qquad \text{the invariant states that } (1\ 1)\times \mathbf{D} \;=\; (m\ \ n)\text{;} \\
 & \qquad \text{this means that } n = b+d \quad \} \\
 & m\times d = n^2+1 \\
= & \quad \{ \quad 0 < m \quad \} \\
 & d = \dfrac{n^2+1}{m} \quad .
\end{array}
$$

The value of b is simply $n-d$, since on termination we have $n = b+d$ (it follows from the invariant). Because $\mathbf{D}$ is a matrix of integer values, d has to be an integer, and so, a necessary condition is that $m \backslash (n^2+1)$, that is, $n^2 \cong -1 \pmod{m}$. We can thus conclude that

$$n^2 \;\cong\; -1 \pmod{m} \;\Leftarrow\; \text{“The sequences are reversed”} \quad .$$

A question that now arises is whether $n^2 \cong -1 \pmod{m}$ is a sufficient condition for the sequences to be reversed. That is, can we prove

(4.6.2) “The final value of $\mathbf{D}$ is $\begin{pmatrix} m-n & n-{}^{(n^2+1)}/_{m} \\ n & {}^{(n^2+1)}/_{m} \end{pmatrix}$” $\quad\Leftarrow\quad n^2 \cong -1 \pmod{m}$?

Using the assumption that $\mathbf{D} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, we can simplify (4.6.2) to:

(4.6.3) “The final value of c is n” $\quad\Leftarrow\quad n^2 \;\cong\; -1 \pmod{m}$,

since c uniquely determines all the other entries (recall that $m = a+c$, $n = b+d$ and $det.\mathbf{D} = 1$). To prove (4.6.3), we first show that $n \cong c \pmod{m}$ follows from $n^2 \cong -1 \pmod{m}$ and then we use the range of n and c to conclude that $n = c$. The following lemma is used to prove that $n \cong c \pmod{m}$.

Lemma 4.6.4 For all integers m, n, and c, the following holds:

$$n \cong c \pmod{m} \;\Leftarrow\; n^2 \;\cong\; -1 \pmod{m} \;\wedge\; n\times c \;\cong\; -1 \pmod{m} \quad .$$

Proof Using the fact that, for all integers a, b, and c, the following law on congruences holds

(4.6.5) $a-c \;\cong\; b-d \pmod{m} \;\Leftarrow\; a \cong b \pmod{m} \wedge c \cong d \pmod{m}$,

we can prove the lemma as follows:

$$
\begin{array}{ll}
& n \cong c \pmod{m} \\
= & \quad \{ \quad \text{arithmetic} \quad \} \\
& n - c \cong 0 \pmod{m} \\
\Leftarrow & \quad \{ \quad m \bigtriangledown n = 1 \text{ and Euclid's lemma; see below for details} \quad \} \\
& n \times (n - c) \cong 0 \pmod{m} \\
= & \quad \{ \quad \text{arithmetic} \quad \} \\
& n^2 - n \times c \cong 0 \pmod{m} \\
= & \quad \{ \quad n^2 \cong -1 \pmod{m} \text{ and } n \times c \cong -1 \pmod{m} \text{ and (4.6.5)} \quad \} \\
& \mathsf{true} \ .
\end{array}
$$

In the second step we can safely assume that $m \bigtriangledown n = 1$, since it follows from the congruence $n^2 \cong -1 \pmod{m}$. A short proof of this fact is:

$$
\begin{array}{ll}
& n^2 \cong -1 \pmod{m} \\
= & \quad \{ \quad \text{definition} \quad \} \\
& \langle \exists q :: \ n^2 + 1 = q \times m \rangle \\
= & \quad \{ \quad \text{arithmetic} \quad \} \\
& \langle \exists q :: \ 1 = q \times m - n \times n \rangle \\
\Rightarrow & \quad \{ \quad (m \bigtriangledown n) \backslash (q \times m - n \times n), \text{ so } (m \bigtriangledown n) \backslash 1; \\
& \qquad \text{division is anti-symmetric} \quad \} \\
& m \bigtriangledown n = 1 \ .
\end{array}
$$

Also, Euclid's lemma states that for all integers a, b, and c:

$$a \backslash c \quad \Leftarrow \quad a \backslash b \times c \ \wedge \ a \bigtriangledown b = 1 \quad .$$

We prove Euclid's lemma in page 82.
□

Now, if, on termination, we have that $n \times c \cong -1 \pmod{m}$, we can use lemma 4.6.4 to conclude that, on termination, we also have that $n \cong c \pmod{m}$ follows from $n^2 \cong -1 \pmod{m}$. Recall that an invariant of the algorithm is

$$(x \ \ y) \times \mathbf{D} \quad = \quad (m \ \ n) \quad = \quad (x \times a + y \times c \quad x \times b + y \times d) \quad .$$

Because the determinant of $\mathbf{D}$ equals 1, the inverse of $\mathbf{D}$ is $\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$, making the following property also invariant:

$$(4.6.6)\quad (x\ y) \;=\; (m\ n)\times \mathbf{D}^{-1} \;=\; (m{\times}d - n{\times}c \quad a{\times}n - b{\times}m)\ .$$

It follows that on termination, when $(x\ y) = (1\ 1)$, we have that $n{\times}c \cong -1 \pmod m$, as the following calculation shows:

$$\begin{array}{ll} & n{\times}c \cong -1 \pmod m \\ = & \quad \{\quad \text{definition} \quad \} \\ & m \backslash (n{\times}c + 1) \\ \Leftarrow & \quad \{\quad \text{division properties} \quad \} \\ & m{\times}d \;=\; n{\times}c + 1 \\ = & \quad \{\quad \text{arithmetic} \quad \} \\ & m{\times}d - n{\times}c \;=\; 1 \\ = & \quad \{\quad \text{invariant (4.6.6), } (x\ y) = (1\ 1) \text{ on termination} \quad \} \\ & \mathsf{true}\ . \end{array}$$

By lemma 4.6.4, we deduce that on termination $n \cong c \pmod m$ follows from $n^2 \cong -1 \pmod m$. Finally, because $0 < a$ and $m = a + c$ we have that $0 < c < m$; this allows us to conclude that $n = c$:

$$\begin{array}{ll} & n \cong c \pmod m \\ = & \quad \{\quad \text{definition} \quad \} \\ & m \backslash (n - c) \\ = & \quad \{\quad 0 < n < m \text{ and } 0 < c < m \text{ imply that } -m < n - c < m; \\ & \qquad \text{the only multiple of } m \text{ in that range is } 0 \quad \} \\ & n - c = 0 \\ = & \quad \{\quad \text{arithmetic} \quad \} \\ & n = c\ . \end{array}$$

The conclusion is that $n^2 \cong -1 \pmod m$ is also a sufficient condition for the sequences to be reversed, leading to the equality:

"The sequences are reversed"

=

$n^2 \cong -1 \pmod{m}$.

To summarise, in the following algorithm

$\{\ 0 < n < m\ \}$

$(x\ y)\,,\mathbf{D}\ :=\ (m{-}n\ \ n)\,,\mathbf{L}\ ;$

$\{$ **Invariant**: $(m\ n) = (x\ y) \times \mathbf{D} = (x{\times}a + y{\times}c\ \ \ x{\times}b + y{\times}d)$

$\quad \wedge \qquad (m\ n) \times \mathbf{D}^{-1} = (x\ y) = (m{\times}d - n{\times}c\ \ \ a{\times}n - b{\times}m)\ \}$

do $y < x\ \ \rightarrow\ \ (x\ y)\,,\mathbf{D}\ :=\ (x\ y) \times \mathbf{L}^{-1}\ ,\ \mathbf{L}{\times}\mathbf{D}\ \ \{\ a < c\ \}$

$\Box\ \ x < y\ \ \rightarrow\ \ (x\ y)\,,\mathbf{D}\ :=\ (x\ y) \times \mathbf{R}^{-1}\ ,\ \mathbf{R}{\times}\mathbf{D}\ \ \{\ c < a\ \}$

od

$\{\ (x\ y)\ =\ (1\ 1) \qquad \wedge \qquad (m\ n)\ =\ (1\ 1) \times \mathbf{D}\ \},$

the sequences of vectors $(x\ y)$ and $(a\ c)$ are reverses of each other exactly when $n^2 \cong -1 \pmod{m}$.

4.6.4 Length of the sequence of vectors

We now have to prove that the final value of matrix $\mathbf{D}$ is decomposed into an odd-length product of the matrices $\mathbf{L}$ and $\mathbf{R}$. However, because $\mathbf{D}$ is initially $\mathbf{L}$ and because it is iteratively premultiplied, $\mathbf{D} = \mathbf{M}{\times}\mathbf{L}$ for some $\mathbf{M}$. So we can alternatively prove that $\mathbf{M}$ is decomposed into an even-length product of the matrices $\mathbf{L}$ and $\mathbf{R}$. Observing that

$$\mathbf{M} = \mathbf{D}{\times}\mathbf{L}^{-1} = \begin{pmatrix} m-(2{\times}n - {}^{(n^2+1)}/_{m}) & n-{}^{(n^2+1)}/_{m} \\ n-{}^{(n^2+1)}/_{m} & {}^{(n^2+1)}/_{m} \end{pmatrix} \quad ,$$

we see that $\mathbf{M}$ has the top-right and bottom-left corners equal, which means that $\mathbf{M} = \mathbf{M}^T$ ($\mathbf{M}$ equals the transpose of $\mathbf{M}$). We also know that $\mathbf{R} = \mathbf{L}^T$ and $\mathbf{L} = \mathbf{R}^T$.

There are also two functions from finite products of $\mathbf{L}$ and $\mathbf{R}$ to naturals, #$\mathbf{L}$ and #$\mathbf{R}$, that give, respectively, the number of $\mathbf{L}$s and the number of $\mathbf{R}$s in the decomposition of their argument[15]. Now, a fundamental property is that #$\mathbf{L}.\mathbf{M} =$ #$\mathbf{R}.\mathbf{M}^T$, whenever $\mathbf{M}$ is

[15]Note that, given that we can easily provide algorithms that compute them, functions length, #$\mathbf{L}$, and #$\mathbf{R}$ are well-defined. As proved in section 4.4.2, there is a bijection between finite products of matrices $\mathbf{L}$ and $\mathbf{R}$, and binary strings made of the symbols L and R; defining these functions in the realm of strings is easy.

a product of **L**s and **R**s. This fundamental property means that the number of **L**s in the decomposition of $\mathbf{M}$ equals the numbers of **R**s in the decomposition of $\mathbf{M}^T$, which is easy to see because $\mathbf{R} = \mathbf{L}^T$ and $\mathbf{L} = \mathbf{R}^T$. Using these observations, a simple calculation showing that the length of $\mathbf{M}$ is an even number is:

$$
\begin{array}{ll}
 & \mathsf{length}.\mathbf{M} \\
= & \quad \{ \quad \mathbf{M} \text{ is a product of } \mathbf{L}\text{s and } \mathbf{R}\text{s} \quad \} \\
 & \#\mathbf{L}.\mathbf{M} + \#\mathbf{R}.\mathbf{M} \\
= & \quad \{ \quad \#\mathbf{L}.\mathbf{M} = \#\mathbf{R}.\mathbf{M}^T \quad \} \\
 & \#\mathbf{R}.\mathbf{M}^T + \#\mathbf{R}.\mathbf{M} \\
= & \quad \{ \quad \mathbf{M}^T = \mathbf{M} \quad \} \\
 & 2 \times \#\mathbf{R}.\mathbf{M} \ .
\end{array}
$$

Hence, the length of $\mathbf{M}$ is an even number. Subsequently, the length of the final value of $\mathbf{D}$ is odd.

4.6.5 Sum of two positive squares

In the above sections we have proved the following theorem:

Theorem 4.6.7 A number m at least 2 can be written as the sum of two positive squares if there is a number n such that $0 < n < m$ and $n^2 \cong -1 \pmod{m}$.
□

The argument we provide is constructive because we show how to use Euclid's algorithm to represent a number as the sum of two positive squares. Indeed we can extend Euclid's algorithm so that it expresses a given number m as the sum of two positive squares:

$\{ \ 1 < m \ \wedge \ \langle \exists n \ : \ 0 < n \ < \ m \ : \ n^2 \cong -1 \pmod{m} \rangle \ \}$

• Find a number n such that $0 < n \ < \ m$ and $n^2 \cong -1 \pmod{m}$;

$\{ \ 0 < n \ < \ m \ \wedge \ n^2 \cong -1 \pmod{m} \ \}$

$(x \ y) , \mathbf{D} \ := \ (m{-}n \ \ n) , \mathbf{L} \ ;$

$\{$ **Invariant**: $(x \ y) \times \mathbf{D} = (m \ n) = (x{\times}a + y{\times}c \quad x{\times}b + y{\times}d) \ \}$

$\mathsf{do} \ \ (x \ y) \neq (a \ c) \ \rightarrow$

$$\begin{array}{rcl} y<x & \rightarrow & (x\ y)\,,\mathbf{D} := (x\ y)\times\mathbf{L}^{-1}\ ,\ \mathbf{L}\times\mathbf{D} \\ \square\ x<y & \rightarrow & (x\ y)\,,\mathbf{D} := (x\ y)\times\mathbf{R}^{-1}\ ,\ \mathbf{R}\times\mathbf{D} \end{array}$$

od

$$\{\ (x\ y)=(a\ c)\ \wedge\ m=x^2+y^2=a^2+c^2\ \}$$

Theorem 4.6.7 is more general than Girard's result: while Girard's theorem is only on odd prime numbers, theorem 4.6.7 concerns all positive integers at least 2. As an example, we can say that the number 10 is expressible as the sum of two positive squares, since $3^2\cong -1 \pmod{10}$ (and, in fact, we have that $10=3^2+1^2$). Moreover, given the following lemma (see [AZ04, p. 17, Lemma 1]), Girard's result is an immediate corollary of theorem 4.6.7.

Lemma 4.6.8 For primes $p = 4k+1$ the equation $s^2\cong -1 \pmod{p}$ has two solutions $s\in\{1\,..\,p-1\}$, for $p=2$ there is only one such solution, while for primes of the form $p = 4k+3$ there is no solution.
□

Although we believe that theorem 4.6.7 may be known by some number-theorists, we have not found it in the literature.

Please note that developing an efficient algorithm to find a number n such that $0<n<m$ and $n^2 \cong -1 \pmod{m}$ is not trivial. For more details on this topic, we recommend [Wag90] and [BW08], where the authors discuss different algorithms that can be used to find such a number n. Finally, the algorithm shown above can be generalised. In a private communication, Wagon told us that the method of using Euclid's algorithm to write a number as a sum of two squares (or, more generally, as $a^2+g\times c^2$) is known as the Smith-Cornacchia algorithm (he referred us to [BW08] and [Cor08]). Also, in [HMW90], Hardy, Muskat, and Williams show a more general algorithm for solving $m = f\times a^2+g\times c^2$ in coprime integers a and c. At the moment, we do not know how to adapt our algorithm to solve the more general problem. Recall that we have started our argument by observing that if, at any point in the execution of the algorithm, $(x\ y)$ equals $(a\ c)$, it follows from the invariant

$$(m\ n) \quad = \quad (x\ y)\times\mathbf{D} \quad = \quad (x\times a+y\times c \quad x\times b+y\times d)$$

that m can be written as a sum of two positive squares. To solve the general problem, we have to investigate when it is possible to have, at any point in the execution of the algorithm, $a\backslash x$ and $c\backslash y$. If this happens, that is, if there are two integers f and g such

that $x = f \times a$ and $y = g \times c$, it follows from the invariant that $m = f \times a^2 + g \times c^2$:

$$(m \;\; n) \quad = \quad (f \times a \;\; g \times c) \times \mathbf{D} \quad = \quad (f \times a^2 + g \times c^2 \;\; f \times a \times b + g \times c \times d) \quad .$$

4.6.6 Discussion

This section shows a new and constructive proof of the two-squares theorem based on a somewhat unusual, but very effective, way of rewriting the so-called extended Euclid's algorithm. As mentioned in the introduction, the use of Euclid's algorithm to prove the theorem is not new: Brillhart [Bri72] and Wagon [Wag90] have used it to *verify* the theorem. Effectively, given the close relationship between Euclid's algorithm and continued fractions, we can say that Serret [Ser48] and Hermite [Her48] were the first to provide the germ of the essential idea presented here (in fact, Brillhart's note is described as an improvement on Hermite's method: in using Euclid's algorithm, Brillhart avoids the calculation of the convergents arising in the continued fractions).

The novel contribution of this section is the use of the algorithm to *investigate* which numbers can be written as the sum of two positive squares. The precise formulation of the problem as an algorithmic problem is the key, since it allows us to use algorithmic techniques and to avoid guessing. The notion of invariance, in particular, plays a central role in our development: it is used initially to observe that Euclid's algorithm can actually be used to represent a given number as a sum of two positive squares, and then it is used throughout the argument to prove relevant properties. Also, section 4.6.2 is an example of how the use of program inversion can make our arguments more precise.

4.7 Conclusion

In our view, much of mathematics is inherently algorithmic; it is also clear that, in the modern age, algorithmic problem solving is just as important, if not much more so, than in the 19th century. Somehow, however, mathematical education in the 20th century lost sight of its algorithmic roots. We hope to have exemplified in this chapter how a fresh approach to introductory number theory that focuses on the algorithmic content of the theory can combine practicality with mathematical elegance. By continuing this endeavour we believe that the teaching of mathematics can be enriched and given new vigour.

We have to admit, however, that this chapter lacks some important material that should be taught in a module on elementary number theory. In particular, we do not discuss

fundamental topics such as primality or number theoretic functions (e.g., functions τ and σ). Nevertheless, our goal was never to make of this chapter a complete textbook! Instead, we wanted to show how we could use algorithmic principles and techniques to rewrite existing material, and, more importantly, we wanted to derive new results. We have achieved both goals and we believe we have done it in a practical and elegant way.

4.8 Appendix: historical remarks on the trees of rationals

One of the primary novel results of this chapter is the construction given in section 4.4.2 of an algorithm to enumerate the rationals in Stern-Brocot order. Apart from minor differences, that section was submitted in April 2007 to the American Mathematical Monthly; it was rejected in November 2007 on the grounds that it was not of sufficient interest to readers of the Monthly. One (of two referees) did, however, recommend publication. The referee made the following general comment.

> Each of the two trees of rationals—the Stern-Brocot tree and the Calkin-Wilf tree—has some history. Since this paper now gives the definitive link between these trees, I encourage the authors, perhaps in their Discussion section, to also give the definitive histories of these trees, something in the same spirit as the Remarks at the end of the Calkin and Wilf paper.

Since the publication of [BF08], we have succeeded in obtaining copies of the original papers and it is indeed interesting to briefly review the papers. But we do not claim to provide "definitive histories of these trees" — that is a task for a historian of mathematics.

Section 4.8.1 is about the paper [Ste58] published in 1858 by Stern. The surprising fact that emerges from the review is that the so-called "Calkin-Wilf" tree of rationals, and not just the "Stern-Brocot" tree, is studied in detail in his paper. Moreover, of the two structures, the "Calkin-Wilf" tree is more readily recognised; the "Stern-Brocot" tree requires rather more understanding to identify. Brocot's paper [Bro61], which we review in section 4.8.2, is interesting because it illustrates how 19th century mathematics was driven by practical, algorithmic problems. (For additional historical remarks, see also [Hay00].)

As mentioned before, the review of Stern's paper was written by Roland Backhouse, since the author of this dissertation does not read any German.

4.8.1 Stern's paper

Earlier we have commented that the structure that has recently been referred to as the "Calkin-Wilf" tree was documented by Stern [Ste58] in 1858. In this section we review those sections of Stern's paper that are relevant to our own.

The Eisenstein array

Stern's paper is a detailed study of what has now become known as the "Eisenstein array" of numbers (see, for example, [Slo, sequence A064881]). (Stern's paper cites two papers written by the more famous mathematician Gotthold Eisenstein; we have not read these papers.) Given two natural numbers m and n, Stern describes a process (which he attributes to Eisenstein) of generating an infinite sequence of rows of numbers. The *zeroth* row in the sequence ("nullte Entwickelungsreihe") is the given pair of numbers:

$$m \quad n \quad .$$

Subsequent rows are obtained by inserting between every pair of numbers the sum of the numbers. Thus the *first* row is

$$m \quad m+n \quad n$$

and the *second* row is

$$m \quad 2\times m+n \quad m+n \quad m+2\times n \quad n \quad .$$

The process of constructing such rows is repeated indefinitely. The sequence of numbers obtained by concatenating the individual rows in order is what is now called the *Eisenstein array* and denoted by $Ei(m,n)$ (see, for example, [Slo, sequence A064881]) . Stern refers to each occurrence of a number in rows other than the zeroth row as either a *sum element* ("Summenglied") or a *source element* ("Stammglied"). The sum elements are the newly added numbers. For example, in the first row the number $m+n$ is a sum element; in the second row the number $m+n$ is a source element.

The Eisenstein-Stern tree of rationals

A central element of Stern's analysis of the Eisenstein array is the consideration of subsequences of numbers in individual rows. He calls these *groups* ("Gruppen") and he records the properties of pairs of consecutive numbers (groups of size two — "zweigliedrige Gruppen") and triples of consecutive numbers (groups of size three — "dreigliedrige Gruppen").

In sections 5 thru 8 of his paper, Stern studies $Ei(1,1)$, the Eisenstein array that begins with the pair $(1,1)$. He proves that all pairs of consecutive numbers in a given row are coprime and every pair of coprime numbers appears exactly once as such a pair of consecutive numbers. He does not use the word "tree" —tree structures are most probably an invention of modern computing science— and he does not refer to "rational numbers" —he refers instead to relatively prime numbers ("relatieve Primzahlen")— but there is no doubt that, apart from the change in terminology, he describes the tree of rationals that in recent years has been referred to as the "Calkin-Wilf" tree of rationals. It is for this reason that we believe it is misleading to use the name "Calkin-Wilf tree" and prefer to use the name "Eisenstein-Stern tree". Figure 4.4 shows the first four rows of $Ei(1,1)$ and figure 4.5 shows all pairs of consecutive numbers for each of the four rows. The pairs have been arranged so that the correspondence between figure 4.2 and figure 4.5 is clear.

$$
\begin{array}{ccccccccccccccc}
 & & & & & & 1 & & 1 & & & & & & \\
 & & & & & 1 & & 2 & & 1 & & & & & \\
 & & & & 1 & & 3 & & 2 & & 3 & & 1 & & \\
1 & & 4 & & 3 & & 5 & & 2 & & 5 & & 3 & & 4 & & 1
\end{array}
$$

Figure 4.4: First four rows of $Ei(1,1)$

$$
\begin{array}{ccccccccccccccc}
 & & & & & & (1,1) & & & & & & & & \\
 & & & & & (1,2) & & & & & (2,1) & & & & \\
 & & & & (1,3) & & & & (3,2) & & & & (2,3) & & & & (3,1) \\
(1,4) & & (4,3) & & (3,5) & & (5,2) & & (2,5) & & (5,3) & & (3,4) & & (4,1)
\end{array}
$$

Figure 4.5: Pairs of consecutive numbers in the first four rows of $Ei(1,1)$

Other sections of Stern's paper record additional properties of the tree, which we do not discuss here. For example, Stern discusses how often each number appears as a sum number.

The Stern-Brocot tree of rationals

Identification of the so-called Stern-Brocot tree of rationals in Stern's paper is more demanding. Recall the process of constructing a sequence of rows of numbers from a given pair of numbers m and n. It is clear that every number is a linear combination of m and n. Stern studies the *coefficients* ("Coefficienten"), i.e. the pair of multiplicative factors of m and n, defined by the linear combination. Figure 4.6 displays the coefficients in a way that allows direct comparison with the Stern-Brocot tree of rationals (figure 4.3). (The reader may also wish to compare figure 4.6 with Graham, Knuth and Patashnik's depiction of the tree [GKP94, p. 117].)

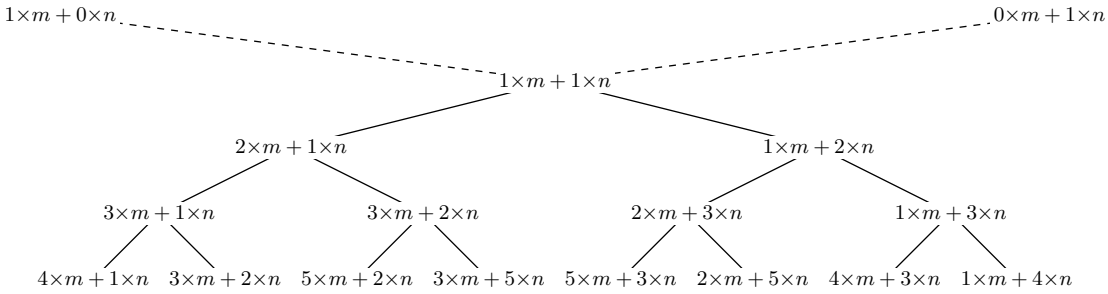


Figure 4.6: Tree of "coefficients" of $Ei(m,n)$

The numbers at the top-left and top-right of figure 4.6 are the numbers m and n written as $1\times m + 0\times n$ and $0\times m + 1\times n$, respectively, in order to make the coefficients clear. This, we recall, is the zeroth row in Stern's structure.

In the subsequent levels of the tree, only the sum elements are displayed. The correspondence between figure 4.6 and figure 4.3 should be easy to see; the number $k\times m + l\times n$ in figure 4.6 is displayed as the rational $\frac{l}{k}$ in figure 4.3. The "fundamental fact" (4.31) in [GKP94] is observed by Stern [Ste58, equation (8), p.207] and used immediately to infer that coefficients are relatively prime. In section 15 of his paper, Stern uses the (already proven) fact that the Eisenstein-Stern tree is a tree of (all) rationals to deduce that the Stern-Brocot tree is also a tree of rationals.

Newman's algorithm

An interesting question is whether Stern also documents the algorithm currently attributed to Moshe Newman for enumerating the elements of the Eisenstein array. This is a question we found difficult to answer because of our limited understanding of German. However, the answer would appear to be: almost, but not quite!

As remarked earlier, Stern documents a number of properties of groups of numbers in rows of the Eisenstein array, in particular groups of size three. Of course, a group of

size three comprises two groups of size two. Since groups of size two in the Eisenstein array correspond to rationals in the Eisenstein-Stern tree, by studying groups of size three Stern is effectively studying consecutive rationals in the Eisenstein-Stern tree of rationals.

It is important to note that Stern's focus is the sequence of *rows* of numbers (in modern terminology, the tree of numbers) as opposed to the (flattened) sequence of numbers defined by $Ei(m,n)$ — significantly, the last number in one row and the first number in the next row do not form a "group" according to Stern's definition. This means that, so far as we have been able to determine, he nowhere considers a triple of numbers that crosses a row boundary.

Newman's algorithm (in the form we use in section 4.4.2) predicts that each triple of numbers in a given row of $Ei(1,1)$ has the form

$$a \quad b \quad (2\left\lfloor \frac{a}{b} \right\rfloor + 1) \times b - a$$

(Variable names have been chosen to facilitate comparison with Stern's paper.) It follows immediately that the sum of the two outer elements of the triple is divisible by the middle element (that is, $a + ((2\left\lfloor \frac{a}{b} \right\rfloor + 1) \times b - a)$ is divisible by b); this fact is observed by Stern (for triples in a given row) in section 4 of his paper. Importantly for what follows, Stern observes that the property holds for $Ei(m,n)$ for arbitrary natural numbers m and n, and not just $Ei(1,1)$. Stern observes further [Ste58, (4) p.198] that each triple in $Ei(m,n)$ has the form

(4.8.1) $a \quad b \quad (2t+1) \times b - a$

for some number t. Stern identifies t as the number of rows preceding the current row in which the number b occurs as a sum element. (In particular, if b is a sum element then t equals 0.) Stern shows how to calculate t from the position of b in the row — effectively by expressing the position as a binary numeral. (Note that "t" is the variable name used in Stern's paper; it has the same role as the variable "j" in our derivation of the algorithm in section 4.4.2.)

So far as we have been able to determine, Stern does not explicitly remark that t equals $\left\lfloor \frac{a}{b} \right\rfloor$ in the case of $Ei(1,1)$, but he does so implicitly in section 10 where he relates the the continued fraction representation of $\frac{a}{b}$ to the row number in which the pair (a,b) occurs. He does not appear to suggest a similar method for computing t in the general case of enumerating $Ei(m,n)$. However, it is straightforward to combine our derivation of Newman's algorithm with Stern's theorems to obtain an algorithm to enumerate the elements of $Ei(m,n)$ for arbitrary natural numbers m and n. Interested readers may consult our website [BF10] where several implementations are discussed.

As stated at the beginning of this section, the conclusion is that Stern almost derives Newman's algorithm, but not quite. On the other hand, because his analysis is of the general case $Ei(m,n)$ as opposed to $Ei(1,1)$, his results are more general.

Stern-Brocot enumeration

We now turn to the question whether Stern also gives an algorithm for enumerating the rationals in Stern-Brocot order.

To this end, we observe that the form (4.8.1) extends to the coefficients of each element of $Ei(M,N)$, and hence to the elements of the Stern-Brocot tree. Specifically, triples in $Ei(M,N)$ have the form

$$n_0M+m_0N \qquad n_1M+m_1N \qquad ((2k+1)n_1-n_0)M+((2k+1)m_1-m_0)N$$

It is easy to exploit this formula directly to get an enumeration of the rationals in Stern-Brocot order, just as we did above to obtain an enumeration of $Ei(M,N)$. Just recall that the Stern-Brocot rationals are given by the coefficients of the sum elements, and the sum elements are the odd-numbered elements in the rows of $Ei(M,N)$ (where numbering starts from zero). The algorithm so obtained is the one we derived in section 4.4.2.

In this sense, Stern does indeed provide an algorithm for enumerating the rationals in Stern-Brocot order, albeit implicitly. However, as with Newman's algorithm, he fails to observe the concise formula for the value of the variable k. Also, a major methodological difference is our exploitation of the concision and precision afforded by matrix algebra. Given the state of development of matrix algebra in 1858, Stern cannot be criticised for not doing the same.

Finally, we remark that Stern returns to the properties of triples in section 19 of his paper. Unfortunately, we have been unable to fully understand this section.

4.8.2 Brocot, the watchmaker

Achille Brocot was a famous French watchmaker who, some years before the publication of his paper [Bro61], had to fix some pendulums used for astronomical measurements. However, the device was incomplete and he did not know how to compute the number of teeth of cogs that were missing. He was unable to find any literature helpful to the solution of the problem, so, after some experiments, he devised a method to compute the numbers. In his paper, Brocot illustrates his method with the following example:

> A shaft turns once in 23 minutes. We want suitable cogs so that another shaft completes a revolution in 3 hours and 11 minutes, that is 191 minutes.

The ratio between both speeds is $\frac{191}{23}$, so we can clearly choose a cog with 191 teeth, and another one with 23 teeth. But, as Brocot wrote, it was not possible, at that time, to create cogs with so many teeth. And because 191 and 23 are coprime, cogs with fewer teeth can only approximate the true ratio.

Brocot's contribution was a method to compute approximations to the true ratios (hence the title of his paper, "Calculus of cogs by approximation"). He begins by observing that $\frac{191}{23}$ must be between the ratios $\frac{8}{1}$ and $\frac{9}{1}$. If we choose the ratio $\frac{8}{1}$, the error is -7 since $8\times23 = 1\times191 - 7$. This means that if we choose this ratio, the slower cog completes its revolution seven minutes early, i.e., after 8×23 minutes. On the other hand, if we choose the ratio $\frac{9}{1}$, the error is 16 since $9\times23 = 1\times191 + 16$, meaning that the slower cog completes its revolution sixteen minutes late, i.e., after 9×23 minutes.

Accordingly, Brocot writes two rows:

8	1	-7
9	1	+16

His method consists in iteratively forming a new row, by adding the numbers in all three columns of the rows that produce the smallest error. Initially, we only have two rows, so we add the numbers in the three columns and we write the row of sums in the middle.

8	1	-7
17	2	+9
9	1	+16

(If we choose the ratio $\frac{17}{2}$, the slower cog completes its revolution $\frac{9}{2}$ minutes later, since $\frac{17}{2} = \frac{191+\frac{9}{2}}{23}$.) Further approximations are constructed by adding a row adjacent to the row that minimises the error term. The process ends once we reach the error 0, which refers to the true ratio. The final state of the table is:

8	1	-7
33	4	-5
58	7	-3
83	10	-1
191	23	0
108	13	+1
25	3	+2
17	2	+9
9	1	+16

The conclusion is that the two closest approximations to $\frac{191}{23}$ are ratios of $\frac{83}{10}$ (which runs $\frac{1}{10}$ minutes faster) and $\frac{108}{13}$ (which runs $\frac{1}{13}$ minutes slower). We could continue this process, getting at each stage a closer approximation to $\frac{191}{23}$. In fact, Brocot refines the table shown above, in order to construct a multistage cog train (see [Bro61, p. 191]).

At each step in Brocot's process we add a new ratio $\frac{m+m'}{n+n'}$, which is usually called the mediant of $\frac{m}{n}$ and $\frac{m'}{n'}$. Similarly, each node in the Stern-Brocot tree is of the form $\frac{m+m'}{n+n'}$, where $\frac{m}{n}$ is the nearest ancestor above and to the left, and $\frac{m'}{n'}$ is the nearest ancestor above and to the right. (Consider, for example, the rational $\frac{4}{3}$ in figure 4.3. Its nearest ancestor above and to the left is $\frac{1}{1}$ and its nearest ancestor above and to the right is $\frac{3}{2}$.) Brocot's process can be used to construct the Stern-Brocot tree: first, create an array that contains initially the rationals $\frac{0}{1}$ and $\frac{1}{0}$; then, insert the rational $\frac{m+m'}{n+n'}$ between two adjacent fractions $\frac{m}{n}$ and $\frac{m'}{n'}$. In the first step we add only one rational to the array

$$\frac{0}{1}, \frac{1}{1}, \frac{1}{0} \quad ,$$

but in the second step we add two new rationals:

$$\frac{0}{1}, \frac{1}{2}, \frac{1}{1}, \frac{2}{1}, \frac{1}{0} \quad .$$

Generally, in the n^{th} step we add 2^{n-1} new rationals. Clearly, this array can be represented as an infinite binary tree, whose first four levels are represented in figure 4.3 (we omit the fractions $\frac{0}{1}$ and $\frac{1}{0}$).

The most interesting aspect to us of Brocot's paper is that it solves an *algorithmic* problem. Brocot was faced with the practical problem of how to approximate rational

numbers in order to construct clocks of satisfactory accuracy and his solution is undisputably an *algorithm*. Stern's paper is closer to a traditional mathematical paper but, even so, it is an in-depth study of an algorithm for generating rows of numbers of increasing length.

4.8.3 Conclusion

There can be no doubt that what has been dubbed in recent years the "Calkin-Wilf" tree of rationals is, in fact, a central topic in Stern's 1858 paper. Calkin and Wilf [CW00] admit that in Stern's paper "there is a structure that is essentially our tree of fractions" but add "in a different garb" and do not clarify what is meant by "a different garb". It is unfortunate that the misleading name has now become prevalent; in order to avoid further misinterpretations of historical fact, it would be desirable for Stern's paper to be translated into English.

We have not attempted to determine how the name "Stern-Brocot" tree came into existence. It has been very surprising to us how much easier it is to identify the Eisenstein-Stern tree in Stern's paper in comparison to identifying the Stern-Brocot tree.

CHAPTER 5

Supporting the Teaching of Algorithmic Problem Solving

What is teaching?
In my opinion, teaching is giving opportunity
to the students to discover things by themselves.
— GEORGE PÓLYA (1966)

5.1 Teaching scenarios

The teaching of any subject can only be effective if the teacher has access to abundant and sufficiently varied educational resources. That is why one of our goals is to develop educational material to support the teaching of algorithmic problem solving.

The material shown in previous chapters can indeed be used to teach some algorithmic principles and techniques. Chapter 4, for example, can be used to rewrite a course on elementary number theory. Nevertheless, and although the material shown contains educational remarks, we believe that the teaching of algorithmic problem solving is more effective if the teacher has access to detailed guidelines on how to solve and present specific algorithmic problems.

Towards that end, we propose the introduction of educational material in the form of *teaching scenarios*, which are fully worked out solutions to algorithmic problems together with detailed guidelines on the principles captured by the problem, how the problem is tackled, and how it is solved. In appendix I, we present a set of teaching scenarios that illustrate the principles discussed in this thesis. The scenarios are example-driven and they usually have a recreational flavour, making them especially

suitable for extra-curricular math clubs. Although they can be directly used by the students, they are primarily written for the teacher. Moreover, they are designed to promote self-discovery, since we believe that the success of teaching depends on the amount of discovery that is left for the students: if the teacher discloses all the information needed to solve a problem, students act only as spectators and become discouraged; if the teacher leaves all the work to the students, they may find the problem too difficult and become discouraged too. Scenarios are designed to maintain a balance between these two extremes.

In general, each scenario is divided into the following sections:

- **Brief description and goals** This section provides a summary of the scenario, allowing the teacher to determine if it is adequate for the students.
- **Problem statement** This section states the problem (or problems) discussed in the scenario.
- **Students should know** This section lists prerequisites that should be met by the students. The teacher can use it to determine if the scenario is adequate for the students.
- **Resolution** This section presents a possible solution for the problem in the style advocated in this thesis.
- **Notes for the teacher** In this section, the solution presented above is decomposed into its main parts and each part is discussed in more detail. To maintain the balance mentioned in the first paragraph, we also recommend how the teacher should present the material, including questions that the teacher should or should not ask, and important concepts that should be introduced.
- **Extensions and exercises** This section can be used for homework or project assignments. All the exercises are accompanied by their solutions.
- **Further reading** Recommended reading for the teacher and the students. It may include discussions and comparisons between conventional solutions and the solutions presented in the scenario.

Some of the problems and solutions shown in the scenarios are not new, but we capture them in a new and accessible way: as a catalogue of problems and solutions having a consistent format. The reader may notice that some of the scenarios have longer solutions than what is conventionally expected. The reason is that the scenarios are

method-oriented, rather than solution-oriented. For us, proving a theorem or solving a problem is only part of the goal; we believe it is more important to demystify mathematical invention and to make of algorithmic problem solving a teachable discipline.

It is important to note that, although scenarios are detailed guidelines with self-contained and complete solutions, we encourage the teachers to adapt them to their own teaching style or to improve them with more effective or elegant solutions.

5.2 How to create a teaching scenario

This section presents some guidelines for creating new teaching scenarios. It also serves to understand better why we have built the catalogue of scenarios the way we did. We discuss each section in turn, giving examples where appropriate.

5.2.1 Brief description and goals

This section provides a summary of the scenario, allowing the teacher to determine if it is adequate for the students. Ideally, it describes the type of problem being solved, its goal, and the principles and techniques that are used in the solution. For example, we have described scenario 5, "A Logical Race", as follows:

> This scenario shows how a calculational approach to logic leads to a concise solution of a type of logic puzzle that is based on unique existential quantifications. It can be used to introduce Boolean inequivalence ($\not\equiv$), to practise formal modelling, and to illustrate how distributivity can be used to simplify mathematical arguments. The puzzle, which we have found in [Hon98, p. 17], is about deducing a conclusion based on the statements of three people. We also show (in the exercises) how we can generalise this type of logic puzzle.

5.2.2 Problem statement

This section states the problem (or problems) discussed in the scenario. The problem statement should be concise, clear and unambiguous. Consider, for example, the problem statement that we use in scenario 1 ("Exploring Algebraic Symmetries"):

> Prove that the product of four consecutive *positive* natural numbers cannot be the square of an integer number.

We found this problem in [Zei06, p. 4], where the formulation omits the adjective *positive*. However, for us, 0 (zero) is a natural number, which means that their formulation is impossible to prove (because 0 is a counter-example: $0 \times 1 \times 2 \times 3 = 0^2$). By adding the adjective *positive*, the problem statement becomes clearer and more precise. As mentioned in chapter 2, concision and avoidance of unnecessary detail can have a great impact on our solutions.

When creating new scenarios, we recommend the use of recreational problems. Based on our experience, it is more appealing for the students to solve a problem that is formulated in a way understandable by a general audience. For example, consider the problem statement of scenario 12, "The King Who Loved Diagonals":

> A very rich king wanted to thank one of his knights for leading his soldiers in a victorious battle. So he chose four large rooms of his castle that had the floor equally tiled. In each of these rooms, he drew a straight diagonal line connecting two opposite corners. Where the line crossed exactly four tiles, he placed one gold coin. (He actually ordered someone to draw the lines and place the coins. After all, *he was the king*!)
>
> The four rooms were all of different sizes:
>
> - Room 0: $(2^{11}-1) \times (2^{13}-1)$ tiles, i.e., 2047×8191 tiles
> - Room 1: $(2^{15}-1) \times (2^{20}-1)$ tiles, i.e., 32767×1048575 tiles
> - Room 2: $(2^{17}-1) \times (2^{21}-1)$ tiles, i.e., 131071×2097151 tiles
> - Room 3: $(2^{20}-1) \times (2^{22}-1)$ tiles, i.e., 1048575×4194303 tiles
>
> On the day that all coins were placed, he explained to the knight what he has done. He told him the sizes of the four rooms and he allowed him to collect all the gold coins from one of the rooms (and only one!).
>
> Which room should the knight choose so that he collects a maximum number of gold coins?

Although unreal, the problem is simple to understand: there is one king, one knight, four rooms, gold coins placed on the diagonals of the rooms in a specific way, and we want to choose the room with the largest number of coins. This problem is solved by using two theorems about the greatest common divisor. From the first theorem, we are able to conclude that the number of coins in each room equals the greatest common divisor of its two dimensions. From the second theorem, we conclude that the so-called Mersenne function (2^k-1) distributes through the greatest common divisor. Putting these two theorems together, we can easily compute the number of coins in each room

and solve the problem. Had we formulated the problem as an exercise in number theory, that is, by asking the students to directly use the theorems, the problem would be less attractive.

5.2.3 Prerequisites

This section lists prerequisites that should be met by the students. Teachers can use it to determine if the scenario is adequate for their students, so it is important to list the most important concepts that the solution depends upon and that are assumed to be known. This section can also be used to express dependencies on other scenarios; for example, scenario 12 depends on some of the exercises included in scenario 11.

5.2.4 Resolution

This section presents a possible solution for the problem, preferably in the style advocated in this thesis. As mentioned in the introduction of this chapter, we recommend the solution to be method-oriented, rather than solution-oriented. It is important that all steps are well motivated and justified.

5.2.5 Notes for the teacher

In this section, the solution presented above is decomposed into its main parts and each part is discussed in detail. This section can contain recommendations on how the teacher should present the material. For example, in scenario 9, "The Chameleons of Camelot", we suggest the teacher to provide an informal explanation:

> The teacher should explain that the above expression executes only once, i.e., once an assignment is selected, it is executed, and the process stops. This motivates the introduction of loops:
>
> $$\begin{array}{lll} \mathsf{do} & 0<g \wedge 0<b & \rightarrow \; g,b,c := g-1,b-1,c+2 \\ \Box & 0<g \wedge 0<c & \rightarrow \; g,b,c := g-1,b+2,c-1 \\ \Box & 0<b \wedge 0<c & \rightarrow \; g,b,c := g+2,b-1,c-1 \\ \mathsf{od} & . & \end{array}$$
>
> An informal explanation can be useful (e.g. "The do $\cdots$ od means that one of the assignments will be repeatedly chosen to be executed until all the guards evaluate to false.").

We also recommend the inclusion of questions that the teacher should ask and questions that the teacher should *not* ask. The inspiration for this came from the book *How to Solve It* [Pol90], where Pólya argues that the teacher's method of questioning should be *unobtrusive*. One of the first examples he uses is about finding the length of the diagonal of a classroom. He suggests a series of questions that aim at helping the students reaching the conclusion that they must use the theorem of Pythagoras. He then elaborates on "Good questions and bad questions":

> Let us go back to the situation as it presented itself at the beginning of section 10 when the question was asked: *Do you know a related problem?*. Instead of this, with the best intention to help the students, the question may be offered: *Could you apply the theorem of Pythagoras?*
>
> The intention may be the best, but the question is about the worst. We must realize in what situation it was offered; then we shall see that there is a long sequence of objections against that sort of "help."

He then lists several situations depending on whether the students understand the goal of the question and he offers some objections:

> (2) If the suggestion is understood, it gives the whole secret away, very little remains for the student to do.
>
> (...)
>
> (4) Even if he understands the suggestion, the student can scarcely understand how the teacher came to the idea of putting such a question. And how could he, the student, find such a question by himself? It appears as an unnatural surprise, as a rabbit pulled out of a hat; it is really not instructive.

We agree with Pólya and we think that by including questions that the teacher should not ask, we are helping the teacher in promoting self-discovery by the students. However, in our experience, it is difficult to come up with a set of effective questions. Some of the questions that we propose were tested in a classroom environment, whilst others had to be imagined. For example, the following question, taken from scenario 8 ("Exchanging the Values of Two Variables") was never tested, but, clearly, it should not be asked, since it gives a crucial property away:

> Can we use unitpotency[1] to eliminate the subexpression $y \otimes y$ in $x \otimes (y \otimes y)$?

[1] We say that $\otimes$ is unitpotent if, for all y, $y \otimes y = 1_{\otimes}$.

The questions that we propose are not rigid; we recommend teachers to ignore or adapt them to their own teaching style, as long as they stay unobtrusive. Also, it is important to note that we do not include all the questions that should not be asked; we only include some suggestions.

The inclusion of questions that the teacher should not ask is also useful for stressing that certain methods should be avoided. For example, in scenario 3 ("Knights or Knaves"), one of the questions that we recommend not to be asked is *What are the possible cases?*, because we think it is better for the students to model and analyse the problem, rather than solving it by brute-force. (In fact, the goal of the scenario is to illustrate how we can calculate the solutions to logic puzzles and avoid case analysis.)

Finally, it is worth mentioning that including both this section and the previous one ("Resolution") can lead to a substantial amount of duplication. Nevertheless, we recommend the inclusion of both, because some readers may not be interested in educational remarks.

5.2.6 Extensions and exercises

This section can be used for homework or for project assignments. Exercises can be additional questions about the problem discussed in the scenario or even variations and generalisations of the problem. Sometimes, we also include different problems that illustrate related principles. We recommend that all the exercises are accompanied by their solutions.

5.2.7 Further reading

This section contains recommended reading for the teacher and the students. It is a good place to include discussions and comparisons with alternative solutions. For example, in scenario 5 ("A Logical Race"), we include the following comment:

> The problem presented in this scenario was taken from [Hon98, p. 17]. In there, Honsberger solves the puzzle by translating it to the domain of numbers and formulating the relevant properties in terms of numbers. His solution is an extreme case of what is conventionally done in school mathematics: problems are always formulated using the more familiar domain of numbers and logic is used implicitly in the arguments. (We consider his solution extreme, since the problem was originally a logic problem. There was no need at all to translate it to a different, more complex, domain.) We

recommend the teacher to compare both solutions.

5.3 A catalogue of teaching scenarios

In this section, we briefly describe the catalogue of teaching scenarios that is included in appendix I. The catalogue contains 12 scenarios divided in two main parts.

The first part consists of scenarios 1 to 6 and is mainly on the use of formalism, the calculational method, and goal-oriented investigations. In particular, scenarios 3, 4, and 5 are on three different types of logic puzzles that can be solved calculationally. We believe this part can be used to introduce formal modelling and calculational logic. It can also be used to practise the techniques of symmetry and distributivity.

The second part consists of scenarios 7 to 12 and its focus is on solving algorithmic problems. It introduces the notion of invariant, it shows how to formally manipulate algorithms, it shows how to model and solve problems that are based on algorithms, and it terminates with the derivation of an algorithm from its formal specification. To facilitate the teaching of material from chapter 4, the final two scenarios are self-contained solutions to problems on the greatest common divisor of two numbers. With the exception of program inversion, the second part can be used to practise all the techniques discussed in chapter 3.

For the reader's convenience, we now list the titles and abstracts of the 12 scenarios.

Scenario 1: Exploring Algebraic Symmetries This teaching scenario presents a problem that admits a simple solution by exploring two important principles in problem solving: goal-directed constructions and algebraic symmetries. It can be used to introduce the notion of calculational proof format and to practise formal modelling.

Scenario 2: Calculating Orderings Between Two Numbers The goal of this teaching scenario is to illustrate the effectiveness of calculational and goal-directed constructions. In particular, it shows how the introduction of variables for representing objects other than numbers can help. It can also be used to introduce the notions of calculational proof and monotonicity.

Scenario 3: The Island of Knights and Knaves This teaching scenario is about a type of logic puzzle where the algebraic properties of equivalence play a central role. We adopt a calculational approach: instead of solving the puzzle by performing case

analysis, we calculate its solution. The scenario can be used as an introduction to calculational logic and to Boolean equality.

Scenario 4: Portia's Casket This scenario is about a type of logic puzzle that can be reduced to a system of simultaneous equations on Booleans. It can be used to introduce logical implication and to practise calculational logic.

Scenario 5: A Logical Race This scenario shows how a calculational approach to logic leads to a concise solution of a type of logic puzzle that is based on unique existential quantifications. It can be used to introduce Boolean inequivalence ($\not\equiv$), to practise formal modelling, and to illustrate how distributivity can be used to simplify mathematical arguments. The puzzle, which we have found in [Hon98, p. 17], is about deducing a conclusion based on the statements of three people. We also show (in the exercises) how we can generalise this type of logic puzzle.

Scenario 6: A Calculational Proof of the Handshaking Lemma This teaching scenario shows a goal-oriented and calculational proof of the Handshaking lemma, an elementary result in graph theory. The lemma states that every finite undirected graph has an even number of vertices with odd degree. The solution presented in this scenario can be used to introduce the Eindhoven quantifier notation.

Scenario 7: Moving a Heavy Armchair This scenario introduces the notion of *invariant* through a simple and recreational example. The problem was taken from [Bac03, Chapter 12] and does not require any prerequisites from the students.

Scenario 8: Exchanging the Values of Two Variables This scenario discusses and generalises a programming trick that can be used to exchange the values of two variables without using additional variables. It serves as an introduction to formal manipulation of algorithms and it can be also be used to introduce the Guarded Command Language. In our view, it is also a good example of investigative mathematics.

Scenario 9: The Chameleons of Camelot This scenario presents a generalisation of the problem "The Chameleons of Camelot", found in [Hon97, p. 140] (a more recent and accessible reference is [Win09]). Its goal is to help students recognise, model, and solve algorithmic problems. The solution is goal-oriented and explores an invariant of

the underlying non-deterministic algorithm. It is also an example of problem decomposition and it can be used to convey the notions of loop, guard, postcondition, and non-determinism. We also show *how* we can achieve the goal, rather than just showing it is possible to achieve it. The constructive argument involves a discussion on program termination that can be used to introduce the concept of bound function.

Scenario 10: Will This Algorithm Terminate? This scenario presents a problem from the St. Petersburg City Olympiad 1996, whose goal is to prove that a given algorithm terminates. It can be used to introduce the topic of program termination, the concept of bound function, and to help the students practise termination proofs.

Scenario 11: Constructing Euclid's Algorithm The goal of this scenario is to derive Euclid's algorithm to compute the greatest common divisor of two positive natural numbers. We also show how to use the algorithm to verify theorems related with the greatest common divisor. The scenario can be used to introduce the construction of programs from their formal specifications and the use of invariance to prove theorems.

Scenario 12: The King who Loved Diagonals This scenario is about a recreational problem that can be solved by using the solutions to two exercises from scenario 11. It can be used to practise formal modelling and to learn interesting properties related with the greatest common divisor of two numbers.

CHAPTER 6

Conclusion

The progress of Science consists in observing interconnections and in showing with a patient ingenuity that the events of this ever-shifting world are but examples of a few general relations, called laws. To see what is general in what is particular, and what is permanent in what is transitory, is the aim of scientific thought.

— ALFRED NORTH WHITEHEAD (1911)

The initial goal of this study was to investigate how algorithmic skills might be used to reinvigorate the teaching of mathematics and computing. Towards that goal, we have decided to create new educational material that can be used to introduce and teach algorithmic principles and techniques. In particular, we have rewritten material on introductory number theory, and we have created teaching scenarios that can be used to introduce and teach specific skills.

Based on the results, we are convinced that goal-oriented, calculational algorithmic skills can indeed be used to enrich and reinvigorate the teaching of mathematics and computing. The focus on the algorithmic content of number theory, for example, allowed the systematisation of existing proofs and, more importantly, the construction of new knowledge in a practical and elegant way. Also, we believe that the solutions presented in the scenarios illustrate the advantages of an emphasis on algorithmic skills. We now feel that we have enough material for the teachers to use and to do a serious and extensive study on the suitability of the methods described in the thesis.

Some preliminary experiences show that the educational material shown in this thesis can be used with success. For example, we have conducted a study at the University of Nottingham, whose goal was to assess whether the students registered on the first-year module "Mathematics for Computer Scientists" appreciate the calculational method. We wanted to determine what the students think of calculational proofs, compared with more conventional ones, and which are easier to verify; we also assessed

how their opinions changed during the term. The module covers basic concepts in mathematics of relevance to the development of computer software (Boolean algebra, elementary number theory, sets, functions and relations, quantifiers, and simple induction on natural numbers). There were a total of 135 students registered on the module. They had two lectures per week, associated coursework and weekly tutorials. Their feedback was collected through supplementary questions included in seven of the nine courseworks released. The participation was on a voluntary basis, but the students would have extra marks if they expressed their opinions.

The study consisted of two parts: "Proof Reading" and "Problem Solving". In "Proof Reading", we have shown calculational and conventional proofs for the same theorem and we have asked the students which one they preferred. We repeated the same questions later in the term to measure how their opinions changed. In "Problem Solving", we have asked them to solve the same problems at the beginning and later in the term, so that we could compare the solutions and determine if our methods have influenced them.

The results obtained show that most students prefer or understand better the calculational proofs. For example, in the first coursework, 69% of the students preferred the calculational version. In the second coursework, the theorem was that the square root of 2 is an irrational number; contrary to the previous results, 63% of the students preferred the conventional version. We believe that the major problem with the calculational proof was the introduction of a new auxiliary function. Nevertheless, the results show that the students' preferences depend on the specific examples being used. Indeed, it is not enough to rewrite a proof using the calculational proof format; we also have to explain and motivate the techniques that we use.

To assess which type of proof is easier to verify, we have included (in the fourth coursework) two incorrect proofs that the square root of 4 is irrational (the proofs were the same as the ones shown in the second coursework, but with all the relevant occurrences of 2 replaced by 4). Surprisingly, 74% of the students did not detect any mistake; moreover, 59% of these preferred the conventional proof, and 41% the calculational one. Of the 26% of the students that detected the mistake, most of them (69%) detected the error in the calculational proof. Interestingly, there were no students detecting the error in the conventional proof and choosing it as their favourite. The results also suggest that a significant number of the students who preferred the conventional proof in the second coursework may not have understood it properly.

There was a relatively large number of students changing their preference from the conventional to the calculational proofs during the term (69%). This suggests that,

as the students got more familiar with the calculational format, they found it better. Indeed, we think that the students would be prepared to switch to the calculational format if given more practice.

Regarding the "Problem Solving" part, we observed that most students had no difficulties solving the problems posed. Moreover, a significant number of students improved their solutions during the term. However, their use of the calculational style was not effective. For more details about this study, we refer the reader to [FM09].

As a result of another experience, we believe that we can introduce material from this thesis at lower levels of the educational system. In July 2010, we organised a one-week workshop for Portuguese secondary-school students (aged between 14 and 17) on algorithmic problem solving. The goal was to show the students how the principles and techniques developed by computing scientists can be used to model and solve problems. In particular, we had the opportunity to test how pre-university students react to the calculational method and proof format, and to the material shown in the scenarios 4, 7, and 9. We also discussed the problem "Goat, Cabbage and Wolf", shown in chapter 2. There were 13 students registered in the workshop; all of them were above average students. The material was surprisingly well received. For example, they have calculated the solution to the logic puzzle shown in scenario 4 very easily, which suggests that calculational logic can indeed be introduced at secondary-school level. Furthermore, most of them enjoyed the recreational flavour of the problems, and, at the end of the week, they were able to apply techniques like invariants and symmetry by themselves (in particular, most of them were able to solve by themselves the arm chair problem shown in scenario 7).

In the workshop, we also used two software tools: the Alloy Analyzer, which was used to analyse specifications written in the Alloy specification language [Jac06], and NetLogo [TW04], a multi-agent programmable modelling environment. The goal was to illustrate how we could use the computer to model problems and reason about some of their properties. In particular, we have modelled the problem shown in the scenario 9, "The Chameleons of Camelot". Using Alloy Analyzer, the students were able to find examples of arguments for which the problem is possible to solve. However, when provided with arguments for which there is no solution, the tool was not able to produce any answer (because it could not find any solution). We then modelled the problem as shown in scenario 9 and we were able to get a definitive answer.

We also modelled the non-deterministic algorithm that arranges meetings between chameleons in NetLogo. The graphical interface of the tool enriched the experience and allowed the students to interact with the problem. Motivated by the inefficiency

of the non-deterministic algorithm, some students removed the non-determinism and constructed an efficient deterministic version.

The feedback from the students was positive. They liked the recreational flavour of the problems, the interactivity provided by the software tools that we have used, and they enjoyed being challenged.

Although we have conducted some preliminary experiences to assess what students think of some of the material developed, it was never our intention to do any extensive educational studies. Instead, our goal was to investigate how calculational algorithmic skills might be used to solve algorithmic problems and to do mathematics, and to create material supporting the methods described in this thesis. In fact, we do not believe that the impact of our study can be assessed by statistically analysing classroom experiments. The standard ways of assessing new educational methods, like the use of test and control groups, randomised trials, and assessment based on lectures to students have serious flaws. (Some of the difficulties involved in the assessment are pointed out by Herbert Wilf in his essay [Wil].) Nevertheless, the novel results achieved in this thesis, the preliminary results on the didactical suitability of the calculational method obtained in [FM09], and our teaching experience encourage us to continue our efforts. Also, the success claimed by related work like [BMPS08] and [MM08], makes us believe that we can have a positive impact.

In our view, this thesis can be used as a starting point for a broader educational programme. There are certainly some aspects that have to improve, and other unexplored aspects that deserve to be explored. In the next section we discuss some of these aspects and some future directions.

6.1 Future work

Supporting the teaching of mathematics Although chapter 4 can be used to support a module on elementary number theory, it lacks some important material that should be taught in such a module. For example, it does not include fundamental topics such as primality or number theoretic functions (e.g., functions τ and σ). A valuable direction would be to extend the chapter in order to support a module that stresses the algorithmic content of the theory.

Also, some material shown in chapter 4 can be further developed. For example, it would be nice to know if the condition in lemma 4.4.3 is necessary for a function to distribute over $\triangledown$. Another interesting extension would be to generalise and adapt

the algorithm developed in section 4.6.5 to the Smith-Cornacchia algorithm (see the discussion in page 137).

Finally, in addition to number theory, there are other mathematical areas that can benefit from an algorithmic and calculational approach. In [GS93], for example, the authors show how the calculational method can be used to teach Discrete Mathematics. Also, some members of the project CryptoForma are working on calculational aspects of cryptography [Gru08, BG10]. Another recent example is [Bou09], where the author shows how to make temporal logic more calculational.

Teaching scenarios The scenarios included in appendix I were not tested by anyone, other than the author of this thesis (who only had the opportunity to test some of them). Therefore, before we can use them in schools, we think it is important to get feedback from mathematics teachers. The feedback would allow us to refine the scenarios and to accurately determine the level at which they can be used.

With the help of mathematics teachers, it would be useful to prepare different packages of scenarios aimed at different levels and audiences. A way to enrich these packages would be to create a glossary with the concepts that each scenario lists. The glossary could be done in the same spirit as the second part of Pólya's book *How to Solve It* [Pol90].

A natural direction is to create more scenarios, for we believe that the method we support can only succeed if there is an abundance of material and guides ready for the teachers to use. In our opinion, providing resources and assistance to the teachers is the best way to overcome the challenge of convincing them to use the approach we propose.

Finally, we think it would be interesting to investigate and create software tools that could be used to support the scenarios. For example, educational computer games could be used to put into practice the relevant techniques to solve river-crossing problems and logic puzzles.

Techniques for algorithmic problem solving The techniques listed in chapter 3 are the main techniques that support the educational material of this thesis. It would be good to create more educational material that illustrates these techniques. In particular, it would be interesting to solve more mathematical problems using program inversion, since it is a technique that is not widely used in mathematics.

Another direction is to explore other techniques. Polynomials, for example, can be

used to model many problems. Combining polynomial arithmetic with the notion of invariance can lead to elegant solutions. An example is [BCF10], which explores this technique to solve one-person solitaire-like games. Another technique that can lead to effective and concise solutions of problems involving sequences is the use of generating functions [GKP94, p. 320], which are functions that "generate" infinite sequences. The idea of capturing an infinite structure into a manipulable and concise expression is appealing. We think it would be useful to investigate calculational approaches to the theory of generating functions and their application to the design of algorithms on sequences. Perhaps a good starting point is the functional approach to streams described in [Hin08].

Finally, we believe it would be interesting to explore and systematise the derivation of algorithms that can be specified by Galois connections (in chapter 4, we have seen two algorithms that were specified by Galois connections: the integer division and Euclid's algorithms). There is some undergoing related work in that direction [Oli10b, Oli10a].

Assessing the impact of the methods supported by this thesis To conclude, we believe that it is important to do more educational studies like the ones described in the introduction of this chapter. Also, it would be valuable to collaborate with researchers in mathematics education, so that we can assess more precisely how the methods supported by this thesis compare with conventional approaches. Now that we have material for the teachers to use, we believe we can do a serious and extensive study on the suitability of the methods described in this thesis.

References

[AZ04] Martin Aigner and Günter Ziegler. *Proofs From The Book, 3rd Edition*. Springer-Verlag, 2004.

[Bac02] Roland Backhouse. The art of effective reasoning. September 2002.

[Bac03] Roland C. Backhouse. *Program Construction. Calculating Implementations from Specifications.* John Wiley & Sons, 2003.

[Bac06] Roland Backhouse. Algorithmic problem solving — three years on. In *Teaching Formal Methods: Practice and Experience*. BCS, The Chartered Institute for IT, 2006.

[Bac07] Roland Backhouse. Algorithmic problem solving. Lecture notes, School of Computer Science, University of Nottingham. Updated at least annually and widely available on the internet, but see author's website for latest version., September 2007.

[Bac09] Ralph-Johan Back. Structured derivations as a unified proof format for teaching mathematics. Technical Report 949, TUCS, Jul 2009.

[BCF10] Roland Backhouse, Wei Chen, and João Ferreira. The algorithmics of solitaire-like games. In Claude Bolduc, Jules Desharnais, and Béchir Ktari, editors, *Mathematics of Program Construction*, volume 6120 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 2010.

[BdM96] Richard S. Bird and Oege de Moor. *Algebra of Programming*. Prentice-Hall International, 1996.

[Bel08] E. T. Bell. *Men of Mathematics - The Lives and Achievements of the Great Mathematicians from Zeno to Poincaré.* Touchstone, July 2008.

[BF08] Roland Backhouse and João F. Ferreira. Recounting the rationals: Twice! In *Mathematics of Program Construction*, volume 5133 of *LNCS*, pages 79–91. Springer-Verlag, 2008.

[BF10] Roland Backhouse and João F. Ferreira. On Euclid's algorithm and elementary number theory. To appear in the journal *Science of Computer Programming.*, 2010.

[BG10] Eerke Boiten and Dan Grundy. The logic of large enough. In Claude Bolduc, Jules Desharnais, and Béchir Ktari, editors, *Mathematics of Program Construction*, volume 6120 of *Lecture Notes in Computer Science*, pages 42–57. Springer-Verlag, 2010.

[BGvW96] Ralph Back, Jim Grundy, and Joakim von Wright. Structured calculational proof. Technical Report TUCS-TR-65, 1996.

[Bla95] Andreas Blass. Seven trees in one. *Journal of Pure and Applied Algebra*, 103(1):1–21, 1995.

[BM06] Roland Backhouse and Diethard Michaelis. Exercises in quantifier manipulation. In *Mathematics of Program Construction 2006*, volume 4014 of *LNCS*, pages 69–81. Springer-Verlag, 2006.

[BMPS08] Ralph-Johan Back, Linda Mannila, Mia Peltomaki, and Patrick Sibelius. Structured derivations: A logic based approach to teaching mathematics. In *FORMED 2008: Formal Methods in Computer Science Education, Budapest*, 2008.

[Bou09] Raymond Boute. Making temporal logic calculational: A tool for unification and discovery. In *FM '09: Proceedings of the 2nd World Congress on Formal Methods*, pages 387–402, Berlin, Heidelberg, 2009. Springer-Verlag.

[BPSvW04] Ralph-Johan Back, Mia Peltomäki, Tapio Salakoski, and Joakim von Wright. Structured derivations supporting high-school mathematics. In A. Laine, J. Lavonen, and V. Meisalo, editors, *Proceedings of the 20th Annual Symposium of the Finnish Mathematics and Science Education Research Association*, Research Report 253, pages 104–122, Helsinki, Finland, 2004. Department of Applied Sciences of Education, University of Helsinki.

[Bri72] John Brillhart. Note on representing a prime as a sum of two squares. *Mathematics of Computation*, 26(120):1011–1013, 1972.

[Bro61] Achille Brocot. Calcul des rouages par approximation, nouvelle méthode. *Revue Chronométrique*, 3:186–194, 1861. Available via `http://joaoff.com/publications/2008/rationals/`.

[Bur05] David M. Burton. *Elementary Number Theory*. McGraw-Hill Higher Education, 6th revised edition, December 2005.

[BvW97] Ralph-Johan Back and Joakim von Wright. Doing high school mathematics carefully. Technical report, 1997.

[BvW06] Ralph-Johan Back and Joakim von Wright. *Mathematics with a little bit of logic: Structured derivations in high-school mathematics*. 2006. Manuscript.

[BW08] Joe Buhler and Stan Wagon. Basic algorithms in number theory. In Joseph P. Buhler and Peter Stevenhagen, editors, *Algorithmic Number Theory. Lattices, Number Fields, Curves and Cryptography*, pages 25–68. Cambridge University Press, 2008.

[BWF06] Tim Bell, Ian H. Witten, and Mike Fellows. *Computer Science Unplugged: An enrichment and extension programme for primary-aged children*. December 2006. Available from `http://csunplugged.org/index.php/en/books`.

[CELV99] F. W. Clarke, W. N. Everitt, L. L. Littlejohn, and S. J. R. Vorster. H. J. S. Smith and the Fermat two squares theorem. *The American Mathematical Monthly*, 106(7):652–665, 1999.

[Che90] Wei Chen. A formal approach to program inversion. In *CSC '90: Proceedings of the 1990 ACM annual conference on Cooperation*, pages 398–403. ACM Press, 1990.

[Cor08] G. Cornacchia. Su di un metodo per la risoluzione in numeri interi dell'equazione $\sum_{h=0}^{n} C_h x^{n-h} y^h = P$. *Giornale di Matematiche di Battaglini*, 46:33–90, 1908.

[CR06] Lucas Carlson and Leonard Richardson. *Ruby Cookbook (Cookbooks (O'Reilly))*. O'Reilly Media, Inc., 2006.

[CW00] Neil Calkin and Herbert S. Wilf. Recounting the rationals. *The American Mathematical Monthly*, 107(4):360–363, 2000.

[Dic99] Leonard E. Dickson. *History of the Theory of Numbers: Diophantine Analysis: Diophantine Analysis Vol 2 (AMS/Chelsea Publication)*. American Mathematical Society, August 1999.

[Dij68] Edsger W. Dijkstra. Go To statement considered harmful. *Comm. ACM*, 11(3):147–148, 1968.

[Dij75] Edsger W. Dijkstra. Guarded commands, non-determinacy and formal derivation of programs. *Comm. ACM*, 18(8):453–457, 1975.

[Dij76] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[Dij82a] Edsger W. Dijkstra. Program inversion. In *Selected Writings on Computing: A Personal Perspective*, pages 351–354. Springer-Verlag, 1982.

[Dij82b] Edsger W. Dijkstra. A theorem about odd powers of odd integers. In *Selected Writings on Computing: A Personal Perspective*, pages 349–350. Springer-Verlag, 1982.

[Dij87] Edsger W. Dijkstra. Our proof format. January 1987.

[Dij90] Edsger W. Dijkstra. Fibonacci and the greatest common divisor. April 1990.

[Dij93] Edsger W. Dijkstra. A derivation of a proof by D. Zagier. August 1993.

[DS90] E.W. Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics*. Springer-Verlag, 1990.

[Erd39a] Paul Erdös. Note on products of consecutive integers, I. *J. London Math. Soc.*, 14:194–198, 1939.

[Erd39b] Paul Erdös. Note on products of consecutive integers, II. *J. London Math. Soc.*, 14:245–249, 1939.

[ES75] Paul Erdös and J. L. Selfridge. The product of consecutive integers is never a power. *Illinois J. Math.*, 19:292–301, 1975.

[Fei87] Wim H. J. Feijen. On programming and mathematical reasoning. Circulated privately, October 1987.

[Fer09a] João F. Ferreira. On the inexistence of a unique existential binary operator. Source code available from `http://github.com/jff`, August 2009.

[Fer09b] João F. Ferreira. On the unique existential quantifier. Available from the author, November 2009.

[Fer10] João F. Ferreira. Designing an algorithmic proof of the two-squares theorem. In Claude Bolduc, Jules Desharnais, and Béchir Ktari, editors, *Mathematics of Program Construction*, volume 6120 of *LNCS*, pages 140–156. Springer-Verlag, 2010.

[FM09] João F. Ferreira and Alexandra Mendes. Students' feedback on teaching mathematics through the calculational method. In *Frontiers in Education Conference, 2009. FIE '09. 39th IEEE*, 2009.

[FMBB09] João F. Ferreira, Alexandra Mendes, Roland Backhouse, and Luís S. Barbosa. Which mathematics for the information society? In *Teaching Formal Methods*, volume 5846 of *LNCS*, pages 39–56. Springer-Verlag, 2009.

[Fra98] John B. Fraleigh. *A First Course in Abstract Algebra*. Addison-Wesley, 6th edition, 1998.

[Gau01] Carl F. Gauss. *Disquisitiones Arithmeticae*. G. Fleischer, Leipzig, 1801. English translation by A. A. Clarke, Springer-Verlag, 1986.

[GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics : a Foundation for Computer Science*. Addison-Wesley, second edition, 1994.

[GLB06] Jeremy Gibbons, David Lester, and Richard Bird. Enumerating the rationals. *Journal of Functional Programming*, 16(3):281–291, 2006.

[Gri81] David Gries. *The Science of Programming*. Springer-Verlag, 1981.

[Gru08] Dan Grundy. *Concepts and Calculation in Cryptography*. PhD thesis, Computing Laboratory, University of Kent, 2008. Available from `www.cs.kent.ac.uk/~eab2/crypto/thesis.web.pdf`.

[GS93] David Gries and Fred B. Schneider. *A Logical Approach to Discrete Math*. Springer-Verlag, 1993.

[GS95] David Gries and Fred B. Schneider. Teaching math more effectively, through calculational proofs. *The American Mathematical Monthly*, 102(8):691–697, 1995.

[Hay00] Brian Hayes. On the teeth of wheels. *American Scientist*, 88(4):296–300, July 2000.

[HE56] Thomas L. Heath and Euclid. *The Thirteen Books of Euclid's Elements, Books III–IX*, volume II. Dover Publications, Incorporated, 1956.

[Her48] Hermite. Note au sujet de l'article précédent. *Journal de Mathématiques Pures et Appliquées*, 13:15, 1848.

[Hin08] Ralf Hinze. Functional pearl: streams and unique fixed points. In *ICFP '08: Proceeding of the 13th ACM SIGPLAN international conference on Functional programming*, pages 189–200, New York, NY, USA, 2008. ACM.

[Hir95] Keith E. Hirst. *Numbers, Sequences and Series*. Edward Arnold, 1995.

[HMW90] Kenneth Hardy, Joseph B. Muskat, and Kenneth S. Williams. A deterministic algorithm for solving $n = fu^2 + gv^2$ in coprime integers u and v. *Mathematics of Computation*, 55(191):327–343, July 1990.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, October 1969.

[Hon97] Ross Honsberger. *In Polya's Footsteps: Miscellaneous Problems and Essays (Dolciani Mathematical Expositions)*. The Mathematical Association of America, October 1997.

[Hon98] Ross Honsberger. *Ingenuity in Mathematics (New Mathematical Library)*. The Mathematical Association of America, August 1998.

[IPL95] The calculational method. volume 53, Amsterdam, The Netherlands, February 1995. Elsevier North-Holland, Inc.

[Jac06] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006.

[JJ98] Gareth A. Jones and Josephine M. Jones. *Elementary Number Theory*. Springer-Verlag, July 1998.

[Knu97] Donald E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, Boston, MA, USA, 3rd edition, 1997.

[KRSS03] Donald E. Knuth, C.P. Rupert, Alex Smith, and Richard Stong. Recounting the rationals, continued. *The American Mathematical Monthly*, 110(7):642–643, 2003.

[Law91] F. W. Lawvere. Some thoughts on the future of category theory. *Lecture Notes in Mathematics*, 1488:1–13, 1991.

[MB03] Shin-Cheng Mu and Richard Bird. Rebuilding a tree from its traversals: A case study of program inversion. In *Programming Languages and Systems*, volume 2895 of *LNCS*, pages 265–282. Springer-Verlag, 2003.

[MJ84] F. L. Morris and C. B. Jones. An early program proof by Alan Turing. *IEEE Ann. Hist. Comput.*, 6(2):139–143, 1984.

[MM08] Z. Michalewicz and M. Michalewicz. *Puzzle-based Learning: Introduction to Critical Thinking, Mathematics, and Problem Solving*. Hybrid Publishers, 1st edition, 2008.

[Oli10a] J.N. Oliveira. A (calculational) look at optimization. Talk at the 1st Workhsop of the Mondrian Project, Aveiro, Portugal, July 2010.

[Oli10b] J.N. Oliveira. A look at program "G"alculation. Talk at IFIP WG 2.1 65th Meeting, Póvoa de Lanhoso, Portugal, January 2010.

[PC06] Tim Patrick and John Craig. *Visual Basic 2005 Cookbook: Solutions for VB 2005 Programmers (Cookbooks (O'Reilly))*. O'Reilly Media, Inc., 2006.

[Pip07] Dan Piponi. Arboreal isomorphisms from nuclear pennies, September 2007. Blog post available at http://blog.sigfpe.com/2007/09/arboreal-isomorphisms-from-nuclear.html.

[Pol81] George Polya. *Mathematical Discovery. On understanding, learning, and teaching problem solving*. John Wiley & Sons, combined edition, 1981.

[Pol90] George Polya. *How to Solve It (Penguin Science)*. Penguin Books Ltd, April 1990.

[Ser48] J. A. Serret. Sur un théorème relatif aux nombres entiers. *Journal de Mathématiques Pures et Appliquées*, 13:12–14, 1848.

[Slo] N. J. A. Sloane. The on-line encyclopedia of integer sequences.

[Ste58] Moritz A. Stern. Üeber eine zahlentheoretische Funktion. *Journal für die reine und angewandte Mathematik*, 55:193–220, 1858.

[TW04] Seth Tisue and Uri Wilensky. Netlogo: A simple environment for modeling complexity. In *in International Conference on Complex Systems*, pages 16–21, 2004.

[vdS91] Jan L. A. van de Snepscheut. Inversion of a recursive tree traversal. *Inf. Process. Lett.*, 39(5):265–267, 1991.

[vdS93] Jan L.A. van de Snepscheut. *What Computing Is All About*. Springer-Verlag, 1993.

[vG90] Antonetta J. M. van Gasteren. *On the Shape of Mathematical Arguments*. Number 445 in Lecture Notes in Computer Science. Springer-Verlag, New York, NY, USA, 1990.

[vW91] J. von Wright. Program inversion in the refinement calculus. *Inf. Process. Lett.*, 37(2):95–100, 1991.

[Wag90] Stan Wagon. Editor's corner: The Euclidean algorithm strikes again. *The American Mathematical Monthly*, 97(2):125–129, 1990.

[Wil] Herbert S. Wilf. Can there be "research in mathematical education"? Available from `http://www.math.upenn.edu/~wilf/website/PSUTalk.pdf`.

[Win06] Jeannette M. Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, March 2006.

[Win09] Peter Winkler. Puzzled: Understanding relationships among numbers. *Communications of the ACM*, 52(5):112, 2009.

[Zag90] D. Zagier. A one-sentence proof that every prime $p \equiv 1 (mod\ 4)$ is a sum of two squares. *The American Mathematical Monthly*, 97(2):144+, 1990.

[Zei06] Paul Zeitz. *The Art and Craft of Problem Solving*. John Wiley & Sons, 2nd edition, September 2006.

Appendix I

Teaching Scenarios for Teaching Algorithmic Problem Solving

SCENARIO 1

Exploring Algebraic Symmetries

1.1 Brief description and goals

This teaching scenario presents a problem that admits a simple solution by exploring two important principles in problem solving: goal-directed constructions and algebraic symmetries. It can be used to introduce distributivity, the notion of calculational proof format, and to practise formal modelling.

1.2 Problem

Prove that the product of four consecutive positive natural numbers cannot be the square of an integer number.

1.3 Prerequisites

Elementary algebraic properties: multiplication distributes over addition and difference of two squares.

1.4 Resolution and notes

The goal is to prove that the product of four consecutive positive natural numbers cannot be the square of an integer number. Our first step is to formalise the goal, so that we can be precise about what the problem requires. Assuming that n is a positive natural number, one way of formalising the goal is

$$S.(n(n+1)(n+2)(n+3))$$

$=$ { justification }

false,

where $S.n$ is defined to be true only when n is the square of an integer. We now have to provide a convincing (and preferably short and well-motivated) justification that allows us to conclude the equality stated above. In other words, we have to manipulate the first expression until we get an intermediate expression that is obviously false. Because we do not know any properties involving S and the product $n(n+1)(n+2)(n+3)$, we have to transform the product into an expression that is not a square. But how can we show that something is not a square? This strategy does not seem very effective, since we cannot think of an easy criterion for determining when a number is a square.

We need a new strategy. Having nothing else to play with, let us manipulate the product $n(n+1)(n+2)(n+3)$ by using the property that multiplication distributes over addition. We record the manipulations using the format shown above, where two equivalent expressions are written in separate lines and the justification for their equality is written between them in curly brackets.

$S.(n(n+1)(n+2)(n+3))$

$=$ { we use distributivity, twice; there are three ways in which
we can develop the product, so we choose to multiply
n by $n+3$ and $n+1$ by $n+2$ in order to introduce symmetry
—both have the term n^2+3n }

$S.((n^2+3n)(n^2+3n+2))$

$=$ { we introduce symmetry again; we want to transform the
argument of S into an expression that looks like a square }

$S.(((n^2+3n+1)-1)((n^2+3n+1)+1))$

$=$ { difference of two squares, i.e., $(m-1)(m+1) = m^2-1$ }

$S.((n^2+3n+1)^2-1)$

$=$ { there are no two consecutive positive integers
that are both squares }

false .

Note how symmetry guided our calculation! In fact, this problem is *asking* for symmetry, since the goal is to determine if we can express the product $n(n+1)(n+2)(n+3)$ as a product of two equal numbers, i.e., as a product with a symmetric shape. In general, exploring and identifying symmetry is an important problem-solving skill, since it can help us to avoid unnecessary duplication and to simplify expressions. Also, if the solution of an algorithmic problem is symmetric, it means that we only need to solve half of the problem: the second half can be immediately derived from the first. The river-crossing problems shown in [Bac07] are excellent examples of symmetric algorithmic problems.

1.5 For the teacher

Formalising the problem in a goal-oriented way The teacher should help the students formalising the problem and defining a structure for their argument. We suggest the following structure for the argument:

$$
\begin{array}{ll}
 & S.(n(n+1)(n+2)(n+3)) \\
= & \quad \{ \quad \text{justification} \quad \} \\
 & \mathsf{false},
\end{array}
$$

where n is a positive natural number and $S.n$ is defined to be true only when n is the square of an integer.

We now have to provide a convincing (and preferably short and well-motivated) justification that allows us to conclude the equality stated above. In other words, we have to manipulate the first expression until we get an intermediate expression that is obviously false. Because we do not know any properties involving S and the product $n(n+1)(n+2)(n+3)$, we have to transform the product into an expression that is not a square. The teacher may extend the structure shown above to reflect this strategy:

$$
\begin{array}{ll}
 & S.(n(n+1)(n+2)(n+3)) \\
= & \quad \{ \quad \text{justification} \quad \} \\
 & S.(E.n) \\
= & \quad \{ \quad \text{the expression } E.n \text{ is not a square} \quad \} \\
 & \mathsf{false} \ .
\end{array}
$$

But how can we show that something is not a square? In other words, how can we specify $E.n$? This strategy does not seem very effective, since we cannot think of an easy criterion for determining when a number is a square. We need a new strategy.

Exploring the algebraic symmetries Having nothing else to play with, let us manipulate the product $n(n+1)(n+2)(n+3)$ by using the property that multiplication distributes over addition. We record the manipulations using the format shown above, where two equivalent expressions are written in separate lines and the justification for their equality is written between them in curly brackets. The teacher may want to provide more details on the proof format.

The students should understand that there are three different ways in which we can develop the expression

$$n(n+1)(n+2)(n+3) \quad .$$

They are:

- $(n^2+n)(n^2+5n+6)$
- $(n^2+2n)(n^2+4n+3)$
- $(n^2+3n)(n^2+3n+2)$

We recommend the teacher to ask the students which one they think is better. The discussion should lead them to choose the one that is more symmetric, that is:

$$(n^2+3n)(n^2+3n+2) \quad .$$

There are several reasons that justify the introduction of symmetry. First, when two formulae share some symmetry, it is easier to spot the differences between them. Second, symmetry implies duplication; by highlighting the places where duplication occurs, we may be able to avoid it. Third, and more specific to this example, we want to transform the product $n(n+1)(n+2)(n+3)$ into something similar to a square, i.e., into something similar to a product with a symmetric shape (a product of two equal numbers). Using the discussion on symmetry as a starting point, the following steps should be symmetry driven and the following calculation can be written.

$$\begin{array}{ll} & S.(n(n+1)(n+2)(n+3)) \\ = & \quad \{ \quad \text{we use distributivity, twice; there are three ways in which} \\ & \qquad \text{we can develop the product, so we choose to multiply} \end{array}$$

n by $n{+}3$ and $n{+}1$ by $n{+}2$ in order to introduce symmetry
—both have the term $n^2{+}3n$ }

$S.((n^2{+}3n)(n^2{+}3n+2))$

$=$ { we introduce symmetry again; we want to transform the argument of S into an expression that looks like a square }

$S.(((n^2{+}3n+1){-}1)((n^2{+}3n+1){+}1))$.

Note how exploring symmetry allowed us to rewrite the initial product into a product of two *almost equal* expressions. We observe that the subexpression $n^2{+}3n+1$ is duplicated and we know how to avoid it, since $(m{-}1)(m{+}1) = m^2{-}1$ for all m. We can thus write:

$S.(((n^2{+}3n+1){-}1)((n^2{+}3n+1){+}1))$

$=$ { difference of two squares, i.e., $(m{-}1)(m{+}1) = m^2{-}1$ }

$S.((n^2{+}3n+1)^2{-}1)$.

The final steps We have now reached a stage where we have the expression

$S.((n^2{+}3n+1)^2{-}1)$.

The teacher must be sure that the students understand that $(n^2{+}3n+1)^2$ is a square and that this expression means that the predecessor of $(n^2{+}3n+1)^2$ is also a square. Then, the teacher may ask the students if they know any two consecutive **positive** integer numbers that are both squares. Once they realise that there are no two consecutive positive squares, they can conclude the argument.

The teacher can ask the students what would happen if we relax the conditions and consider all natural numbers, that is, if we allow n to be a natural number (including zero).

1.5.1 Questions that the teacher should ask

Formalising the problem in a goal-oriented way

- *How can we express formally the product of four consecutive positive natural numbers?*

  The goal is to express formally the product of four consecutive numbers. Depending on the level of the students, the teacher can use this first step as an

exercise in formal modelling. If the students do not know how to express it, the teacher may write down the product of two consecutive positive natural numbers and ask for the answer for three and four.

- *What is our goal and how can we layout our calculation?*

  Once the product is formally expressed, the students should understand clearly what is the goal. A traditional way to express the goal is saying that the following equation has no solution in x:

  $$n(n+1)(n+2)(n+3) = x^2 \quad .$$

  Using this approach, the students should manipulate the left-hand expression and get a new expression that cannot be written as a square. Another alternative is to introduce a predicate S and use the following structure for the argument:

  $$\begin{array}{ll} & S.(n(n+1)(n+2)(n+3)) \\ = & \quad \{ \quad \text{justification} \quad \} \\ & \mathsf{false}, \end{array}$$

  where $S.n$ is defined to be true only when n is the square of an integer. We prefer this alternative, because it is a concise and goal-oriented translation of the problem statement.

- *How can we show that an expression is **not** a square?*

  The goal is to conclude that it is not easy to come up with a general criterion for determining when a number is not a square. It is possible that some students propose that an expression is not a square if it has the shape n^2-1 (although, according to our experience, that has never happened!). If that is the case, the teacher may act unaware that that is the criterion we are going to use. We usually write down the students' proposals in one of the corners of the whiteboard and when we reach the solution, we confront the students' guesses with the solution we have constructed. We think this is an effective way of showing that guessing is not always the best foundation for a mathematical investigation. (It also shows that, usually, students are not good guessers.)

Exploring the algebraic symmetries

- *In how many ways can we develop the expression* $n(n+1)(n+2)(n+3)$*?*

The goal is to make the students aware that the calculation can be developed in different ways. The teacher should write down (with the students' help) the three possible expressions.

- *From the three possible expressions, which one would you choose?* ***Why?***

  The goal is to introduce the idea that it is generally better to exploit symmetry. If they do not know which one to choose, we suggest the teacher to explain them that it is usually better to exploit symmetric formulae. The teacher can help them by explaining what algebraic symmetry is and by asking which one of the three possibilities is more symmetric. The teacher can also elaborate on the reasons shown above to justify the introduction of symmetry.

- *We now have the expression* $(n^2{+}3n)(n^2{+}3n+2)$. *Can we introduce more symmetry?*

  Once it is shown to the students how to make this expression more symmetric, it is easy to see how to do it. However, rewriting 2 as $1{+}1$ and 0 as $-1{+}1$ is not an obvious step for someone who does not have practice in algebraic manipulations. We suggest the teacher to first rewrite $n^2{+}3n$ as n^2+3n+0 and then rewrite 2 as $1{+}1$ and 0 as $-1{+}1$.

The final step

- *We have reached the expression* $S.((n^2{+}3n+1)^2{-}1)$. *What does it mean?*

  It is important to understand that $(n^2{+}3n+1)^2{-}1$ is the predecessor of a square number.

- *What is the distance between two consecutive squares?*

  Depending on the level of the students, the teacher may elaborate on the distance between two consecutive squares. We can do a simple calculation to compute this distance d :

$$
\begin{aligned}
& n^2{+}d = (n{+}1)^2 \\
= \quad & \{ \quad \text{arithmetic} \quad \} \\
& n^2{+}d \;=\; n^2+2n+1 \\
= \quad & \{ \quad \text{cancellation} \quad \} \\
& d \;=\; 2n+1 \;\; .
\end{aligned}
$$

  A consequence is that there are not two consecutive positive square numbers, and therefore $S.((n^2{+}3n+1)^2{-}1)$ is false.

- *We have seen that the theorem is valid for positive natural numbers. Is the theorem valid for all natural numbers?*

  The students should understand that if we consider 0, then the theorem is not valid. (Because $0\times1\times2\times3$ is 0^2.)

1.5.2 Questions that the teacher should not ask

Formalising the problem in a goal-oriented way

- *Is $n(n+1)(n+2)(n+3)$ a suitable formalisation of the product of four consecutive numbers?*

  It is better to leave to the students the formalisation of the product. If they are having problems doing it, the teacher may illustrate how we could formalise the product of two consecutive numbers.

Exploring the algebraic symmetries

- *What are the three possible developments for $n(n+1)(n+2)(n+3)$?*

  If they propose some specific manipulation, the teacher can ask why they have chosen that one and he may ask which other (and how many) manipulations are possible. In such a simple problem as this one, the teacher should leave most of the work to the students.

The final step

- *Can you see that there are no two consecutive positive natural numbers that are both squares?*

  This question gives away the crucial property that is used to solve the problem. We think it is better to ask what they can say about $(n^2+3n+1)^2-1$ and what is the distance between two consecutive squares, because these questions guide the students towards the relevant property, rather than just giving it away.

1.5.3 Concepts that the teacher should introduce

Calculational proof

Goal-directed investigations

Symmetry

1.6 Extensions and exercises

Exercise 1.6.1 (Warm-up) Can the product of two consecutive positive natural numbers be a square?

□

Exercise 1.6.2 (Odd number of divisors) Suppose that n is a natural number with an odd number of divisors. What can you say about n? (Try to exploit the following symmetry between the divisors of n: if k divides n, $n \div k$ also divides n.)

□

Exercise 1.6.3 (n doors in a row) You have n doors in a row that are all initially closed. Each door is sequentially numbered: the first door is door number 1, the second door is door number 2, etc. You make n passes by the doors starting with the first door every time. The first time through you visit every door and *toggle* the door (i.e., if the door is closed, you open it; if it is open, you close it). The second time you only visit every second door (doors 2,4,6, etc.). The third time, every 3rd door (doors 3, 6, 9, etc.), etc, until you only visit the 100th door.

At the end of this process, which doors are open?

□

1.7 Solutions to extensions and exercises

1.6.1 Using a similar approach as shown above, we want to determine the validity of

$$S.(n(n+1)) \quad ,$$

which is the same as

$$S.(n^2+n) \quad .$$

Now, the distance d between two consecutive squares can be easily computed:

$$\begin{array}{ll} & n^2+d = (n+1)^2 \\ = & \quad \{ \quad \text{arithmetic} \quad \} \end{array}$$

$$
\begin{aligned}
& n^2+d \;=\; n^2+2n+1 \\
= \quad & \{ \quad \text{cancellation} \quad \} \\
& d \;=\; 2n+1 \;\; .
\end{aligned}
$$

Because $n < 2n+1$, then n^2+n can not be a square and $S.(n^2+n)$ is false.
□

1.6.2 This exercise can be solved by using the following symmetry between the divisors of n: if k is a divisor of n, then $n \div k$ is also a divisor of n. This means that divisors come in pairs: for each divisor k, there is an associated divisor $n \div k$ (also, note that if k is a divisor of n, then $n \div (n \div k)$ is k). Take the example of the number 28, whose divisors are 1, 2, 4, 7, 14, and 28. Identifying the symmetries, we see that 28 has three pairs of divisors: 1 and $28 \div 1$, 2 and $28 \div 2$, and 4 and $28 \div 4$.

A consequence is that a natural number n has an odd number of divisors exactly when there exists a divisor k such that $k = n \div k$, that is, $k^2 = n$. For example, the number 16 has five divisors: 1, 2, 4, 8, and 16. Identifying the symmetries, we see that 1 is associated with 16, 2 is associated with 8, and 4 is associated with itself (i.e., $4 = 16 \div 4$). Therefore, we conclude that a number n with an odd number of divisors is a perfect square.
□

1.6.3 In the kth pass by the doors, you will toggle all the doors whose numbers are multiples of k. Moreover, because the doors were closed initially, if you toggle a door an even number of times, that door remains closed. Similarly, if you toggle a door an odd number of times, that door becomes open.

Therefore, at the end of the process, the open doors are the ones that were toggled an odd number of times. In other words, the open doors are the doors whose numbers have an odd number of divisors.

Now, using the result from the previous exercise, the open doors are the doors whose numbers are perfect squares.
□

1.8 Further reading

The problem discussed in this scenario was found in [Zei06, p. 4], where it is used to illustrate the strategy "get your hands dirty". The author computes the value of

$n(n+1)(n+2)(n+3)$ for the first six positive integers and he gets 24, 120, 360, 840, 1680, 17160. Then he writes "Just about everyone notices that the first two values are one less than a perfect square." and he conjectures that the value of $n(n+1)(n+2)(n+3)$ is one less than a perfect square. Finally, he verifies this conjecture by exploring symmetry (as seen in our solution). In our view, the guessing part is completely unnecessary and should be avoided. Whilst the guessed conjecture is specific to this problem, the emphasis on symmetry and algebraic manipulation can be helpful in many different problems.

To conclude, the problem discussed in this scenario is an instance of the more general theorem proved in 1939 by Erdös, stating that the *product of consecutive integers is never a square* [Erd39a, Erd39b]. The theorem is even more general, since Erdös and Selfridge published in 1975 a proof that the *product of two or more consecutive positive integers is never a power* [ES75].

SCENARIO 2

Calculating Orderings Between Two Numbers

2.1 Brief description and goals

The goal of this teaching scenario is to illustrate the effectiveness of calculational and goal-directed constructions. In particular, it shows how the introduction of variables for representing objects other than numbers can help. It can also be used to introduce the notions of calculational proof and monotonicity.

2.2 Problem

Is $\sqrt{2} + \sqrt{7}$ less, equal or greater than $\sqrt{3} + \sqrt{5}$? (Assume that all arising square roots are taken with the positive sign.)

2.3 Prerequisites

Ordering relations and elementary algebra skills.

2.4 Resolution and notes

The goal of the problem is to determine if the numbers $\sqrt{2} + \sqrt{7}$ and $\sqrt{3} + \sqrt{5}$ are equal, or if the first is greater, or if the second is greater. More formally, we want to *calculate* the relation R between the two numbers:

$$(\sqrt{2} + \sqrt{7}) \quad R \quad (\sqrt{3} + \sqrt{5}) \quad ,$$

where R is either less-than ($<$), equals ($=$), or greater-than ($>$). The difficulty in comparing the numbers is the presence of the square roots. However, we can eliminate them, because we know that we can square both sides related by R. In other words, we know that for positive a and b,

(2.4.1) $a^2 \; R \; b^2 \quad \equiv \quad a \; R \; b \quad .$

We usually describe this rule as "squaring is invertible and monotonic with respect to R". Together with the fact that addition is also invertible and monotonic with respect to R, we can calculate the relation R as follows:

$$
\begin{array}{ll}
 & (\sqrt{2} + \sqrt{7}) \quad R \quad (\sqrt{3} + \sqrt{5}) \\
= & \quad \{ \quad \text{squaring is invertible and monotonic with respect to } R \quad \} \\
 & (\sqrt{2} + \sqrt{7})^2 \quad R \quad (\sqrt{3} + \sqrt{5})^2 \\
= & \quad \{ \quad \text{arithmetic} \quad \} \\
 & (9 + 2{\times}\sqrt{14}) \quad R \quad (8 + 2{\times}\sqrt{15}) \\
= & \quad \{ \quad \text{addition is invertible and monotonic with respect to } R \quad \} \\
 & (1 + 2{\times}\sqrt{14}) \quad R \quad (2{\times}\sqrt{15}) \\
= & \quad \{ \quad \text{squaring is invertible and monotonic with respect to } R \quad \} \\
 & (57 + 4{\times}\sqrt{14}) \quad R \quad 60 \\
= & \quad \{ \quad \text{addition is invertible and monotonic with respect to } R \quad \} \\
 & (4{\times}\sqrt{14}) \quad R \quad 3 \\
= & \quad \{ \quad \text{squaring is invertible and monotonic with respect to } R \quad \} \\
 & 224 \quad R \quad 9 \\
= & \quad \{ \quad \text{conclusion} \quad \} \\
 & \text{R is} > \; .
\end{array}
$$

The calculation shows that the relation R is greater-than, that is, the number $\sqrt{2} + \sqrt{7}$ is greater than $\sqrt{3} + \sqrt{5}$.

2.5 For the teacher

Name the unknown The teacher must be sure the students understand that the unknown is an ordering relation. An idea is to make the analogy with equations; the

students should have no problem understanding that the variable x in an equation like $x + 3 = 5$ represents a number. Similarly, the teacher should introduce a variable for the relation:

$$(\sqrt{2} + \sqrt{7}) \quad R \quad (\sqrt{3} + \sqrt{5}) \quad .$$

The goal is to calculate R. It is important that the students understand that R can be one of less-than, equals or greater-than relation.

Understanding the problem in terms of symbols The teacher should ask the students why they cannot compare immediately the two expressions; they should understand that the problem is calculating the square roots. This motivates the exploration of symbol dynamics: is it possible to eliminate the occurrences of the square root operator? The idea is to introduce the following rule that is valid for positive a and b:

$$(2.5.1) \quad a^2 \; R \; b^2 \quad \equiv \quad a \; R \; b \quad .$$

We usually describe this rule as "squaring is invertible and monotonic with respect to R". We say that function f is monotonic with respect to an ordering R when:

$$a \; R \; b \; \Rightarrow \; f.a \;\; R \;\; f.b \quad .$$

In other words, the ordering determined by R is preserved by f. We also have the implication in the other direction, because the inverse of f is monotonic:

$$\begin{array}{ll} & f.a \;\; R \;\; f.b \\ \Rightarrow & \quad \{ \quad f^{-1} \text{ is monotonic with respect to } R \quad \} \\ & f^{-1}.f.a \;\; R \;\; f^{-1}.f.b \\ = & \quad \{ \quad f \text{ and } f^{-1} \text{ are inverse functions} \quad \} \\ & a \; R \; b \quad . \end{array}$$

Instantiating f with the square function and f^{-1} with the square root function, we can immediately conclude rule (2.5.1).

Also, the teacher may refer that addition is invertible and monotonic with respect to R, i.e., for all a, b, and k:

$$(a+k) \; R \; (a+k) \quad \equiv \quad a \; R \; b \quad .$$

Once these rules are introduced, the calculation becomes a straightforward exercise in algebraic manipulation. We recommend the teacher to explain these rules in terms of

symbol dynamics: for example, whenever we have an expression of the shape $a \ R \ b$, we can replace it by another expression where both sides are squared, i.e., $a^2 \ R \ b^2$. (Alternatively, whenever we have an expression of the shape $a^2 \ R \ b^2$, we can replace it by another expression where the squares are removed.) As the calculation proceeds, the teacher should also explain the proof format.

Explaining the proof format If the teacher is using the scenario to introduce the format of calculational proofs, we recommend the emphasis on the importance of the hints and on the advantages of the format.

Discussing generalisations If the opportunity arises, we recommend the teacher to present the generalisation shown in exercise 2.6.2. It can also be given as homework.

2.5.1 Questions that the teacher should ask

Name the unknown:

- *In the problems we have seen before, you have introduced variables for the unknowns. Can you apply the same strategy here? If so, what is the unknown? What would a variable represent in this problem?*

  The goal is to help the students understand that the unknown in this problem is an ordering relation.

Understanding the problem in terms of symbols:

- *Why can't you compare the two expressions immediately? In terms of symbols, what should you do? Do you know any rule that allows you to do that?*

  The goal is to help the students understand that the origin of the problem is the square root operator. If they don't know any rule to eliminate the square roots, the teacher can present and explain rule (2.5.1).

- *Do you understand why saying that a function is monotonic is the same as saying that the order is preserved after applying the function to both sides?*

  The goal is to help the students make the connection between the formulation of monotonicity and its meaning. The teacher may also ask the students if they think that multiplication by a negative number is monotonic. (They should realise that it is not; that is why both sides have to remain positive.)

2.5.2 Questions that the teacher should not ask

Name the unknown:

- *Can you use a variable for the ordering relation?*

  This question is obtrusive and it should occur naturally to the student. There is only one unknown, and that is what the students must name.

Understanding the problem in terms of symbols:

- *Can you see that the goal is to eliminate the square root operator?*

  The teacher should only ask this question if the students fail to see the symbol manipulations they have to perform. It is important to ask first what kind of manipulations they can do in order to get the answer.

2.5.3 Concepts that the teacher should introduce

Goal-directed investigations

Calculational proof

Monotonicity

2.6 Extensions and exercises

Exercise 2.6.1 (Warm-up) Determine the ordering relation between $\sqrt{3} + \sqrt{13}$ and $\sqrt{5} + \sqrt{11}$. Use the same style of calculation as before.

□

Exercise 2.6.2 (Generalisation) Explain how you could determine the ordering relation between $\sqrt{a} + \sqrt{b}$ and $\sqrt{c} + \sqrt{d}$.

□

Exercise 2.6.3 (Specialisation) Apply the method you devised in exercise 2.6.2 to solve the problem of section 2.2 and exercise 2.6.1.

□

2.7 Solutions to extensions and exercises

2.6.1 We can determine the ordering relation as before.

$$
\begin{array}{ll}
 & (\sqrt{3} + \sqrt{13}) \quad R \quad (\sqrt{5} + \sqrt{11}) \\
= & \quad \{ \quad \text{squaring is invertible and monotonic with respect to } R \quad \} \\
 & (\sqrt{3} + \sqrt{13})^2 \quad R \quad (\sqrt{5} + \sqrt{11})^2 \\
= & \quad \{ \quad \text{arithmetic} \quad \} \\
 & (16 + 2{\times}\sqrt{39}) \quad R \quad (16 + 2{\times}\sqrt{55}) \\
= & \quad \{ \quad \text{addition is invertible and monotonic with respect to } R \quad \} \\
 & (2{\times}\sqrt{39}) \quad R \quad (2{\times}\sqrt{55}) \\
= & \quad \{ \quad \text{squaring is invertible and monotonic with respect to } R \quad \} \\
 & (4{\times}39) \quad R \quad (4{\times}55) \\
= & \quad \{ \quad \text{arithmetic} \quad \} \\
 & 156 \quad R \quad 220 \\
= & \quad \{ \quad \text{conclusion} \quad \} \\
 & \text{R is} < .
\end{array}
$$

□

2.6.2 We can determine the ordering relation as before, but we need to ensure that the terms that occur in both sides remain positive.

$$
\begin{array}{ll}
 & (\sqrt{a} + \sqrt{b}) \quad R \quad (\sqrt{c} + \sqrt{d}) \\
= & \quad \{ \quad \text{squaring is invertible and monotonic with respect to } R \quad \} \\
 & (\sqrt{a} + \sqrt{b})^2 \quad R \quad (\sqrt{c} + \sqrt{d})^2 \\
= & \quad \{ \quad \text{arithmetic; } u := a{+}b \text{ , } v := a{\times}b, \, x := c{+}d \text{ and } y := c{\times}d \quad \} \\
 & (u + 2{\times}\sqrt{v}) \quad R \quad (x + 2{\times}\sqrt{y}) \\
= & \quad \{ \quad \text{addition is invertible and monotonic with respect to } R; \\
 & \qquad \text{to guarantee that both sides are positive, assume that } u \leq x; \\
 & \qquad \text{use } z := x{-}u \quad \} \\
 & (2{\times}\sqrt{v}) \quad R \quad (z + 2{\times}\sqrt{y})
\end{array}
$$

$=$ { squaring is invertible and monotonic with respect to R;
$p := z^2 + 4\times y$ and $q := z \times 2\times\sqrt{y}$ }
$(4\times v)\ \ R\ \ (p + 2\times q)$
$=$ { there are two cases }
if $4\times v < p \ \rightarrow\ R$ is $<$
$\Box\ p \leq 4\times v \ \rightarrow\ R$ is the same as in $(4\times v - p)^2\ \ R\ \ (4\times q^2)$
fi .

Note that we have assumed the condition $u \leq x$, which means that we may have to swap the terms of the relation to apply the result. This assumption is to guarantee that both sides are positive. Alternatively, we can subtract the minimum of u and x from both sides.
□

2.6.3 Let us start with exercise 2.6.1. Instantiating the variables with the concrete values, we get

$$a = 3 \ \wedge\ b = 13 \ \wedge\ c = 5 \ \wedge\ d = 11 \ .$$

From these values, we compute the value of the other variables:

$$u = 16 \ \wedge\ v = 39 \ \wedge\ x = 16 \ \wedge\ y = 55 \ \text{, and}$$

$$4\times v = 156 \ \wedge\ z = 0 \ \wedge\ p = z^2 + 4\times y = 220 \ .$$

Since $u \leq x$, we can safely apply the result obtained in exercise 2.6.2. Hence, and because $4\times v < p$, we conclude that R is $<$.

Let's now solve the problem of section 2.2. For this problem, we have to swap the terms, because $u > x$. That means that we are going to determine the relation R' in the expression

$$(\sqrt{3} + \sqrt{5})\ \ R'\ \ (\sqrt{2} + \sqrt{7}) \ .$$

Instantiating the variables with the concrete values yields

$$a = 3 \ \wedge\ b = 13 \ \wedge\ c = 5 \ \wedge\ d = 11 \ .$$

From these values, we compute the value of the other variables:

$$u = 8 \ \wedge\ v = 15 \ \wedge\ x = 9 \ \wedge\ y = 14 \ \text{, and}$$

$$4\times v = 60 \quad \wedge \quad z = 1 \quad \wedge \quad p = z^2 + 4\times y = 57 \quad \wedge \quad q = 2\times\sqrt{14} \quad .$$

Now, since $p < 4\times v$, the relation R' is the same as in

$$(60 - 57)^2 \quad R' \quad (4\times(2\times\sqrt{14})^2) \quad .$$

Evaluating both terms we get

$$9 \quad R' \quad 224 \quad ,$$

from where we deduce that R' is $<$. Since we have swapped the original terms, we conclude that R is $>$.

□

2.8 Further reading

This problem was taken from the chapter 3 of [Bac03]. We recommend that chapter as an introduction to calculational proofs.

Also, in the chapter 8 of [Pol81], Pólya discusses this problem. In that chapter, Pólya uses the terms "working backward [from the goal]", "regressive planning", and "analysis" for what we call "goal-oriented".

SCENARIO 3

The Island of Knights and Knaves

3.1 Brief description and goals

This teaching scenario is about a type of logic puzzle where the algebraic properties of equivalence play a central role. We adopt a calculational approach: instead of solving the puzzle by performing case analysis, we calculate its solution. The scenario can be used as an introduction to calculational logic and to Boolean equality.

3.2 Problem

The island of knights and knaves has two types of inhabitants: 'knights', who always tell the truth; and 'knaves', who always lie.

Someone tells you that the island has gold and you go there to collect it. You find a native at a fork in the road and you want to determine whether the gold can be found by following the left or the right fork. Which question do you have to formulate such that the reply will be "yes" if the left fork should be followed and "no" if the right fork should be followed?

3.3 Prerequisites

Familiarity with the calculational proof style can be helpful.

3.4 Resolution and notes

The solution we present here is the same as the one shown in [Bac03, chapter 5]. Suppose A is the proposition 'person A is a knight' and suppose that person A makes a statement S[1]. Then A is true is the same as S is true. That is,

(3.4.1) $A \equiv S$.

This is the most important insight for solving puzzles about the island. For example, if A says 'I am the same type as B', then A's statement is the same as

$A \equiv B$.

Replacing S in (3.4.1) by this statement, we get $A \equiv (A \equiv B)$, which by associativity and reflexivity of equivalence simplifies to B. So, from this statement, we can infer that B is a knight, but nothing about A.

Similarly, if a native is asked a yes/no question Q, then the response to the question is $A \equiv Q$. That is, the response will be 'yes' if the native 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 'is B a knight?' A will respond 'yes' if they are both the same type, otherwise 'no'. That is, A's response is 'yes' or 'no' depending on the truth or falsity of $A \equiv B$.

The goal of the problem is to construct a question Q, to which the native responds 'yes' if the left fork leads to the gold or 'no' otherwise. Let G be the proposition 'the left fork leads to the gold'. We require that G equivales the response to the question is yes. But the response to the question is yes is the same as $A \equiv Q$. So, we require that $G \equiv (A \equiv Q)$. Now, by associativity and symmetry of equivalence, we have $Q \equiv (G \equiv A)$. The question is thus: *is the statement that the left fork leads to the gold equivalent to you being a knight?*

3.5 Notes for the teacher

Key observation We suggest the teacher to introduce first the key property (3.5.1). So, suppose A is the proposition 'person A is a knight' and suppose A makes a statement S. *Then A is true is the same as S is true.* That is, using $\equiv$ to denote equivalence (i.e., equality of propositions),

(3.5.1) $A \equiv S$.

[1] Note that we overload A to denote both 'person A' and the proposition that the 'person A is a knight'. This means that we need to be careful and to rely on the context when using the symbol A.

This is the most important insight for solving puzzles about the island. The teacher has to be sure that the students understand what (3.5.1) means: A and S are equal, that is, they always have the same value. In other words, saying that A is a knight is the same as saying that A's statements are true, and saying that A is a knave is the same as saying that A's statements are false. (Note that we overload A to denote both 'person A' and the proposition 'person A is a knight'. This means that we need to be careful and to rely on the context when using the symbol A. Alternatively, the teacher may want to introduce different symbols.)

Properties of equivalence Because the most important insight is based on equivalence, we suggest the teacher to elaborate on its algebraic properties. First, $\equiv$ is associative, which means that for all p, q, and r, we have

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

This means that we can write $p \equiv q \equiv r$ without ambiguity (however we bracket the expression, its value does not change).

Please note that when we have continued equivalences without any brackets, we need to be careful with the interpretation we use. The conventional reading of continued equalities is based on transitivity, that is, when we write the expression

$$a = b = c$$

we usually mean that a, b, and c are all equal. More formally, we usually mean that

$$a = b \ \wedge \ b = c \quad .$$

However, in the Boolean domain, the associative reading and the transitive reading can conflict. For example, suppose that we have the expression

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

If we read it associatively, we can simplify it as follows:

$$\begin{array}{ll} & \mathsf{true} \equiv \mathsf{false} \equiv \mathsf{false} \\ = & \quad \{ \quad \text{associativity} \quad \} \\ & \mathsf{true} \equiv (\mathsf{false} \equiv \mathsf{false}) \\ = & \quad \{ \quad \mathsf{false} \text{ is equivalent to false, so we can simplify} \\ & \qquad (\mathsf{false} \equiv \mathsf{false}) \text{ to } \mathsf{true} \quad \} \end{array}$$

$$
\begin{array}{ll}
 & \mathsf{true} \equiv \mathsf{true} \\
= & \quad \{ \quad \mathsf{true} \text{ is equivalent to } \mathsf{true}\text{, so we can simplify} \\
 & \qquad\quad (\mathsf{true} \equiv \mathsf{true}) \text{ to } \mathsf{true} \quad \} \\
 & \mathsf{true} \ .
\end{array}
$$

On the other hand, if we read it using transitivity, we simplify it differently:

$$
\begin{array}{ll}
 & \mathsf{true} \equiv \mathsf{false} \equiv \mathsf{false} \\
= & \quad \{ \quad \text{transitive reading} \quad \} \\
 & (\mathsf{true} \equiv \mathsf{false}) \ \wedge \ (\mathsf{false} \equiv \mathsf{false}) \\
= & \quad \{ \quad (\mathsf{true} \equiv \mathsf{false}) \text{ is false and } (\mathsf{false} \equiv \mathsf{false}) \text{ is true} \quad \} \\
 & \mathsf{false} \ \wedge \ \mathsf{true} \\
= & \quad \{ \quad \mathsf{false} \ \wedge \ p \text{ is false, for all } p \quad \} \\
 & \mathsf{false} \ .
\end{array}
$$

The conclusion is thus that different readings of continued equivalences may yield different values! To avoid ambiguities, when we use the conventional equality symbol $=$, we read the expression in a transitive way. To read the expression associatively, we use the less conventional symbol $\equiv$ and we call it "equivales". We suggest the teacher to explain the difference between the two readings and to introduce equivales, since all the educational material on logic contained in this thesis explores the associativity of equivalence.

Another important property of equivalence is reflexivity, which states that for all p, we have

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

Note that, thanks to associativity, reflexivity captures two rules:

$$(p \equiv p) \equiv \mathsf{true} \quad \text{, and}$$

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

In terms of symbols, the first rule means that whenever we have an expression of the shape $p \equiv p$ we can replace by true. The second rule means that whenever we have true involved in an equivalence, we can simply remove it.

Now, from associativity and reflexivity, we can prove that equivalence is symmetric, that is:

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

The proof is very simple and illustrative of the calculational format that we use for teaching logic:

$$\begin{array}{ll} & (p \equiv q) \equiv (q \equiv p) \\ = & \quad \{ \quad \text{associativity} \quad \} \\ & p \equiv (q \equiv q) \equiv p \\ = & \quad \{ \quad \text{reflexivity and associativity} \quad \} \\ & (p \equiv \mathsf{true}) \equiv p \\ = & \quad \{ \quad \text{reflexivity} \quad \} \\ & p \equiv p \\ = & \quad \{ \quad \text{reflexivity} \quad \} \\ & \mathsf{true} \ . \end{array}$$

Some simple examples Before solving the proposed problem, we suggest that the teacher shows some simple examples where the properties of equivalence are used. For example, if A says 'I am the same type as B', then A's statement is the same as

$$A \equiv B \quad .$$

Replacing S in (3.5.1) by this statement, we get $A \equiv (A \equiv B)$. We can calculate the value of this expression as follows:

$$\begin{array}{ll} & A \equiv (A \equiv B) \\ = & \quad \{ \quad \text{associativity} \quad \} \\ & (A \equiv A) \equiv B \\ = & \quad \{ \quad \text{reflexivity} \quad \} \\ & \mathsf{true} \equiv B \\ = & \quad \{ \quad \text{reflexivity} \quad \} \\ & B \ . \end{array}$$

So, from this statement, we can infer that B is a knight, but nothing about A.

Yes/No questions The problem is about formulating a yes/no question and we can use the same reasoning as above. That is, if a native is asked a yes/no question Q, then the response to the question is $A \equiv Q$. (Note that we are associating 'yes' to true and 'no' to false.) The response will be 'yes' if the native 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 'is B a knight?' A will respond 'yes' if they are both the same type, otherwise 'no'. That is, A's response is 'yes' or 'no' depending on the truth or falsity of $A \equiv B$. In other words, the response to the question is yes is the same as $A \equiv Q$.

The teacher should ask what is the goal of problem, so that the students' attention gets redirected to the problem statement. The goal of the problem is to construct a question Q, to which the native responds 'yes' if the left fork leads to the gold or 'no' otherwise. Let G be the proposition 'the left fork leads to the gold'. We require that G equivales the response to the question is yes. But the response to the question is yes is the same as $A \equiv Q$. So, we require that $G \equiv (A \equiv Q)$. Now, by associativity and symmetry of equivalence, we have $Q \equiv (G \equiv A)$. The question is thus: *is the statement that the left fork leads to the gold equivalent to you being a knight?*

The teacher may analyse the different possible cases to convince the students that the question is indeed correct. However, we suggest that the students practise calculational solutions to different problems on knights and knaves (see section 3.6 for more exercises). We believe that the calculational approach is superior to the conventional approach by case analysis, since it is more concise and can be used as a platform to learn algebraic rules that are useful to solve other problems on logic.

Generalisations and other examples Another simple example is when A says 'I am a knight'. In this case, A's statement is A, so we want to simplify the expression $A \equiv A$. By reflexivity, this simplifies to true, which means that we cannot say anything about A's type. (Similarly, if you ask a native 'Are you a knight?', the answer will always be 'yes'.)

Negation is a unary operator that can also be useful for solving puzzles. If p is a Boolean expression, $\neg p$ represents the negation of p and is read 'not p'. The law governing negation is

$$\neg p \equiv p \equiv \mathsf{false} \quad .$$

The relevance of negation is that if we want to express that A is a knave, we can simply write $\neg A$.

If A says 'I am a knave', then A's statement is $\neg A$, so we want to simplify the expression $A \equiv \neg A$. By negation, this simplifies to false, which means that A could never have said that. (Similarly, if you ask a native 'Are you a knave?', the answer will always be 'no'.)

Section 3.6 contains more exercises that can be used in lectures or as homework.

3.5.1 Questions that the teacher should ask

Key observation

- *Do you understand the problem?*

  The teacher has to be sure that the students understand what is required.

- *Suppose person A is a knight. What can you say about A's statements? What if A is a knave?*

  The students should understand that the statements of knights are always true, whilst the statements of knaves are always false. This motivates the introduction of the key property (3.5.1).

- *Do you understand property (3.5.1)?*

  Property (3.5.1) is the key to solve this type of puzzle. It is important that the students understand it.

Properties of equivalence

- *Do you understand what associativity means?*

  Although we believe that manipulating expressions without interpreting their meaning brings many advantages, we think that it is important for students to understand what is the meaning of some rules. In particular, we suggest that the teacher explains the meaning in terms of symbols (for associativity, it means that we can bracket a continued expression as we wish). The teacher may also give examples of other associative operators, like addition and multiplication.

  The same question applies to reflexivity and symmetry.

- *When we write something like $a = b = c$, what do we usually mean?*

  We expect the students to say that the conventional reading of $a = b = c$ is transitive, that is, a, b, and c are all equal. This question can be followed by the simplification of $\mathsf{true} \equiv \mathsf{false} \equiv \mathsf{false}$ using the transitive reading. Then the teacher can show how the associative reading yields a different result.

- *What is the difference between $p = q = r$ and $p \equiv q \equiv r$?*

  The students should understand that whenever we have a continued equivalence with the symbol $\equiv$, we read it associatively. If the symbol used is $=$, we read it transitively.

Yes/No Questions

- *What is our goal?*

  After explaining some of the properties of equivalence and showing some examples, the teacher should redirect the students' attention to the goal of the problem: to formulate a particular yes/no question. In particular, the teacher may say that we want to calculate the question, instead of guessing it.

- *G is the proposition "the left fork leads to the gold". What is the requirement on G?*

  The teacher should allow the students to carefully articulate the requirement on G. We require that G equivales the response to the question is yes. But the response to the question is yes is the same as $A \equiv Q$. Therefore, G equivales $A \equiv Q$ and from here, we can calculate Q. Once the expression $G \equiv (A \equiv Q)$ is achieved, the teacher can ask the students what is the question that we have to ask.

3.5.2 Questions that the teacher should not ask

Some simple examples

- *[For all examples] What are the possible cases?*

  The idea of this scenario is to illustrate that we can calculate the solutions to logic puzzles and avoid case analysis. Therefore, the teacher should not ask the students to analyse the possible cases. Although initially the students will have the tendency to analyse all the possible cases, we observe that after some practise, most of them will try a calculational approach.

Yes/No Questions

- *G is the proposition "the left fork leads to the gold". Can you see that the requirement is $G \equiv (A \equiv Q)$?*

  The teacher should allow the students to think and try to articulate the requirement on G. If they are unsuccessful, the teacher may help with hints.

3.5.3 Concepts that the teacher should introduce

Calculational logic

Boolean equality

Associativity

Reflexivity

Symmetry

Negation

3.6 Extensions and exercises

The following exercises (and solutions) are taken from [Bac03, chapter 5].

Exercise 3.6.1 (Warm-up) You ask one of the natives, A, whether there is gold on the island. He makes the following response: 'There is gold on the island equivales I am a knight'. Can it be determined whether A is a knight or a knave? Can it be determined whether there is gold on the island?

□

Exercise 3.6.2 (Interrogation) What single question allows you to determine whether A is a knight?

□

Exercise 3.6.3 (Interrogation) What single question should you ask A to determine whether B is a knight?

□

Exercise 3.6.4 (Negation) Negation is a unary operator that can also be useful for solving puzzles. If p is a Boolean expression, $\neg p$ represents the negation of p and is read 'not p'. The law governing negation is

$$\neg p \equiv p \equiv \mathsf{false} \quad .$$

The relevance of negation is that if we want to express that A is a knave, we can simply write $\neg A$. Now, suppose there are two natives, A and B. A says 'B is a knight is the same as I am a knave'. What can you determine about A and B?

□

Exercise 3.6.5 (Negation) What single question should you ask A to determine whether B is a knave?

□

Exercise 3.6.6 (Negation) What single question should you ask A to determine whether A and B are different types?

□

3.7 Solutions to extensions and exercises

3.6.1 Let G denote the proposition 'There is gold on the island'. A's statement is $A \equiv G$. So, recalling (3.5.1), what we are given is

$$A \equiv A \equiv G \quad .$$

By associativity and reflexivity, this is equivalent to G. Therefore, we can conclude that there is gold in the island. However, we cannot determine whether A is a knight or a knave.
□

3.6.2 Let Q be the question. Asking the question Q will produce the response $A \equiv Q$, which we require to be A. So, we require that $A \equiv A \equiv Q$, which, by reflexivity, simplifies to Q. Therefore, we should ask A to confirm or deny any true statement (for example, 'is 0 equals to 0'?).
□

3.6.3 Let Q be the question. Asking the question Q will produce the response $A \equiv Q$, which we require to be B. So, we require $B \equiv A \equiv Q$. Therefore, we should ask whether A and B are the same type.
□

3.6.4 A's statement is $B \equiv \neg A$. So, what we are given is

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

This simplifies to $\neg B$ as follows.

$$
\begin{array}{ll}
& A \equiv B \equiv \neg A \\
= & \quad \{ \quad \text{associativity and symmetry} \quad \} \\
& A \equiv \neg A \equiv B \\
= & \quad \{ \quad \text{negation law: } \neg p \equiv p \equiv \mathsf{false} \text{ with } p := A \quad \} \\
& \mathsf{false} \equiv B \\
= & \quad \{ \quad \text{symmetry and negation law: } \neg p \equiv p \equiv \mathsf{false} \text{ with } p := B \quad \} \\
& \neg B \ .
\end{array}
$$

So, B is a knave and we cannot say whether A is a knight or a knave.
□

3.6.5 Let Q be the question. Asking the question Q will produce the response $A \equiv Q$, which we require to be $\neg B$. So, we require $\neg B \equiv A \equiv Q$. Therefore, we should ask whether A and B are of different types.
□

3.6.6 Let Q be the question. Asking the question Q will produce the response $A \equiv Q$, which we require to be $\neg B \equiv A$. So, we require $\neg B \equiv A \equiv A \equiv Q$. By associativity and reflexivity of equivalence, this simplifies to $\neg B \equiv Q$. Therefore, we should ask A whether B is a knave.
□

3.8 Further reading

The problem, exercises, and solutions were taken from chapter 5 of [Bac03]. Moreover, chapter 7 of the same book contains examples where implication is used. In our view, these two chapters constitute an excellent introduction to calculational logic.

SCENARIO 4

Portia's Casket

4.1 Brief description and goals

This scenario is about a type of logic puzzle that can be reduced to a system of simultaneous equations on Booleans. It can be used to introduce logical implication and to practise calculational logic.

4.2 Problem

In an abridged version of Shakespeare's *Merchant of Venice*, Portia had two caskets: gold and silver. Inside one of these caskets, Portia had put her portrait, and on each was an inscription. Portia explained to her suitor that each inscription could be either true or false but, on the basis of the inscriptions, he was to choose the casket containing the portrait. If he succeeded, he could marry her.

The inscriptions were:

Gold: *The portrait is in this casket.*
Silver: *If the inscription on the gold casket is true, this inscription is false.*

Which casket contained the portrait? What can we deduce about the inscriptions?

4.3 Prerequisites

Algebraic properties of equality. Knowledge of solving systems of linear equations on numbers can be useful.

4.4 Resolution and notes

The goal is to determine the casket that contains the portrait, so the first step in modelling the problem is to introduce the two following variables:

pg	the portrait is in the gold casket
ps	the portrait is in the silver casket .

We want to calculate the values of these variables. Clearly, the portrait is in only one of the caskets. As a result, only one of pg and ps is true. So, an equation given by the problem statement is:

$$pg \equiv \neg ps \quad .$$

Also, because we need to model what the inscriptions say, we introduce the following variables:

ig	the inscription on the gold casket is true
is	the inscription on the silver casket is true .

The inscription on the gold casket is true whenever the portrait is in the gold casket. So, we have the equation:

$$ig \equiv pg \quad .$$

The inscription on the silver casket is true whenever the implication $ig \Rightarrow \neg is$ is true. Therefore, we also have the equation:

$$is \;\equiv\; ig \Rightarrow \neg is \quad .$$

The goal is to determine the values of the variables pg and ps knowing that the three equations shown above are simultaneously true. That is, we know that the following conjunction is true:

$$(pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge (is \;\equiv\; ig \Rightarrow \neg is) \quad .$$

Using Boolean algebra, we can simplify the conjunction as follows:

$$\begin{array}{ll} & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge (is \;\equiv\; ig \Rightarrow \neg is) \\ = & \quad \{ \quad \text{definition of } \Rightarrow, \text{ i.e., } ig \Rightarrow \neg is \;\equiv\; \neg is \;\equiv\; \neg is \vee ig; \\ & \qquad\quad \text{associativity} \quad \} \end{array}$$

$$
\begin{array}{ll}
 & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge ((is \equiv \neg is) \;\equiv\; \neg is \vee ig) \\
= & \quad \{ \quad \text{negation (twice), i.e., } is \equiv \neg is \equiv \mathsf{false} \quad \} \\
 & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge \neg(\neg is \vee ig) \\
= & \quad \{ \quad \text{De Morgan, i.e., } \neg(\neg is \vee ig) \;\equiv\; is \wedge \neg ig \quad \} \\
 & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge is \wedge \neg ig \\
= & \quad \{ \quad \text{reflexivity, negation, and Leibniz} \quad \} \\
 & (pg \equiv \neg ps) \wedge (\mathsf{false} \equiv pg) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{false}) \\
= & \quad \{ \quad \text{Leibniz and negation} \quad \} \\
 & (\mathsf{true} \equiv ps) \wedge (\mathsf{false} \equiv pg) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{false}) \quad .
\end{array}
$$

We conclude that the portrait is in the silver casket, that the inscription on the silver casket is true, and that the inscription on the gold casket is false.

4.5 Notes for the teacher

Introduction to the problem We believe that the best way to introduce logic puzzles like this one, where the goal is to solve simultaneous equations on Booleans, is by analogy with simultaneous equations on numbers. For example, consider the following problem:

> Suppose Ben is twice as old as Anne, but two years ago, Ben was three times as old as Anne. How old are Ben and Anne?

In our experience, most secondary-school students know how to solve this problem. Their first step is to model the problem as the two simultaneous equations

$$b = 2\times a \quad \wedge \quad b-2 = 3\times(a-2) \quad ,$$

where a and b denote, respectively, Ben and Anne's ages. Then, and using the calculational proof format, most of them know how to calculate the correct solution:

$$
\begin{array}{ll}
 & b = 2\times a \quad \wedge \quad b-2 = 3\times(a-2) \\
= & \quad \{ \quad \text{replace } b \text{ by } 2\times a \quad \} \\
 & b = 2\times a \quad \wedge \quad 2\times a - 2 \;=\; 3\times(a-2) \\
= & \quad \{ \quad \text{arithmetic} \quad \}
\end{array}
$$

$$
\begin{array}{ll}
 & b = 2 \times a \ \wedge \ 4 = a \\
= & \quad \{ \quad \text{replace } a \text{ by } 4 \quad \} \\
 & b = 8 \ \wedge \ 4 = a \ .
\end{array}
$$

Although most students would solve the logic puzzle by case analysis, it can be solved just like we solved the problem on Ben and Anne's ages. The main difference is the domain: whilst the problem above was about solving simultaneous equations on natural numbers, this puzzle is about solving simultaneous equations on Booleans. As a result, instead of using the algebra of numbers, we use the algebra of Booleans. One of the main algebraic rules, the one that allows the substitution of equals for equals — used in the first and third steps of the calculation above — is essentially the same in the Boolean domain:

$$
\begin{array}{ll}
 & (p \equiv q) \wedge f.p \\
= & \quad \{ \quad \text{substitution of equals for equals; we also call} \\
 & \qquad \text{this rule "Leibniz"} \quad \} \\
 & (p \equiv q) \wedge f.q \ .
\end{array}
$$

In words, if we have that p and q are equivalent, we can replace p by q (and vice-versa) in a given context f. Another two rules that concern Booleans and are commonly used are reflexivity

$$p \equiv p \equiv \mathsf{true} \quad ,$$

and negation

$$\neg p \equiv p \equiv \mathsf{false} \quad .$$

A concrete example combining the rules of reflexivity and Leibniz is:

$$
\begin{array}{ll}
 & (p \equiv q) \wedge p \\
= & \quad \{ \quad \text{reflexivity} \quad \} \\
 & (p \equiv q) \wedge (p \equiv \mathsf{true}) \\
= & \quad \{ \quad \text{Leibniz} \quad \} \\
 & (\mathsf{true} \equiv q) \wedge (p \equiv \mathsf{true}) \\
= & \quad \{ \quad \text{reflexivity (twice)} \quad \} \\
 & q \wedge p \ .
\end{array}
$$

We suggest the teacher to show the students an example on using the rule of Leibniz with Booleans.

Model the problem The first step in the solution is to model the problem. The goal is to determine the casket that contains the portrait, so we introduce the two following variables:

pg	the portrait is in the gold casket
ps	the portrait is in the silver casket .

We want to calculate the values of these variables. The teacher can make the analogy with the problem on numbers, where we also have introduced variables that represent the goal (the ages of Ben and Anne).

We suggest the teacher to ask the students if they can identify any relation between pg and ps. The idea is to introduce the equation stating that the portrait is in only one of the caskets, that is, that only one of pg and ps is true. The equation is:

$$pg \equiv \neg ps \quad .$$

Also, because we need to model what the inscriptions say, we introduce the following variables:

ig	the inscription on the gold casket is true
is	the inscription on the silver casket is true .

We now have to model what the inscriptions say. The inscription on the gold casket is true whenever the portrait is in the gold casket. We suggest the teacher asks the students what is the equation that models this. The goal is to introduce the equation:

$$ig \equiv pg \quad .$$

The inscription on the silver casket may require the introduction of implication. Whenever we have a proposition of the form "If a then b", we can model it formally as $a \Rightarrow b$. Implication admits two definitions:

$$a \Rightarrow b \equiv a \equiv a \wedge b \quad \text{, and}$$

$$a \Rightarrow b \equiv b \equiv a \vee b \quad .$$

Depending on the context, we use the definition that is more convenient. For example, if we have $b \equiv a \Rightarrow b$, we would choose the second definition because it introduces another b; this would allow to eliminate the new sub-expression $b \equiv b$ by reflexivity.

Now, the inscription on the silver casket is true whenever the implication $ig \Rightarrow \neg is$ is true. Again, we suggest the teacher asks the students what is the equation that models this. The equation is:

$$is \equiv ig \Rightarrow \neg is \quad .$$

The goal is to determine the values of the variables pg and ps knowing that the three equations shown above are simultaneously true. That is, we know that the following conjunction is true:

$$(pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge (is \equiv ig \Rightarrow \neg is) \quad .$$

Again, we suggest the teacher to make the analogy with the problem on numbers.

Solution Now, to solve the problem, we use Boolean algebra to simplify the conjunction. For each step, we suggest the teacher to ask the students what to do. Although there are several possibilities for each step, we show the steps we think are more relevant. For example, in the first step, the most relevant rule to apply is the definition of implication:

$$\begin{array}{ll} & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge (is \equiv ig \Rightarrow \neg is) \\ = & \quad \{ \quad \text{definition of } \Rightarrow, \text{ i.e., } ig \Rightarrow \neg is \equiv \neg is \equiv \neg is \vee ig; \\ & \qquad \text{associativity} \quad \} \\ & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge ((is \equiv \neg is) \equiv \neg is \vee ig) \end{array}$$

We have chosen the definition involving disjunction, because it allows to introduce the subexpression $is \equiv \neg is$. We simplify it in the next step:

$$\begin{array}{ll} & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge ((is \equiv \neg is) \equiv \neg is \vee ig) \\ = & \quad \{ \quad \text{negation (twice), i.e., } is \equiv \neg is \equiv \mathsf{false} \quad \} \\ & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge \neg(\neg is \vee ig) \end{array}$$

Now, we have a disjunction negated. There is a rule involving negation and disjunction called De Morgan:

$$\neg(a \vee b) \equiv \neg a \wedge \neg b \quad .$$

We can use this rule to simplify the third conjunct:

$$
\begin{array}{ll}
 & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge \neg(\neg is \vee ig) \\
= & \quad \{ \quad \text{De Morgan, i.e., } \neg(\neg is \vee ig) \equiv is \wedge \neg ig \quad \} \\
 & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge is \wedge \neg ig
\end{array}
$$

Using reflexivity and negation we can rewrite the two last conjuncts as:

$$
\begin{array}{ll}
 & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge is \wedge \neg ig \\
= & \quad \{ \quad \text{reflexivity and negation} \quad \} \\
 & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{false})
\end{array}
$$

We know that ig is false, so we can rewrite the second conjunct using substitution of equals for equals (Leibniz):

$$
\begin{array}{ll}
 & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{false}) \\
= & \quad \{ \quad \text{Leibniz} \quad \} \\
 & (pg \equiv \neg ps) \wedge (\mathsf{false} \equiv pg) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{false})
\end{array}
$$

Finally, from the second conjunct, we know that pg is false; we can rewrite the first conjunct using the rules of Leibniz and negation:

$$
\begin{array}{ll}
 & (pg \equiv \neg ps) \wedge (\mathsf{false} \equiv pg) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{false}) \\
= & \quad \{ \quad \text{Leibniz and negation} \quad \} \\
 & (\mathsf{true} \equiv ps) \wedge (\mathsf{false} \equiv pg) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{false}) \quad .
\end{array}
$$

We conclude that the portrait is in the silver casket, that the inscription on the silver casket is true, and that the inscription on the gold casket is false.

We suggest the teacher to conclude with the advantages of the calculational approach. In our experience, most people would agree that using case analysis in the problem on Ben and Anne's ages is not a good idea, because case analysis is very specific and does not scale well to more complicated problems; most people would agree that it is more important to teach the students how to solve general systems of simultaneous equations. Moreover, it is important to remark that when the students are solving such problems on numbers, they are not interpreting the formulae.

4.5.1 Questions that the teacher should ask

Introduction to the problem

- *Do you understand the problem?*

  The teacher has to be sure that the students understand what is required.

- *Have you ever solved any related problem?*

  The goal of this question is to introduce the analogy with simultaneous equations on numbers.

- *[After showing the problem on Ben and Anne's ages] How would you solve this problem?*

  This question is useful to know if the students know how to solve simultaneous equations on numbers.

- *Can you see how Leibniz rule is applied in the Boolean domain?*

  It is important that the students understand Leibniz rule.

Model the problem

- *Which variables should we introduce?*

  It is important to discuss with the students which elements of the problem should be named.

- *What is the relation between pg and ps? How can we express it formally?*

  The goal of this question is to let the students think about the equations. We believe it is better if all the equations are introduced by the students.

Solution

- *[At each step of the calculation] What should we do now?*

  It is important to make the students think about the possible rules that can be applied at each step. If they do not suggest any property, we recommend the teacher to suggest symbol manipulations (e.g. "I think it would be a good idea to remove the implication. How can we do that?").

4.5.2 Questions that the teacher should not ask

Model the problem

- *Can we model the relation between these variables as...*

  We think the teacher should not suggest any equation. Instead, if the students do not suggest any equations, we recommend the teacher to give some clues.

Solution

- *[At each step of the calculation] Can we use property X?*

  We think the teacher should not suggest any properties. If the students do not suggest any properties to be used, the teacher can suggest symbol manipulations, rather than properties (e.g., "It would be useful to eliminate the implication").

4.5.3 Concepts that the teacher should introduce

Substitution of equals for equals

Implication

Negation

De Morgan rules

4.6 Extensions and exercises

Exercise 4.6.1 (Variation) Suppose now that the inscription on the silver casket was the following:

> *"The inscription on the gold casket is true if this inscription is true".*

In this case, which casket contained the portrait? And what can we deduce about the inscriptions?

□

Exercise 4.6.2 (Variation) Suppose now that the inscription on the silver casket was the following:

> *"The inscription on the gold casket is false if this inscription is true".*

In this case, which casket contained the portrait? And what can we deduce about the inscriptions?

□

Exercise 4.6.3 (Variation) Suppose now that the inscription on the silver casket was the following:

> *"If the inscription on the gold casket is false, this inscription is false".*

In this case, which casket contained the portrait? And what can we deduce about the inscriptions?

□

Exercise 4.6.4 (Variation) Suppose now that the inscriptions on the silver and gold caskets were the following:

> Gold: *The portrait is in the gold casket if the inscription on the silver casket is true.*
> Silver: *If the inscription on the gold casket is true, this inscription is false.*

In this case, which casket contained the portrait? And what can we deduce about the inscriptions?

□

4.7 Solutions to extensions and exercises

4.6.1 The inscription on the silver casket is true whenever the implication $ig \Leftarrow is$ is true. Therefore, the equation that models the inscription on the silver casket is:

$$is \equiv ig \Leftarrow is \quad .$$

The calculation is:

$$\begin{array}{ll} & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge (is \equiv ig \Leftarrow is) \\ = & \quad \{ \quad \text{definition of } \Leftarrow \text{ and associativity} \quad \} \\ & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge ((is \equiv is) \equiv is \wedge ig) \\ = & \quad \{ \quad \text{reflexivity (three times)} \quad \} \\ & (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{true}) \\ = & \quad \{ \quad \text{Leibniz} \quad \} \\ & (pg \equiv \neg ps) \wedge (\mathsf{true} \equiv pg) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{true}) \end{array}$$

$$
\begin{array}{ll}
= & \{ \quad \text{Leibniz and negation} \quad \} \\
& (\mathsf{false} \equiv ps) \wedge (\mathsf{true} \equiv pg) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{true}) \quad .
\end{array}
$$

We conclude that the portrait is in the gold casket and that both inscriptions are true.
□

4.6.2 The inscription on the silver casket is true whenever the implication $\neg ig \Leftarrow is$ is true. Therefore, the equation that models the inscription on the silver casket is:

$$is \equiv \neg ig \Leftarrow is \quad .$$

The calculation is:

$$
\begin{array}{ll}
& (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge (is \equiv \neg ig \Leftarrow is) \\
= & \{ \quad \text{definition of } \Leftarrow \text{ and associativity} \quad \} \\
& (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge ((is \equiv is) \equiv is \wedge \neg ig) \\
= & \{ \quad \text{reflexivity (twice) and negation} \quad \} \\
& (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{false}) \\
= & \{ \quad \text{Leibniz} \quad \} \\
& (pg \equiv \neg ps) \wedge (\mathsf{false} \equiv pg) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{false}) \\
= & \{ \quad \text{Leibniz and negation} \quad \} \\
& (\mathsf{true} \equiv ps) \wedge (\mathsf{false} \equiv pg) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{false}) \quad .
\end{array}
$$

We conclude that the portrait is in the silver casket, that the inscription on the silver casket is true, and that the inscription on the gold casket is false.

Note that, by the contrapositive rule, we know that $\neg ig \Leftarrow is$ is the same as $ig \Rightarrow \neg is$. Therefore, this exercise is the same as the problem solved in the scenario.
□

4.6.3 The inscription on the silver casket is true whenever the implication $\neg ig \Rightarrow \neg is$ is true. Therefore, the equation that models the inscription on the silver casket is:

$$is \equiv \neg ig \Rightarrow \neg is \quad .$$

If we use the contrapositive rule, this problem is the same as exercise 4.6.1:

$$(pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge (is \equiv \neg ig \Rightarrow \neg is)$$

$$
\begin{array}{ll}
= & \{ \quad \text{contrapositive} \quad \} \\
& (pg \equiv \neg ps) \wedge (ig \equiv pg) \wedge (is \equiv ig \Leftarrow is) \\
= & \{ \quad \text{exercise } 4.6.1 \quad \} \\
& (\mathsf{false} \equiv ps) \wedge (\mathsf{true} \equiv pg) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{true}) \quad .
\end{array}
$$

We conclude that the portrait is in the gold casket and that both inscriptions are true. □

4.6.4 The inscription on the gold casket is true whenever the implication $pg \Leftarrow is$ is true and the inscription on the silver casket is true whenever the implication $ig \Rightarrow \neg is$ is true. Therefore, the equations that model the inscriptions are:

$$ig \equiv pg \Leftarrow is \quad \text{, and}$$

$$is \;\equiv\; ig \Rightarrow \neg is \quad .$$

We can calculate the solution to the puzzle as follows:

$$
\begin{array}{ll}
& (pg \equiv \neg ps) \wedge (ig \equiv pg \Leftarrow is) \wedge (is \;\equiv\; ig \Rightarrow \neg is) \\
= & \{ \quad \text{definition of } \Rightarrow, \text{ i.e., } ig \Rightarrow \neg is \;\equiv\; \neg is \;\equiv\; \neg is \vee ig; \\
& \quad \text{associativity} \quad \} \\
& (pg \equiv \neg ps) \wedge (ig \equiv pg \Leftarrow is) \wedge ((is \equiv \neg is) \;\equiv\; \neg is \vee ig) \\
= & \{ \quad \text{negation (twice), i.e., } is \equiv \neg is \equiv \mathsf{false} \quad \} \\
& (pg \equiv \neg ps) \wedge (ig \equiv pg \Leftarrow is) \wedge \neg(\neg is \vee ig) \\
= & \{ \quad \text{De Morgan, i.e., } \neg(\neg is \vee ig) \;\equiv\; is \wedge \neg ig \quad \} \\
& (pg \equiv \neg ps) \wedge (ig \equiv pg \Leftarrow is) \wedge is \wedge \neg ig \\
= & \{ \quad \text{reflexivity and negation} \quad \} \\
& (pg \equiv \neg ps) \wedge (ig \equiv pg \Leftarrow is) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{false}) \\
= & \{ \quad \text{Leibniz} \quad \} \\
& (pg \equiv \neg ps) \wedge (\mathsf{false} \equiv pg \Leftarrow \mathsf{true}) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{false}) \\
= & \{ \quad pg \Leftarrow \mathsf{true} \text{ is the same as } pg \quad \} \\
& (pg \equiv \neg ps) \wedge (\mathsf{false} \equiv pg) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{false}) \\
= & \{ \quad \text{Leibniz and negation} \quad \} \\
& (\mathsf{true} \equiv ps) \wedge (\mathsf{false} \equiv pg) \wedge (is \equiv \mathsf{true}) \wedge (ig \equiv \mathsf{false}) \quad .
\end{array}
$$

We conclude that the portrait is in the silver casket, that the silver inscription is true and the gold inscription is false.
□

4.8 Further reading

We recommend chapters 5 and 7 of [Bac03]. In our view, these two chapters constitute an excellent introduction to calculational logic.

SCENARIO 5

A Logical Race

5.1 Brief description and goals

This scenario shows how a calculational approach to logic leads to a concise solution of a type of logic puzzle that is based on unique existential quantifications. It can be used to introduce Boolean inequivalence ($\not\equiv$), to practise formal modelling, and to illustrate how distributivity can be used to simplify mathematical arguments. The puzzle, which we have found in [Hon98, p. 17], is about deducing a conclusion based on the statements of three people. We also show (in the exercises) how we can generalise this type of logic puzzle.

5.2 Problem

Lucy, Minnie, Nancy, and Opey ran a race. Asked how they made out, they replied:

Lucy: "Nancy won; Minnie was second."
Nancy: "Opey was last; Lucy was second."
Minnie: "Nancy was second; Opey was third."

If each of the girls made one and only one true statement, who won the race?

5.3 Prerequisites

All the required knowledge can be introduced with the solution, but elementary knowledge of calculational logic helps.

5.4 Resolution and notes

The first step in our solution is to formally model the problem. We need to express each of the three answers, so we need to introduce a way of associating a person with a position. Moreover, we need to express that, given two statements, one and only one of them is true.

To associate a person with a position, we write pn to denote that the person whose name starts by letter p ends the race in position n. For example, $N1$ means that Nancy wins the race and $M2$ means that Minnie is second[1]. These are, in fact, the two statements of Lucy, but because only one of them is true, Lucy's statement is equivalent to

$$N1 \not\equiv M2 \quad .$$

(Because $N1$ and $M2$ are Booleans, saying that they are different is the same as saying that one of them is true and the other is false.) In the same way, the statements of Nancy and Minnie are, respectively,

$$O4 \not\equiv L2 \quad \text{and}$$

$$N2 \not\equiv O3 \quad .$$

Now, to calculate the solution to the puzzle, we have to simplify the conjunction of the three statements:

$$(N1 \not\equiv M2) \wedge (N2 \not\equiv O3) \wedge (O4 \not\equiv L2) \quad .$$

Given that we do not know how to simplify any of these three inequivalences, we have to investigate properties involving both conjunction and inequivalence. A useful property is that conjunction distributes over $\not\equiv$, that is, for all p, q, and r:

$$\begin{array}{ll} & p \wedge (q \not\equiv r) \\ = & \\ & (p \wedge q) \not\equiv (p \wedge r) \quad . \end{array}$$

So, considering the notation introduced above, we have the following example:

$$\begin{array}{ll} & N1 \wedge (N2 \not\equiv M1) \\ = & \quad \{ \quad \text{conjunction distributes over } \not\equiv \quad \} \\ & (N1 \wedge N2) \not\equiv (N1 \wedge M1) \quad . \end{array}$$

[1] In this scenario, we assume that p and q range over the set $\{L,M,N,O\}$ and that m and n are positive natural numbers at most 4.

You may have noticed that this example is peculiar: conjuncts $N1 \wedge N2$ and $N1 \wedge M1$ are both false. It is impossible that Nancy finishes in first and second positions and we also exclude the possibility that Nancy and Minnie finish both in first position. In fact, we have chosen it because it reveals two important properties implicit in the problem statement. The first one reflects the impossibility of the same girl ending the race in different positions. Formally, we express this property as:

(5.4.1) $pn \wedge pm \equiv m = n$.

The second property reflects the impossibility of different girls ending the race in the same position. We express this property as follows:

(5.4.2) $pn \wedge qn \equiv p = q$.

Now, to calculate the solution to the puzzle, we just have to simplify the conjunction of the three statements:

$$(N1 \not\equiv M2) \wedge (N2 \not\equiv O3) \wedge (O4 \not\equiv L2) \quad .$$

One possible calculation, based essentially on distributivity, is as follows:

$$\begin{array}{ll} & (N1 \not\equiv M2) \wedge (N2 \not\equiv O3) \wedge (O4 \not\equiv L2) \\ = & \quad \{ \quad \text{distributivity, (5.4.1), and (5.4.2)} \quad \} \\ & (N1 \not\equiv M2) \wedge (N2 \wedge O4 \not\equiv O3 \wedge L2) \\ = & \quad \{ \quad \text{distributivity, (5.4.1), and (5.4.2)} \quad \} \\ & N1 \wedge O3 \wedge L2 \quad . \end{array}$$

The conclusion is that Nancy won the race ($N1$), Lucy was second ($L2$), Opey was third ($O3$), and Minnie was fourth (by elimination).

You may want to compare this solution with the one shown in [Hon98, p. 17]. We make a short comparison in section 5.8.

5.5 Notes for the teacher

Model the problem and express the goal The first step in our solution is to formally model the problem. We have three girls and three answers. Each answer consists of two different and mutual-exclusive statements related with the result of the race. For example, Lucy answered "Nancy won; Minnie was second.". This means that we need

to introduce a way of associating a person with a position. Moreover, we need to express that, given two statements, one and only one of them is true.

The teacher may ask for notation suggestions. We recommend to use pn to denote that the person whose name starts by letter p ends the race in position n. For example, $N1$ means that Nancy wins the race and $M2$ means that Minnie is second[2]. The teacher may use a different notation, but we recommend a concise notation that gives the same status to both people and positions. For example, writing $p(n)$ instead of pn, unnecessarily highlights people rather than positions.

Now, the two statements of Lucy are $N1$ and $M2$, but because only one of them is true, Lucy's statement is equivalent to

$$N1 \not\equiv M2 \quad .$$

(Because $N1$ and $M2$ are Booleans, saying that they are different is the same as saying that one of them is true and the other is false. An alternative formulation is $N1 \equiv \neg M2$, but we prefer to use $\not\equiv$ due to its interaction with conjunction.) We suggest the teacher to ask the students how they would formulate Lucy's statement. Once they understand Lucy's statement, the teacher should ask them to formulate the answers of the other two girls. The statements of Nancy and Minnie are, respectively,

$$O4 \not\equiv L2 \quad \text{and}$$

$$N2 \not\equiv O3 \quad .$$

Now, putting all together, we know (or we assume) that the girls spoke the truth. So, to calculate the solution to the puzzle, we have to simplify the conjunction of the three statements:

$$(N1 \not\equiv M2) \wedge (N2 \not\equiv O3) \wedge (O4 \not\equiv L2) \quad .$$

Discuss calculational strategies and implicit properties of the problem Given that we do not know how to simplify any of these three inequivalences, we have to investigate properties involving both conjunction and inequivalence. A useful property is that conjunction distributes over $\not\equiv$, that is, for all p, q, and r:

$$\begin{array}{ll} & p \wedge (q \not\equiv r) \\ = & \\ & (p \wedge q) \not\equiv (p \wedge r) \quad . \end{array}$$

[2]In this scenario, we assume that p and q range over the set $\{L,M,N,O\}$ and that m and n are positive natural numbers at most 4.

At this point, the teacher may want discuss exercise 5.6.1 with the students. Moreover, we suggest the teacher to illustrate the property with an example. We have chosen the following one, because it highlights some implicit properties of the problem. Suppose we have the expression $N1 \wedge (N2 \not\equiv M1)$. Using distributivity, we can rewrite it as follows:

$$\begin{array}{ll} & N1 \wedge (N2 \not\equiv M1) \\ = & \quad \{ \quad \text{conjunction distributes over } \not\equiv \quad \} \\ & (N1 \wedge N2) \not\equiv (N1 \wedge M1) \quad . \end{array}$$

The teacher has to be sure that the students understand how distributivity was used. Moreover, we suggest the teacher to ask the students what they can say about the expressions $N1 \wedge N2$ and $N1 \wedge M1$. The students should realise that they are both false. It is impossible that Nancy finishes in first and second positions and we also exclude the possibility that Nancy and Minnie finish both in first position. In fact, we have chosen it because it reveals two important properties implicit in the problem statement. The first one reflects the impossibility of the same girl ending the race in different positions. Formally, we express this property as:

(5.5.1) $\quad pn \wedge pm \equiv m = n \quad .$

The second property reflects the impossibility of different girls ending the race in the same position. We express this property as follows:

(5.5.2) $\quad pn \wedge qn \equiv p = q \quad .$

(We are not expecting the students to formulate this properties by themselves, but we suggest the teacher to explain how they can be used.)

Calculate the solution Now, to calculate the solution to the puzzle, we just have to simplify the conjunction of the three statements:

$$(N1 \not\equiv M2) \wedge (N2 \not\equiv O3) \wedge (O4 \not\equiv L2) \quad .$$

A detailed calculation, based on distributivity and the properties (5.5.1) and (5.5.2), is:

$$\begin{array}{ll} & (N1 \not\equiv M2) \wedge (N2 \not\equiv O3) \wedge (O4 \not\equiv L2) \\ = & \quad \{ \quad \text{distributivity} \quad \} \end{array}$$

$$
\begin{array}{ll}
 & (N1 \not\equiv M2) \ \wedge \ (N2 \wedge (O4 \not\equiv L2) \ \not\equiv \ O3 \wedge (O4 \not\equiv L2)) \\
= & \quad \{ \quad \text{distributivity, associativity} \quad \} \\
 & (N1 \not\equiv M2) \ \wedge \ (N2 \wedge O4 \ \not\equiv \ N2 \wedge L2 \ \not\equiv \ O3 \wedge O4 \ \not\equiv \ O3 \wedge L2) \\
= & \quad \{ \quad \text{from (5.5.2), } N2 \wedge L2 \text{ is false; from (5.5.1), } O3 \wedge O4 \text{ is false;} \\
 & \qquad \text{false is the unit of } \not\equiv \quad \} \\
 & (N1 \not\equiv M2) \ \wedge \ (N2 \wedge O4 \ \not\equiv \ O3 \wedge L2) \\
= & \quad \{ \quad \text{distributivity} \quad \} \\
 & N1 \wedge (N2 \wedge O4 \ \not\equiv \ O3 \wedge L2) \ \not\equiv \ M2 \wedge (N2 \wedge O4 \ \not\equiv \ O3 \wedge L2) \\
= & \quad \{ \quad \text{distributivity, associativity} \quad \} \\
 & N1 \wedge N2 \wedge O4 \ \not\equiv \ N1 \wedge O3 \wedge L2 \ \not\equiv \ M2 \wedge N2 \wedge O4 \ \not\equiv \ M2 \wedge O3 \wedge L2 \\
= & \quad \{ \quad \text{from (5.5.2), } M2 \wedge N2 \wedge O4 \text{ and } M2 \wedge O3 \wedge L2 \text{ are false;} \\
 & \qquad \text{from (5.5.1), } N1 \wedge N2 \wedge O4 \text{ is false;} \\
 & \qquad \text{false is the unit of } \not\equiv \quad \} \\
 & N1 \wedge O3 \wedge L2 \ .
\end{array}
$$

The conclusion is that Nancy won the race ($N1$), Lucy was second ($L2$), Opey was third ($O3$), and Minnie was fourth (by elimination).

The teacher may want to compare this solution with the one shown in [Hon98, p. 17]. We make a short comparison in section 5.8. The main message is that conventional solutions tend to unnecessarily convert problems on the Boolean domain to the more familiar domain of numbers.

Discuss generalisations One obvious generalisation is to increase the number of girls. Another is to increase the number of statements that each girl makes. Exercise 5.6.2 is an example of the second type of generalisation. We recommend the teacher to discuss it with the students.

5.5.1 Questions that the teacher should ask

Model the problem and express the goal

- *Do you understand the problem?*

  The teacher has to be sure that the students understand what is required.

- *What is our goal? What do we want to prove?*

  Every time we are working in a goal-oriented fashion, this question should be asked explicitly. The goal is to deduce from the answers of the girls who won the race. This means that to model the problem, we have to model the answers.

- *How can we express the answers more formally?*

  The students should understand that we need to introduce a way of associating a person with a position. The teacher may ask for notation suggestions. (Please see the recommendation above.)

  Moreover, we need to express that, given two statements, one and only one of them is true. The teacher should ask how we can express that from two propositions exactly one is true; help may be given when modelling one of the answers, but the other two should be modelled by the students.

Discuss calculational strategies and implicit properties of the problem

- *Can you simplify any of the conjuncts (i.e. any of the inequivalences)?*

  The students should understand that the inequivalences cannot be simplified. The teacher should guide the students to the conclusion that, in order to simplify the expression, we have to investigate properties involving conjunction and inequivalence.

- *Do you know any properties involving conjunction and inequivalence?*

  We do not expect the students to know any property involving conjunction and inequivalence. However, we suggest the teacher to allow the students to think about properties. We also suggest the teacher to prove the distributivity property together with the students (see exercise 5.6.1).

- *What can we say about* $N1 \wedge N2$ *and* $N1 \wedge M1$*?*

  We expect that most students understand that these expressions are false. If that is the case, we suggest the teacher to ask them why and to introduce the properties (5.5.1) and (5.5.2). It is important that the students understand how they can use these properties in a calculation: because false is the unit of $\not\equiv$, whenever we have such an expression, we can remove it.

Calculate the solution

- *[After the second step in the calculation.] How can we simplify this new expression?*

This is where we use the properties (5.5.1) and (5.5.2). We suggest the teacher to let the students perform the simplification.

- *[At the end of the calculation] What is the solution to the problem?*

  When we reach the final conjunction, the students should realise that we cannot simplify it anymore. The teacher should ask what is the solution to the problem.

Discuss generalisations

- *Can we generalise this problem? How?*

  Asking this question explicitly helps to cultivate inquisitive minds. The students should realise that a problem is never really solved, as we can always raise new questions. Some generalisations can be set as homework.

  The teacher can take the opportunity to discuss the generalisation shown in the exercise 5.6.2 and to elaborate on some of the advantages of a calculational approach to this type of puzzle. Whilst some people can easily solve the problem shown above using intuition, the generalisation shown in the exercise 5.6.2 is more difficult. However, using a calculational approach, both problems have the same complexity.

5.5.2 Questions that the teacher should not ask

Model the problem and express the goal

- *How can we associate a person with a position?*

  We recommend the teacher to let the students reach the conclusion that the notation has to express the association between people and positions. It is important for them to learn how to model problems more formally.

Discuss calculational strategies and implicit properties of the problem

- *Can you see that $N1 \wedge N2$ and $N1 \wedge M1$ are both false?*

  From experience, we believe that most students will understand that these expressions are false. We suggest the teacher to ask what the value of these expressions is, rather than revealing it. (See related question in the previous section.)

5.5.3 Concepts that the teacher should introduce

Calculational logic

Distributivity

Inequivalence

5.6 Extensions and exercises

Exercise 5.6.1 (Distributivity) Prove that conjunction distributes over inequivalence, i.e., prove the following equality for all p, q, and r:

$$\begin{array}{ll} & p \wedge (q \not\equiv r) \\ = & \\ & (p \wedge q) \not\equiv (p \wedge r) \ . \end{array}$$

□

Exercise 5.6.2 (Generalisation) Lucy, Minnie, Nancy, and Opey ran a race. Asked how they made out, they replied:

Lucy: "Nancy won; Minnie was second; Opey was fourth."
Minnie: "Minnie won; Nancy was second; Opey was third."
Nancy: "Opey won; Opey was second; Nancy was fourth."

If each of the girls made one and only one true statement, who won the race?

□

5.7 Solutions to extensions and exercises

5.6.1 Perhaps the simplest way of proving this property is by case analysis on p. If p is true, both sides are equal because true is the unit of conjunction. If p is false, both sides are false because false is the zero of conjunction and false $\not\equiv$ false is false.

An alternative and calculational proof, based on equivalence, is:

$$
\begin{array}{ll}
& p \wedge (q \not\equiv r) \\
= & \quad \{ \quad \text{golden rule, definition of } \not\equiv \quad \} \\
& p \equiv (q \equiv r \equiv \mathsf{false}) \equiv p \vee (q \equiv r \equiv \mathsf{false}) \\
= & \quad \{ \quad \text{disjunction distributes over } \equiv \text{, symmetry, associativity} \quad \} \\
& (p \equiv q \equiv p \vee q) \equiv (p \equiv r \equiv p \vee r) \equiv \mathsf{false} \\
= & \quad \{ \quad \text{definition of } \not\equiv \quad \} \\
& (p \wedge q) \not\equiv (p \wedge r) \quad .
\end{array}
$$

□

5.6.2 Using the same notation as in the previous solution and the brackets $\langle\!\langle\ \rangle\!\rangle$ to express uniqueness, the three statements are formally expressed as:

$$\langle\!\langle N1,M2,O4 \rangle\!\rangle \wedge \langle\!\langle M1,N2,O3 \rangle\!\rangle \wedge \langle\!\langle O1,O2,N4 \rangle\!\rangle \quad .$$

The expression $\langle\!\langle N1,M2,O4 \rangle\!\rangle$ means that exactly one of $N1$, $M2$, and $O4$ is true. Distributivity, in this more general case, can be expressed as:

$$p \wedge \langle\!\langle q,r,s \rangle\!\rangle \ = \ \langle\!\langle p \wedge q \ , p \wedge r \ , p \wedge s \rangle\!\rangle \quad .$$

To calculate the solution, we use distributivity together with the properties (5.5.1) and (5.5.2). We also use the following two rules, where L is a list of expressions and p is a single proposition:

$$\langle\!\langle \mathsf{false},L \rangle\!\rangle = \langle\!\langle L \rangle\!\rangle \quad \text{, and}$$

$$\langle\!\langle p \rangle\!\rangle = p \quad .$$

We calculate the solution as follows:

$$
\begin{array}{ll}
& \langle\!\langle N1,M2,O4 \rangle\!\rangle \wedge \langle\!\langle M1,N2,O3 \rangle\!\rangle \wedge \langle\!\langle O1,O2,N4 \rangle\!\rangle \\
= & \quad \{ \quad \text{distributivity, (5.5.1), and (5.5.2)} \quad \} \\
& \langle\!\langle \langle\!\langle N1 \wedge O3 \rangle\!\rangle, \langle\!\langle M2 \wedge O3 \rangle\!\rangle, \langle\!\langle O4 \wedge M1 \ , O4 \wedge N2 \rangle\!\rangle \rangle\!\rangle \wedge \langle\!\langle O1,O2,N4 \rangle\!\rangle \\
= & \quad \{ \quad \text{distributivity, } \langle\!\langle p \rangle\!\rangle = p \text{, (5.5.1), (5.5.2),} \\
& \qquad \text{and } \langle\!\langle \mathsf{false},L \rangle\!\rangle = \langle\!\langle L \rangle\!\rangle \quad \} \\
& \langle\!\langle \langle\!\langle \mathsf{false} \rangle\!\rangle, \langle\!\langle M2 \wedge O3 \wedge N4 \rangle\!\rangle, \langle\!\langle \langle\!\langle \mathsf{false} \rangle\!\rangle, \langle\!\langle \mathsf{false} \rangle\!\rangle \rangle\!\rangle \rangle\!\rangle \\
= & \quad \{ \quad \langle\!\langle \mathsf{false},L \rangle\!\rangle = \langle\!\langle L \rangle\!\rangle \text{ and } \langle\!\langle p \rangle\!\rangle = p \quad \} \\
& M2 \wedge O3 \wedge N4 \quad .
\end{array}
$$

We conclude that Lucy won the race (by elimination), Minnie was second ($M2$), Opey was third ($O3$), and Nancy was fourth ($N4$).
□

5.8 Further reading

The problem presented in this scenario was taken from [Hon98, p. 17]. In there, Honsberger solves the puzzle by translating it to the domain of numbers and formulating the relevant properties in terms of numbers. His solution is an extreme case of what is conventionally done in school mathematics: problems are always formulated using the more familiar domain of numbers and logic is used implicitly in the arguments. (We consider his solution extreme, since the problem was originally a logic problem. There was no need at all to translate it to a different, more complex, domain.) We recommend the teacher to compare both solutions.

Finally, to the best of our knowledge, the type of generalisation shown in exercise 5.6.2 is new. In our view, it illustrates well how a calculational approach to logic can be effective, since the complexity of its solution is essentially the same as the one of the solution to the original problem.

SCENARIO 6

A Calculational Proof of the Handshaking Lemma

6.1 Brief description and goals

This teaching scenario shows a goal-oriented and calculational proof of the Handshaking lemma, an elementary result in graph theory. The lemma states that every finite undirected graph has an even number of vertices with odd degree. The solution presented in this scenario can be used to introduce the Eindhoven quantifier notation.

6.2 Problem

Let a finite number of points be joined in pairs by any system of curves, including the possibility of loops (for example, joining a point C with itself; see figure 6.1) and of multiple edges (joining the same pair of points). We define the *local degree* of a vertex A, denoted by $d.A$, to be the number of edges incident with the point A, counting loops twice. For example, in figure 6.1,

$$d.A = 6,\ d.B = 3, \text{ and } d.C = 3 \quad .$$

We want to show that in any network, as outlined above, the number of vertices which have odd local degree is an even number. (Note that in the system shown in figure 6.1, precisely *two* vertices, B and C, have odd local degrees.)

The Handshaking Lemma This property is also known as the Handshaking Lemma. If we think of the vertices as people, and the joining of two vertices A and B (say)

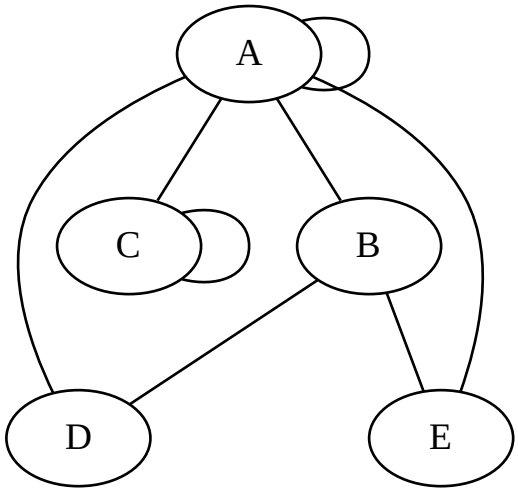


Figure 6.1: System of curves with five points

to mean that A and B shook hands (loops, if any, indicating one shook hands with himself and counting as two handshakes), the local degree $d.A$ of a vertex A gives the total number of times A shook hands. What we want to show, then, is that the number of people who have shaken hands an odd number of times is even. This application is all the more interesting because it is independent of time — one can state without fear of contradiction that the number of people at the opera next Thursday (or in the whole world from the beginning of time if you like) who will shake hands an odd number of times is even. (One might enjoy verifying this result with a group of friends.)

A remark on terminology A system of curves that join a finite number of points is also called an *undirected graph*. In the problem statement, we use the same terminology as in [Hon98, p. 7], because we find it more accessible. (In fact, most of the text is transcribed from [Hon98].)

6.3 Prerequisites

All the required knowledge can be introduced with the solution, but an elementary knowledge of quantifiers can be useful.

6.4 Resolution and notes

The problem asks us to show that in any network, as outlined in the problem statement, the number of vertices which have odd local degree is an even number. As typical in

mathematics textbooks, this problem asks for a verification of a given fact. But how can one derive such a fact in the first place? Suppose, for a moment, that we ask the question "what is the parity of the number of vertices which have odd local degree?". How could we proceed in that case?

Well, the first step is to express the goal. We need to express the number of vertices which have odd local degree. Assuming that V is the set of all vertices, we want to count the number of vertices $a{\in}V$ such that $d.a$ is odd. One way of formally expressing this is by using the summation quantifier:

$$\langle \Sigma a \,:\, a{\in}V \wedge odd.(d.a) \,:\, 1 \rangle \quad .$$

There are five components to the notation we are using. The first component is the *quantifier* Σ, which denotes summation of an arbitrary number of values. The second component is the *dummy* variable a. The third component is the *range* of the dummy; in this case, the range is $a{\in}V \wedge odd.(d.a)$. The range is a Boolean-valued expression that determines the set of values of the dummy for which the expression is true. The fourth component is the *term*. In this case, the term is the natural number 1, meaning that we add 1 for each value a that satisfies the range (in other words, we are adding 1 (counting) for each node a with an odd degree in V.). The final component of the notation is the angle brackets; these serve to delimit the scope of the dummy variable.

Our goal is to determine the value of the following expression:

$$even.\langle \Sigma a \,:\, a{\in}V \wedge odd.(d.a) \,:\, 1 \rangle \quad .$$

If the result is true, there is an even number of vertices with odd degree; otherwise, there is an odd number. The problem statement claims that the result is always true. However, the goal we have proposed is to calculate its value. We know that predicate *even* distributes through addition, so we calculate:

$$\begin{array}{ll}
 & even.\langle \Sigma a \,:\, a{\in}V \wedge odd.(d.a) \,:\, 1 \rangle \\
= & \quad \{ \quad even \text{ distributes over addition} \quad \} \\
 & \langle \equiv a{:}\ a{\in}V \wedge odd.(d.a){:}\ even.1 \rangle \\
= & \quad \{ \quad even.1 \text{ is false} \quad \} \\
 & \langle \equiv a{:}\ a{\in}V \wedge odd.(d.a){:}\ \mathsf{false} \rangle \\
= & \quad \{ \quad \text{the range can be simplified by using the trading rule} \quad \} \\
 & \langle \equiv a{:}\ a{\in}V{:}\ odd.(d.a) \Rightarrow \mathsf{false} \rangle \\
= & \quad \{ \quad odd.(d.a) \Rightarrow \mathsf{false} \ \equiv \ even.(d.a) \quad \}
\end{array}$$

$$\begin{array}{ll} & \langle \equiv a: a{\in}V: even.(d.a) \rangle \\ = & \quad \{ \quad even \text{ distributes over addition} \quad \} \\ & even.\langle \Sigma a : a{\in}V : d.a \rangle \quad . \end{array}$$

This calculation shows that the parity of the number of vertices with odd degree is the same as the parity of the sum of all the degrees. But because each edge has two ends, the sum of all the degrees is simply twice the total number of edges. We thus have:

$$\begin{array}{ll} & even.\langle \Sigma a \ : \ a{\in}V \wedge odd.(d.a) \ : \ 1 \rangle \\ = & \quad \{ \quad \text{calculation above} \quad \} \\ & even.\langle \Sigma a : a{\in}V : d.a \rangle \\ = & \quad \{ \quad \text{the sum of all the degrees is twice the number of edges, i.e.,} \\ & \qquad \text{it is an even number} \quad \} \\ & \texttt{true} \ . \end{array}$$

And so we can conclude that every undirected graph contains an even number of vertices with odd degree.

6.5 For the teacher

Formalising the problem in a goal-oriented way We suggest the teacher to start by observing that this problem asks for a verification of a given fact and by reformulating the problem to the following: "what is the parity of the number of vertices which have odd local degree?". This reformulation is goal-oriented and reflects reality better: when solving new problems, we usually do not know the answer. Although guessing and verifying is a valid technique (and useful when the guesser is very good), it teaches little on mathematical invention.

The first step is to express the goal. We need to express the number of vertices which have odd local degree. Assuming that V is the set of all vertices, we want to count the number of vertices $a{\in}V$ such that $d.a$ is odd. One way of formally expressing this is by using the summation quantifier:

$$\langle \Sigma a \ : \ a{\in}V \wedge odd.(d.a) \ : \ 1 \rangle \quad .$$

We use the Eindhoven quantifier notation and, because it is convenient for calculational purposes, we recommend the teacher to use it too. There are five components to

the notation. The first component is the *quantifier* Σ, which denotes summation of an arbitrary number of values. The second component is the *dummy* variable a. The third component is the *range* of the dummy; in this case, the range is $a{\in}V \wedge odd.(d.a)$. The range is a Boolean-valued expression that determines the set of values of the dummy for which the expression is true. The fourth component is the *term*. In this case, the term is the natural number 1, meaning that we add 1 for each value a that satisfies the range (in other words, we are adding 1 (counting) for each node a with an odd degree in V.). The final component of the notation is the angle brackets; these serve to delimit the scope of the dummy variable. For a comprehensive presentation of the quantifier calculus that we use in this scenario, we recommend [Bac03, Chapter 11].

Our goal is to determine the value of the following expression:

$$even.\langle \Sigma a \;:\; a{\in}V \wedge odd.(d.a) \;:\; 1\rangle \quad .$$

If the result is true, there is an even number of vertices with odd degree; otherwise, there is an odd number. The problem statement claims that the result is always true. However, the goal we have proposed is to calculate its value. The teacher should make clear that the final result is a Boolean value, that is, it is either true or false.

Manipulating quantifiers Now, because we have the predicate *even* applied to a summation, we can use the following distributivity property:

$$even.(m{+}n) \equiv even.m \equiv even.n \quad .$$

In other words, *even* distributes over addition. In terms of arbitrary summations, this rule can be expressed as:

$$even.\langle \Sigma a{:}R{:}T\rangle = \langle \equiv a{:}\; R{:}\; even.T\rangle \quad .$$

(Some examples may be helpful if the students are not familiar with this property.) This means that we can manipulate our goal similarly:

$$\begin{array}{ll} & even.\langle \Sigma a \;:\; a{\in}V \wedge odd.(d.a) \;:\; 1\rangle \\ = & \quad \{ \quad even \text{ distributes over addition} \quad \} \\ & \langle \equiv a{:}\; a{\in}V \wedge odd.(d.a){:}\; even.1\rangle \quad . \end{array}$$

We now have a quantified expression that represents a continued equivalence. The term *even*.1 is the same as false, so there is not much we can do to it. We can, however, simplify the range by using the so-called trading rule:

$$
\begin{array}{ll}
 & \langle \equiv a : R \wedge P : T \rangle \\
= & \quad \{ \quad \text{trading rule} \quad \} \\
 & \langle \equiv a : R : P \Rightarrow T \rangle \ .
\end{array}
$$

In terms of symbols, an implication, $P\Rightarrow$, in the term is "traded" into a conjunct, $P\wedge$, in the range. This means that we can continue the calculation as:

$$
\begin{array}{ll}
 & \langle \equiv a : \ a \in V \wedge odd.(d.a) : \ even.1 \rangle \\
= & \quad \{ \quad even.1 \text{ is the same as } \mathsf{false} \quad \} \\
 & \langle \equiv a : \ a \in V \wedge odd.(d.a) : \ \mathsf{false} \rangle \\
= & \quad \{ \quad \text{trading rule} \quad \} \\
 & \langle \equiv a : \ a \in V : \ odd.(d.a) \Rightarrow \mathsf{false} \rangle \ .
\end{array}
$$

Now, the expression $odd.(d.a) \Rightarrow \mathsf{false}$ is the same as $\neg(odd.(d.a))$, i.e., $even.(d.a)$. A simple calculational proof of this fact is:

$$
\begin{array}{ll}
 & odd.(d.a) \Rightarrow \mathsf{false} \\
= & \quad \{ \quad \text{definition of } \Rightarrow \quad \} \\
 & odd.(d.a) \ \equiv \ odd.(d.a) \wedge \mathsf{false} \\
= & \quad \{ \quad \mathsf{false} \text{ is the zero of conjunction} \quad \} \\
 & odd.(d.a) \equiv \mathsf{false} \\
= & \quad \{ \quad \text{negation} \quad \} \\
 & even.(d.a) \ .
\end{array}
$$

We can now continue the above calculation as follows:

$$
\begin{array}{ll}
 & \langle \equiv a : \ a \in V : \ odd.(d.a) \Rightarrow \mathsf{false} \rangle \\
= & \quad \{ \quad odd.(d.a) \Rightarrow \mathsf{false} \ \equiv \ even.(d.a) \quad \} \\
 & \langle \equiv a : a \in V : even.(d.a) \rangle \ .
\end{array}
$$

If we now apply the distributivity property in reverse, we conclude the following equality:

$$
\begin{array}{ll}
 & even.\langle \Sigma a \ : \ a \in V \wedge odd.(d.a) \ : \ 1 \rangle \\
= & \quad \{ \quad \text{steps shown above} \quad \}
\end{array}
$$

$$
\begin{array}{ll}
 & \langle \equiv a : a \in V : even.(d.a) \rangle \\
= & \quad \{ \quad even \text{ distributes over addition} \quad \} \\
 & even.\langle \Sigma a : a \in V : d.a \rangle \quad .
\end{array}
$$

The final step This calculation shows that the parity of the number of vertices with odd degree is the same as the parity of the sum of all the degrees. But because each edge has two ends, the sum of all the degrees is simply twice the total number of edges. We thus have:

$$
\begin{array}{ll}
 & even.\langle \Sigma a \ : \ a \in V \wedge odd.(d.a) \ : \ 1 \rangle \\
= & \quad \{ \quad \text{calculation above} \quad \} \\
 & even.\langle \Sigma a : a \in V : d.a \rangle \\
= & \quad \{ \quad \text{the sum of all the degrees is twice the number of edges, i.e.,} \\
 & \qquad \text{it is an even number} \quad \} \\
 & \texttt{true} \ .
\end{array}
$$

And so we can conclude that every undirected graph contains an even number of vertices with odd degree.

6.5.1 Questions that the teacher should ask

Formalising the problem in a goal-oriented way

- *Do you understand the problem?*

  The problem statement is quite long, so it is important that the teacher spends sufficient time making sure that all the students understand the problem statement. Showing different examples may help.

- *What is the parity of the number of vertices which have odd local degree?*

  The problem statement answers this question, but we suggest the teacher to ask it. As mentioned above, the first step is to transform the problem in a goal-oriented way, as if we did not know the answer. We suggest the teacher to stress that in research problems we never know the answer before tackling the problems. That is why it is important to ask goal-oriented questions.

Manipulating quantifiers

- *Do you understand how the property* $even.(m+n) \equiv even.m \equiv even.n$ *generalises to quantified summations?*

  It is important that the students understand why we can transform a summation into a continued equivalence. In case the students do not understand, we suggest the teacher illustrates the properties with some examples. For example, it is not difficult to see that

  $$even.\langle \Sigma a : 0 \leq k \leq 3 : k \rangle = \langle \equiv a : 0 \leq k \leq 3 : even.k \rangle$$

  (just expand both quantifications; we also know that $even.0$ is true).

- *Do you see any way of simplifying the range? Is it possible to remove the occurrence of* $odd.(d.a)$ *from the range?*

  These questions prepare the introduction of the trading rule that is used to simplify the range. When presenting the trading rule, the teacher can illustrate it with simple examples.

The final step

- *We have reached the equality*

  $$\begin{array}{ll} & even.\langle \Sigma a \ : \ a \in V \wedge odd.(d.a) \ : \ 1 \rangle \\ = & \quad \{ \quad \textit{calculation above} \quad \} \\ & even.\langle \Sigma a : a \in V : d.a \rangle \end{array}$$

  What does it mean?

  The students should understand that we have proved that the parity of the number of vertices with odd degree is the same as the parity of the sum of all the degrees.

- *What can we say about the parity of the sum of all the degrees? What do we know about the sum of all the degrees?*

  The teacher should lead the students to the conclusion that the sum of all degrees is twice the number of edges. Therefore, it is an even number and the conclusion is that $even.\langle \Sigma a \ : \ a \in V \wedge odd.(d.a) \ : \ 1 \rangle$ is true.

6.5.2 Questions that the teacher should not ask

The final step

- *Can you see that the sum of all degrees is twice the number of edges?*

  This question gives away a crucial property that is used to solve the problem. The teacher should allow the students to discover this fact by themselves.

6.5.3 Concepts that the teacher should introduce

Calculational proof

Distributivity

Eindhoven quantifier notation

Goal-directed investigations

6.6 Extensions and exercises

Exercise 6.6.1 (Warm-up) Prove that the parity of a sum of a set of integers is odd equivales the number of odd elements is odd. More formally, given a set of integers S, prove the following equality:

$$\begin{array}{ll} & even.\langle \Sigma a : a{\in}S : a \rangle \\ = & \\ & even.\langle \Sigma a \ : \ a{\in}S \wedge odd.a \ : \ 1 \rangle \ . \end{array}$$

□

6.7 Solutions to extensions and exercises

6.6.1 The proof is quite similar to the one we have used to establish the Handshaking Lemma:

$$\begin{array}{ll} & even.\langle \Sigma a : a{\in}S : a \rangle \\ = & \quad \{ \quad even \text{ distributes over addition} \quad \} \\ & \langle \equiv a{:}\ a{\in}S{:}\ even.a \rangle \\ = & \quad \{ \quad \text{we want to introduce the expression } odd.a \text{ in the range;} \\ & \qquad \text{we use the trading rule, because } even.a \ \equiv \ odd.a \Rightarrow false \quad \} \end{array}$$

$$
\begin{array}{ll}
 & \langle \equiv a:\ a{\in}S \wedge odd.a:\ \mathsf{false} \rangle \\
= & \quad \{\quad even.1 \equiv \mathsf{false} \quad \} \\
 & \langle \equiv a:\ a{\in}S \wedge odd.a:\ even.1 \rangle \\
= & \quad \{\quad even \text{ distributes over addition} \quad \} \\
 & even.\langle \Sigma a\ :\ a{\in}S \wedge odd.a\ :\ 1 \rangle \ .
\end{array}
$$

□

6.8 Further reading

Conventional solutions for this problem are usually very similar to the following one, taken from [Hon98, p. 8]:

> The proof in general is simple. We denote by T the total of all the local degrees:
>
> (1) $T = d(A) + d(B) + d(C) + \cdots + d(K)$.
>
> In evaluating T we count the number of edges running into A, the number into A, etc., and add. Because each edge has two ends, T is simply twice the number of edges; hence T is even.
>
> Now the values $d(P)$ on the right-hand side of (1) which are even add up to a sub-total which is also even. The remaining values $d(P)$ each of which is odd, must also add up to an even sub-total (since T is even). This shows that there is an even number of odd $d(P)$'s (it takes an even number of odd numbers to give an even sum). Thus there must be an even number of vertices with odd local degree.

There is nothing wrong with this solution in the sense that it shows why the property holds. However, it is clearly oriented to verification: it starts by introducing the total sum of all the local degrees, observing that its value is even; then it analyses that sum to conclude the property. The question is: how can we teach students to consider the total sum of all the local degrees? In general, how can we teach students to identify seemingly unrelated concepts that will be crucial in the development of their arguments? We don't think we can.

On the other hand, if we look at the goal-oriented proof, we see that the goal is simple to express. Furthermore, with some training, most students would write it correctly and

would be able to calculate that the parity of the number of vertices with odd degree is the same as the parity of the sum of all the degrees. And then (and only then) the introduction of the total sum of all the degrees would make sense. In conclusion, we believe it is more valuable to work in a formal and goal-oriented way, since it allows us to discover the crucial properties.

Finally, for more information on the quantifiers notation and manipulation rules, we recommend [Bac03, Chapter 11].

SCENARIO 7

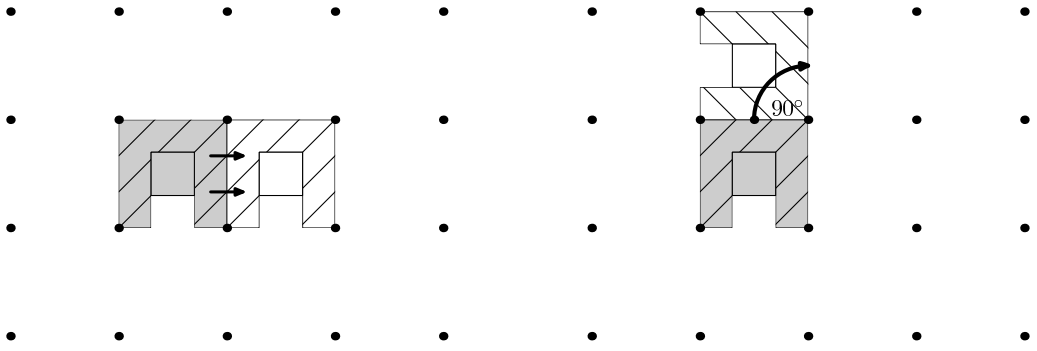
Moving a Heavy Armchair

7.1 Brief description and goals

This scenario introduces the notion of *invariant* through a simple and recreational example. The problem was taken from [Bac03, Chapter 12] and does not require any prerequisites from the students.

7.2 Problem

Suppose it is required to move a square armchair sideways by a distance equal to its own width (see figure 7.1(a)). However, the chair is so heavy that it can only be moved by rotating it through 90°, around one of its corners (see figure 7.1(b)). Is it possible to move the chair as desired? If so, how? If not, why not? You can assume that the room is of infinite size (the figures illustrate only a small part).



(a) Goal: move the armchair sideways (b) The chair has to be rotated 90°

Figure 7.1: Moving a heavy armchair

7.3 Prerequisites

No prerequisites.

7.4 Resolution and notes

An affirmative answer to the question means that there is an algorithm that moves the chair as required. In particular, it is a finite sequence of rotations around one of the corners of the chair. (Whenever we write rotations, we obviously mean 90° rotations.)

Since the only instructions involved in the algorithm are rotations, we should investigate what happens to the chair after a rotation is done. Taking a second look at figure 7.1(b) and imagining that the black dots form a square grid, it is easy to see that:

- the chair moves to one of the four vertically or horizontally adjacent squares;
- the orientation of the chair changes; more specifically, if the chair was facing north-south before a rotation, it will be facing east-west after the rotation (and vice-versa).

The first point suggests that we should find a way of distinguishing a square from its four vertically or horizontally adjacent squares. One way of doing that is by painting the floor alternately with black and white squares, like a chessboard, with each of the squares being the same size as the armchair (see figure 7.2).

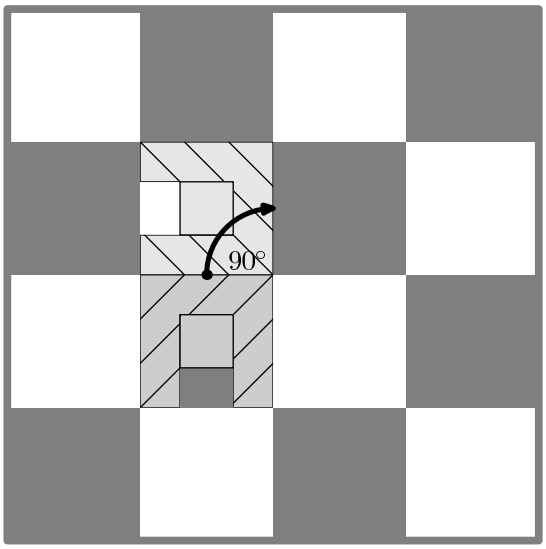


Figure 7.2: The floor is painted alternately with black and white squares.

Based on figure 7.2, instead of saying that the chair moves to one of the four vertically or horizontally adjacent squares, we can simply say that the chair moves to a square of a different colour.

Let us assume that the chair is initially on a black square and that its orientation is north-south. Then, the goal is to move the chair to the white square at its left or at its right, using only rotations. Also, based on the discussion above, an invariant of rotating the armchair around a corner is

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

Clearly, the invariant is false when the chair in on a white square and facing north-south. Therefore, it is impossible to move the chair as desired.

7.5 Notes for the teacher

Analysis of the problem This problem is algorithmic, because an affirmative answer to the question means that there is an algorithm that moves the chair as required. In particular, it is a finite sequence of rotations around one of the corners of the chair. It is important that the students understand this.

Since the only instructions involved in the algorithm are rotations, we should investigate what happens to the chair after a rotation is done. We recommend the teacher to ask the students for suggestions. Looking at figure 7.1(b) and imagining that the black dots form a square grid, the students should observe the following:

- the chair moves to one of the four vertically or horizontally adjacent squares;
- the orientation of the chair changes; more specifically, if the chair was facing north-south before a rotation, it will be facing east-west after the rotation (and vice-versa).

If the students do not identify these properties, the teacher can help with questions like "to which squares does the chair move to after a rotation?" or "what happens to the chair when a rotation is done?".

Painting the floor The first point suggests that we should find a way of distinguishing a square from its four vertically or horizontally adjacent squares. The teacher should ask the students if they know any simple way of doing that. One way consists in painting the floor alternately with black and white squares, like a chessboard, with

each of the squares being the same size as the armchair (see figure 7.2). Using colours to distinguish the elements of a problem is a common strategy.

Based on figure 7.2 , the teacher should ask the students how to rephrase the first point shown above. They should realise that instead of saying that the chair moves to one of the four vertically or horizontally adjacent squares, we can simply say that the chair moves to a square of a different colour.

Finding the invariant We suggest the teacher to assume that the chair is initially on a black square and that its orientation is north-south. Then, the teacher can ask what is the goal of the problem. Clearly, the goal is to move the chair to the white square at its left or at its right, using only rotations.

We suggest the teacher to ask if they can find any invariant of the problem. Asking for a relation between the colour of the square and the orientation may help the students. The students should realise that an invariant of rotating the armchair around a corner is

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

Or, symmetrically,

$$\text{the chair in on a white square} \equiv \text{the chair is facing east-west} \quad .$$

Clearly, the invariant is false when the chair in on a white square and facing north-south. Therefore, it is impossible to move the chair as desired.

7.5.1 Questions that the teacher should ask

Analysis of the problem

- *Do you understand the problem?*

  The teacher has to be sure that the students understand what is required. From our experience, the most common misunderstandings are on the type of rotation, the size of the floor (it is unlimited), and the allowed number of rotations (we are allowed to make an unlimited number of rotations).

- *Is this an algorithmic problem?*

  One of the basic and most important skills in algorithmic problem solving is to be able to identify problems of algorithmic nature. We suggest the teacher to ask this question explicitly, so that in subsequent problems students ask the same question to themselves.

- *What happens to the chair after a rotation is done?*

  The goal is to help the students identify the two crucial properties of the problem. If the students do not identify these properties, we suggest the teacher to refine the question, as in, for example, "to which squares does the chair move after a rotation?" or "how does the chair change when a rotation is done?".

Painting the floor

- *How can we distinguish a square from its four vertically or horizontally adjacent squares?*

  The goal is to introduce the painting of the floor. If the students do not suggest the binary distinction (black/white, A/B, etc.), we recommend the teacher to ask something like "Suppose that we paint this square black. How can we distinguish the four neighbours?". If the students suggest the painting of the neighbours with four different colours, we suggest the teacher to ask if we need all these colours.

- *Now that the floor is painted as a chessboard, how can we rephrase the first point shown above?*

  The goal is to help the students realise that after a rotation, the chair moves to a square of a different colour.

Finding the invariant

- *Assume that initially the chair is in a black square facing north-south. What is the goal of the problem?*

  The students should understand that, with this assumption, the goal is to move the chair to the white square at its left or at its right.

- *Can you think of any property that remains constant after a rotation? That is, can you think of an invariant property?*

  If the students do not suggest any invariant, we suggest the teacher to repeat the assumptions and properties. For example: "The chair is initially on a black square facing north-south. We have seen that whenever we rotate the chair, both the colour of its square and its orientation change. Can you think of any property that remains constant?". A more specific question that can be used is: "Is there any relation between the colour of the square and the orientation?".

7.5.2 Questions that the teacher should not ask

Analysis of the problem

- *How does the orientation of the chair change after a rotation is done?*

  This question suggests that the orientation is an important property of the problem. We suggest the teacher to start with more general questions (e.g. "what happens to the chair?") and ask this question only if the students fail to identify a change in the orientation.

Painting the floor

- *Can we paint the floor to distinguish a square from its four vertically or horizontally adjacent squares?* or *Can we paint the floor like a chessboard to distinguish a square from its four vertically or horizontally adjacent squares?*

  These questions give the colouring strategy away. Before asking this type of question, the teacher should ask how we can distinguish a square from its neighbours.

7.5.3 Concepts that the teacher should introduce

Invariant

7.6 Extensions and exercises

Exercise 7.6.1 (Mutilated chessboard) A chessboard has had its top-right and bottom-left squares removed so that there are 62 squares remaining. (See figure 7.3.) An unlimited supply of dominoes has been provided; each domino will cover exactly two

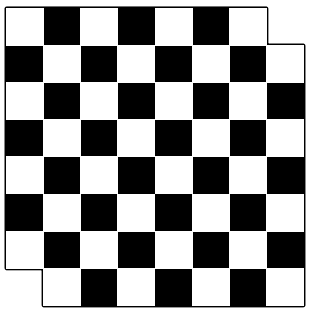


Figure 7.3: Mutilated Chess Board

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?

□

Exercise 7.6.2 (Knockout tournament) 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.)

□

Exercise 7.6.3 (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?

□

7.7 Solutions to extensions and exercises

7.6.1 When we place a domino on the chessboard, we cover one black square and one white square. As a result, the number of white covered squares and black covered squares is equal (this is an invariant). But we have removed two black squares, so the mutilated chessboard has more white squares than black squares. Therefore, it is impossible to cover the chessboard with dominoes.
□

7.6.2 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 (ie, $k = g$ is invariant). 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.
□

7.6.3 We are given the initial and final numbers of empty boxes and we are required to find the total number of boxes at the end of the process. This motivates the introduction of variables for these values; we use t to denote the total number of boxes, and e to denote the number of empty boxes. Initially, we know that $t = e = 11$. We want to determine the value of t after the small boxes are placed into the medium boxes.

Since the number of empty boxes that is selected is unknown, let us focus on the atomic action of placing boxes inside an empty box. Whenever we put eight boxes inside an empty box, the total number of boxes increases by eight and the number of empty boxes increases by seven. Therefore, the assignment that models this is:

$$t\,,e \;:=\; t{+}8\,,e{+}7 \quad .$$

An invariant of this type of assignment is easy to calculate. We know that there must be some linear combination of t and e that remains constant after execution of the assignment, so we propose to calculate x and y, such that

$$(x{\cdot}t + y{\cdot}e)[t\,,e \;:=\; t{+}8\,,e{+}7] \;=\; x{\cdot}t + y{\cdot}e \quad .$$

In words, we want to calculate x and y, such that the value of $x{\cdot}t + y{\cdot}e$ after executing the assignment $t\,,e := t{+}8\,,e{+}7$ remains the same. The calculation is straightforward:

$$\begin{array}{ll} & (x{\cdot}t + y{\cdot}e)[t\,,e \;:=\; t{+}8\,,e{+}7] \;=\; x{\cdot}t + y{\cdot}e \\ = & \quad \{ \quad \text{substitution} \quad \} \\ & x{\cdot}(t{+}8) + y{\cdot}(e{+}7) \;=\; x{\cdot}t + y{\cdot}e \\ = & \quad \{ \quad \text{arithmetic} \quad \} \\ & 8{\cdot}x + 7{\cdot}y \;=\; 0 \\ \Leftarrow & \quad \{ \quad \text{arithmetic} \quad \} \\ & x = 7 \;\wedge\; y = -8 \quad . \end{array}$$

Thus, an invariant of the assignment is $7{\cdot}t - 8{\cdot}e$. We know that its initial value is -11 (because $t = e = 11$). Since it is an invariant, its final value has to be -11. This means

that on termination, when $e = 102$, we have

$$7 \cdot t - 8 \cdot 102 \; = \; -11 \quad .$$

Therefore, the final value of t is 115. There is, at the end of the process there are 115 boxes.

This solution is also an example of appropriate naming: we have introduced only two variables, one to express the goal, and the other to model the concrete data given by the problem statement. If we had introduced variables for the numbers of small, medium, and large boxes, the solution would be more complicated.
□

7.8 Further reading

The problem of the heavy chair was taken from [Bac03, Chapter 12] and the problems shown in the exercises where taken from [Bac07]. We would like to thank to Roland Backhouse, who kindly authorised the use of the figures.

SCENARIO 8

Exchanging the Values of Two Variables

8.1 Brief description and goals

This scenario discusses and generalises a programming trick that can be used to exchange the values of two variables without using additional variables. It serves as an introduction to formal manipulation of algorithms and it can be also be used to introduce the Guarded Command Language. In our view, it is also a good example of investigative mathematics.

8.2 Problem

One way of exchanging the values of two variables x and y consists in using a temporary variable z to store one of the values. Using the Guarded Command Language, we can write the exchange of values as:

$$\{\ x = X\ \wedge\ y = Y\ \}$$
$$z := x\ ;\ x := y\ ;\ y := x$$
$$\{\ x = Y\ \wedge\ y = X\ \}\ .$$

The program is made of three assignments separated by a semi-colon. We read the first assignment, $z := x$, as "z becomes x" and it means that after execution, the value of z is the same as the value of x. The expressions between curly brackets correspond to assertions. An expression of the form $\{\ \mathrm{P}\ \}\ S\ \{\ \mathrm{Q}\ \}$ where P and Q are properties of the program variables and S is a program statement is called a *Hoare triple*. It means that if

the program variables satisfy P before the execution of the statement S, execution of S is guaranteed to terminate in a state where the variables satisfy property Q. In this case, if we execute the three assignments in a state that satisfies the property $x = X \ \wedge \ y = Y$, we are guaranteed to terminate in a state that satisfies $x = Y \ \wedge \ y = X$. In other words, the values of the variables x and y are interchanged.

More surprisingly, it is also possible to exchange the values of two variables without using any additional variables (it is, in fact, a well-known programming trick!). A conventional solution assumes that the values of the variables can be represented as sequences of bits and exploits the bitwise exclusive-or operation (here denoted by $\not\equiv$):

$$\{ \ x = X \ \wedge \ y = Y \ \}$$
$$x \ := \ x \not\equiv y \ \ ; \ \ y \ := \ x \not\equiv y \ \ ; \ \ x \ := \ x \not\equiv y$$
$$\{ \ x = Y \ \wedge \ y = X \ \} \ .$$

The exclusive-or operator corresponds to bitwise inequivalence. For example, the binary representations of the numbers 5 and 3 are, respectively, 101 and 011. Since $(0 \not\equiv 1) = (1 \not\equiv 0) = 1$ and $(0 \not\equiv 0) = (1 \not\equiv 1) = 0$, we have that $(101 \not\equiv 011) = 110$, that is, $5 \not\equiv 3 \ = \ 6$.

Suppose now that we want to write a program to exchange the values of two variables without using additional variables in a programming language that has no support for bitwise operators. Besides the bitwise exclusive-or, which other operators can we use to achieve the same result?

8.3 Prerequisites

Familiarity with bitwise operators like exclusive-or, with the calculational proof format and with associativity may be helpful.

8.4 Resolution and notes

In order to determine which properties of $\not\equiv$ are involved and which other operators can be used, let's change $\not\equiv$ to an arbitrary operator $\otimes$ and present all the relevant annotations. Working back from the postcondition to the precondition, we get the following annotated program:

$$\{ \ x = X \ \wedge \ y = Y \ \}$$

$$
\begin{array}{l}
\{\ (x \otimes y) \otimes ((x \otimes y) \otimes y)\ =\ Y\ \ \wedge\ \ (x \otimes y) \otimes y = X\ \} \\
x\ :=\ x \otimes y \\
\{\ x \otimes (x \otimes y)\ =\ Y\ \ \wedge\ \ x \otimes y = X\ \}; \\
y\ :=\ x \otimes y \\
\{\ x \otimes y\ =\ Y\ \ \wedge\ \ y\ =\ X\ \}; \\
x\ :=\ x \otimes y \\
\{\ x\ =\ Y\ \wedge\ y\ =\ X\ \}\ .
\end{array}
$$

Now, given the first assertion, we can rewrite the second one as the conjunction of the following two conditions:

$$(x \otimes y) \otimes ((x \otimes y) \otimes y) = y \quad , \quad \text{and}$$

$$(x \otimes y) \otimes y = x \quad .$$

We want to find properties of the operator $\otimes$ that make these conditions hold. Starting with the simpler condition (i.e., with the second one) and using square brackets to denote universal quantification over all free variables, we calculate:

$$
\begin{array}{ll}
 & (x \otimes y) \otimes y \\
= & \quad \{\quad \text{assume that } \otimes \text{ is associative,} \\
 & \qquad\quad \text{in order to isolate } x\quad \} \\
 & x \otimes (y \otimes y) \\
= & \quad \{\quad \text{assume that } \otimes \text{ is unitpotent, that is:} \\
 & \qquad\quad [\ z \otimes z\ =\ 1_{\otimes}\ ]\text{, where } 1_{\otimes} \text{ is the unit of } \otimes\quad \} \\
 & x\ .
\end{array}
$$

The second condition is thus satisfied by assuming that $\otimes$ is associative and unitpotent. The first condition can be calculated using the same properties:

$$
\begin{array}{ll}
 & (x \otimes y) \otimes ((x \otimes y) \otimes y) \\
= & \quad \{\quad \otimes \text{ is associative}\quad \} \\
 & ((x \otimes y) \otimes (x \otimes y)) \otimes y \\
= & \quad \{\quad \otimes \text{ is unitpotent}\quad \} \\
 & y\ .
\end{array}
$$

Thus the correctness of the program presented above follows from the following two properties of $\otimes$:

$\otimes$ is associative , and

$\otimes$ is unitpotent .

Clearly, the bitwise exclusive-or is suitable. But note that the bitwise equivalence (usually denoted by $\equiv$) can also be used.

Generalising $\otimes$

Note, however, that using only one operator is limiting. We now generalise $\otimes$ by replacing each occurrence with a separate operator. The new program and corresponding annotations become:

$\{\ x = X\ \wedge\ y = Y\ \}$

$\{\ (x \otimes y) \ominus ((x \otimes y) \oplus y) = Y\ \wedge\ (x \otimes y) \oplus y = X\ \}$

$x\ :=\ x \otimes y$

$\{\ x \ominus (x \oplus y)\ =\ Y\ \wedge\ x \oplus y\ =\ X\ \};$

$y\ :=\ x \oplus y$

$\{\ x \ominus y\ =\ Y\ \wedge\ y\ =\ X\ \};$

$x\ :=\ x \ominus y$

$\{\ x\ =\ Y\ \wedge\ y\ =\ X\ \}$.

Again, from the two initial assertions we get the two following conditions:

$(x \otimes y) \ominus ((x \otimes y) \oplus y) = y$, and

$(x \otimes y) \oplus y = x$.

As before, the goal is to investigate which properties of the operators make these conditions hold. Starting with the second condition, we calculate:

$(x \otimes y) \oplus y$

$=$ { assume that $\otimes$ associates with $\oplus$ }

$x \otimes (y \oplus y)$

$=$ { assume that $\oplus$ is unitpotent with respect to $\otimes$, that is:

$$\{\ [\ z \oplus z = 1_{\otimes}\ ]\text{, where } 1_{\otimes} \text{ is the unit of } \otimes\ \}$$

$$x\ .$$

Now, the first condition:

$$\begin{aligned}
&(x \otimes y) \ominus ((x \otimes y) \oplus y) \\
=&\quad \{\quad \text{previous calculation}\quad \} \\
&(x \otimes y) \ominus x \\
=&\quad \{\quad \text{assume that } \otimes \text{ is symmetric}\quad \} \\
&(y \otimes x) \ominus x \\
=&\quad \{\quad \text{assume that } \otimes \text{ associates with } \ominus\quad \} \\
&y \otimes (x \ominus x) \\
=&\quad \{\quad \text{assume that } \ominus \text{ is unitpotent with respect to } \otimes\quad \} \\
&y\ .
\end{aligned}$$

Note that the choices made in this calculation could be different. Exercise 8.6.1 is about a calculation that leads to different properties.

We thus conclude from the two previous calculations that our new program is correct if the following properties hold:

$\otimes$ is symmetric ,

$\otimes$ associates with $\oplus$,

$\otimes$ associates with $\ominus$,

$\oplus$ is unitpotent with respect to $\otimes$, and

$\ominus$ is unitpotent with respect to $\otimes$.

As we can see, operations $\oplus$ and $\ominus$ are identical with respect to these conditions. In fact, we can prove that these five properties imply that $\oplus$ and $\ominus$ are equal:

$$\begin{aligned}
&x \oplus y \\
=&\quad \{\quad \text{unitpotency of } \ominus \text{ with respect to } \otimes, \\
&\qquad \text{in order to introduce } \ominus\quad \} \\
&(x \oplus y) \otimes (y \ominus y) \\
=&\quad \{\quad \otimes \text{ associates with } \ominus\quad \}
\end{aligned}$$

$$
\begin{array}{ll}
 & ((x \oplus y) \otimes y) \ominus y \\
= & \quad \{ \quad \text{deferred proof obligation of } [\ (x \oplus y) \otimes y = x\ ]; \\
 & \qquad \text{see below} \quad \} \\
 & x \ominus y \ .
\end{array}
$$

The assumption in the last step can be easily proved from the other properties as follows:

$$
\begin{array}{ll}
 & (x \oplus y) \otimes y \\
= & \quad \{ \quad \otimes \text{ is symmetric} \quad \} \\
 & y \otimes (x \oplus y) \\
= & \quad \{ \quad \otimes \text{ associates with } \oplus \quad \} \\
 & (y \otimes x) \oplus y \\
= & \quad \{ \quad \otimes \text{ is symmetric} \quad \} \\
 & (x \otimes y) \oplus y \\
= & \quad \{ \quad \otimes \text{ associates with } \oplus \quad \} \\
 & x \otimes (y \oplus y) \\
= & \quad \{ \quad \oplus \text{ is unitpotent with respect to } \otimes \quad \} \\
 & x \ .
\end{array}
$$

Thus we write both $\oplus$ and $\ominus$ as $\oplus$ and our program becomes:

$$
\begin{array}{l}
\{\ x = X\ \wedge\ y = Y\ \} \\
x\ :=\ x \otimes y\ ; \\
y\ :=\ x \oplus y\ ; \\
x\ :=\ x \oplus y \\
\{\ x = Y\ \wedge\ y = X\ \}\ .
\end{array}
$$

Recall that this program is correct if the following properties hold:

$\otimes$ is symmetric ,

$\otimes$ associates with $\oplus$, and

$\oplus$ is unitpotent with respect to $\otimes$.

A simple refinement

An immediate corollary is that if we have a group with a symmetric operation $\otimes$, and if we define the operator $\oplus$ as

$$x \oplus y = x \otimes y^{-1} \quad ,$$

where y^{-1} is the inverse of y , then the above properties will hold, as the reader can verify. If we take, for instance, real addition for $\otimes$ and real subtraction for $\oplus$, we get the following program:

$$\{ \; x = X \; \wedge \; y = Y \; \}$$
$$x \; := \; x+y \; ;$$
$$y \; := \; x-y \; ;$$
$$x \; := \; x-y$$
$$\{ \; x = Y \; \wedge \; y = X \; \} \; .$$

Note that, in practise, we have to take into account the size of the variables to avoid overflow problems. (Overflow occurs when an operation attempts to create a value that is larger than the maximum value that can be represented within the available storage space.) We omit considerations on overflows for brevity and simplicity.

8.5 Notes for the teacher

Model the problem and annotate the program The goal of the problem is to investigate which other operators can be used in the place of $\neq$, so the first thing we do is to change to an arbitrary operator $\otimes$:

$$\{ \; x = X \; \wedge \; y = Y \; \}$$
$$x \; := \; x \otimes y \quad ; \quad y \; := \; x \otimes y \quad ; \quad x \; := \; x \otimes y$$
$$\{ \; x = Y \; \wedge \; y = X \; \}$$

Consider now the last assignment and the postcondition. What is the weakest precondition that, after execution of the last assignment, establishes the postcondition? To answer this question, we use the *assignment axiom*:

$$\{ \; Q[x \; := \; e] \; \} \; x \; := \; e \; \{ \; Q \; \} \quad .$$

The assignment axiom is a very straightforward rule; the key is to work backwards from postconditions to preconditions. Suppose the assignment $x := e$ is required to establish the postcondition Q. The postcondition is some Boolean-valued expression in the program variables, one of which is x. After the assignment, x will have the value of expression e before the assignment. So, if Q is to apply to x after the assignment, Q should apply to e before the assignment. The condition Q[$x := e$] is called the *weakest precondition*. The teacher can provide more examples to illustrate how to calculate weakest preconditions. For example, if we require the assignment $k := k{+}1$ to establish the postcondition $0 < k$, its weakest precondition is $0 < k{+}1$. Once the students understand the concept (and we think it can only be done with practice), we are ready to annotate the program with the weakest precondition relative to the last assignment:

$$\{\ x = X\ \wedge\ y = Y\ \}$$
$$x := x \otimes y\ \ ;\ \ y := x \otimes y\ \ ;$$
$$\{\ x \otimes y = Y\ \ \wedge\ \ y = X\ \}$$
$$x := x \otimes y$$
$$\{\ x = Y\ \wedge\ y = X\ \}$$

Working back from the postcondition to the precondition, we can now repeat the same for the other assignments and annotate the program as follows:

$$\{\ x = X\ \wedge\ y = Y\ \}$$
$$\{\ (x \otimes y) \otimes ((x \otimes y) \otimes y) = Y\ \ \wedge\ \ (x \otimes y) \otimes y = X\ \}$$
$$x := x \otimes y$$
$$\{\ x \otimes (x \otimes y) = Y\ \ \wedge\ \ x \otimes y = X\ \};$$
$$y := x \otimes y$$
$$\{\ x \otimes y = Y\ \ \wedge\ \ y = X\ \};$$
$$x := x \otimes y$$
$$\{\ x = Y\ \wedge\ y = X\ \}\ .$$

Investigate properties Now, given the first assertion, we can rewrite the second one as the conjunction of the following two conditions (we replace X by x and Y by y):

$$(x \otimes y) \otimes ((x \otimes y) \otimes y) = y\ \ ,\ \ \text{and}$$

$$(x \otimes y) \otimes y = x \quad .$$

The goal is to find properties of the operator $\otimes$ that make these conditions hold. The teacher has to be sure that the students understand this. Starting with the simpler condition (i.e., with the second one), the goal is to find properties of the operator $\otimes$ that guarantee $(x \otimes y) \otimes y = x$. One way of doing that is to "let the symbols do the work":

$$\begin{array}{ll} & (x \otimes y) \otimes y \\ = & \quad \{ \quad \text{assume that } \otimes \text{ is associative,} \\ & \qquad \text{in order to isolate } x \quad \} \\ & x \otimes (y \otimes y) \\ = & \quad \{ \quad \text{assume that } \otimes \text{ is unitpotent, that is:} \\ & \qquad [\ z \otimes z = 1_{\otimes}\ ] \text{, where } 1_{\otimes} \text{ is the unit of } \otimes \quad \} \\ & x \ . \end{array}$$

Note that we use square brackets to denote universal quantification over all free variables (the teacher may want to write it differently). The second condition is thus satisfied by assuming that $\otimes$ is associative and unitpotent. We recommend the teacher to stress the investigative approach that the calculation follows: we are postulating properties based on the shape of the expressions. Syntactic-guided investigations are practical and useful. The first condition can be calculated using the same properties:

$$\begin{array}{ll} & (x \otimes y) \otimes ((x \otimes y) \otimes y) \\ = & \quad \{ \quad \otimes \text{ is associative} \quad \} \\ & ((x \otimes y) \otimes (x \otimes y)) \otimes y \\ = & \quad \{ \quad \otimes \text{ is unitpotent} \quad \} \\ & y \ . \end{array}$$

Thus the correctness of the program presented above follows from the following two properties of $\otimes$:

$$\otimes \text{ is associative} \quad \text{, and}$$

$$\otimes \text{ is unitpotent} \quad .$$

Clearly, the bitwise exclusive-or is suitable. But note that the bitwise equivalence (usually denoted by $\equiv$) can also be used. (Before advancing to the next section, the teacher should guarantee that the students understood what was done so far.)

Generalise the operator Using only one operator is limiting, so we now generalise $\otimes$ by replacing each occurrence with a separate operator. The new program and corresponding annotations become:

$$
\begin{array}{l}
\{\ x = X\ \wedge\ y = Y\ \} \\
\{\ (x \otimes y) \ominus ((x \otimes y) \oplus y) = Y\ \ \wedge\ \ (x \otimes y) \oplus y = X\ \} \\
x\ :=\ x \otimes y \\
\{\ x \ominus (x \oplus y)\ =\ Y\ \ \wedge\ \ x \oplus y\ =\ X\ \}; \\
y\ :=\ x \oplus y \\
\{\ x \ominus y\ =\ Y\ \ \wedge\ \ y\ =\ X\ \}; \\
x\ :=\ x \ominus y \\
\{\ x\ =\ Y\ \wedge\ y\ =\ X\ \}\ .
\end{array}
$$

(We recommend the teacher to let the students practise the assignment axiom by asking them to annotate the program.) Again, from the two initial assertions we get the two following conditions:

$$
\begin{array}{l}
(x \otimes y) \ominus ((x \otimes y) \oplus y) = y\quad \text{, and} \\
(x \otimes y) \oplus y = x\qquad .
\end{array}
$$

As before, the goal is to investigate which properties of the operators make these conditions hold. Starting with the second condition, we calculate:

$$
\begin{array}{ll}
 & (x \otimes y) \oplus y \\
= & \quad \{\quad \text{assume that } \otimes \text{ associates with } \oplus, \\
 & \qquad\ \ \text{in order to isolate } x\quad \} \\
 & x \otimes (y \oplus y) \\
= & \quad \{\quad \text{assume that } \oplus \text{ is unitpotent with respect to } \otimes, \text{ that is:} \\
 & \qquad\ \ [\ z \oplus z = 1_{\otimes}\ ], \text{ where } 1_{\otimes} \text{ is the unit of } \otimes\quad \} \\
 & x\ .
\end{array}
$$

Now, the first condition:

$$
\begin{array}{ll}
 & (x \otimes y) \ominus ((x \otimes y) \oplus y) \\
= & \quad \{\quad \text{previous calculation}\quad \}
\end{array}
$$

$$
\begin{array}{ll}
 & (x \otimes y) \ominus x \\
= & \quad \{ \quad \text{assume that } \otimes \text{ is symmetric} \quad \} \\
 & (y \otimes x) \ominus x \\
= & \quad \{ \quad \text{assume that } \otimes \text{ associates with } \ominus \quad \} \\
 & y \otimes (x \ominus x) \\
= & \quad \{ \quad \text{assume that } \ominus \text{ is unitpotent with respect to } \otimes \quad \} \\
 & y \ .
\end{array}
$$

Note that the choices made in this calculation could be different. Exercise 8.6.1 is about a calculation that leads to different properties.

We thus conclude from the two previous calculations that our new program is correct if the following properties hold:

$\otimes$ is symmetric ,

$\otimes$ associates with $\oplus$,

$\otimes$ associates with $\ominus$,

$\oplus$ is unitpotent with respect to $\otimes$, and

$\ominus$ is unitpotent with respect to $\otimes$.

The teacher should remark that the operations $\oplus$ and $\ominus$ are identical with respect to these conditions. In fact, we can prove that these five properties imply that $\oplus$ and $\ominus$ are equal:

$$
\begin{array}{ll}
 & x \oplus y \\
= & \quad \{ \quad \text{unitpotency of } \ominus \text{ with respect to } \otimes, \\
 & \qquad \text{in order to introduce } \ominus \quad \} \\
 & (x \oplus y) \otimes (y \ominus y) \\
= & \quad \{ \quad \otimes \text{ associates with } \ominus \quad \} \\
 & ((x \oplus y) \otimes y) \ominus y \\
= & \quad \{ \quad \text{deferred proof obligation of } [\ (x \oplus y) \otimes y = x\ ]; \\
 & \qquad \text{see below} \quad \} \\
 & x \ominus y \ .
\end{array}
$$

The assumption in the last step can be easily proved from the other properties as follows:

$$
\begin{array}{ll}
 & (x \oplus y) \otimes y \\
= & \quad \{ \quad \otimes \text{ is symmetric} \quad \} \\
 & y \otimes (x \oplus y) \\
= & \quad \{ \quad \otimes \text{ associates with } \oplus \quad \} \\
 & (y \otimes x) \oplus y \\
= & \quad \{ \quad \otimes \text{ is symmetric} \quad \} \\
 & (x \otimes y) \oplus y \\
= & \quad \{ \quad \otimes \text{ associates with } \oplus \quad \} \\
 & x \otimes (y \oplus y) \\
= & \quad \{ \quad \oplus \text{ is unitpotent with respect to } \otimes \quad \} \\
 & x \ .
\end{array}
$$

Thus we write both $\oplus$ and $\ominus$ as $\oplus$ and our program becomes:

$$
\begin{array}{l}
\{ \ x = X \ \wedge \ y = Y \ \} \\
x \ := \ x \otimes y \ ; \\
y \ := \ x \oplus y \ ; \\
x \ := \ x \oplus y \\
\{ \ x = Y \ \wedge \ y = X \ \} \ .
\end{array}
$$

Recall that this program is correct if the following properties hold:

$$
\begin{array}{l}
\otimes \text{ is symmetric} \quad , \\
\otimes \text{ associates with } \oplus \quad \text{, and} \\
\oplus \text{ is unitpotent with respect to } \otimes \quad .
\end{array}
$$

A simple refinement An immediate corollary is that if we have a group with a symmetric operation $\otimes$, and if we define the operator $\oplus$ as

$$x \oplus y = x \otimes y^{-1} \quad ,$$

where y^{-1} is the inverse of y , then the above properties will hold, as the reader can verify. Depending on the level of the students, the teacher may want to omit this corollary. However, we suggest the discussion of the following example. If we take, for instance, real addition for $\otimes$ and real subtraction for $\oplus$, we get the following program:

$$\{\ x = X\ \wedge\ y = Y\ \}$$
$$x\ :=\ x+y\ ;$$
$$y\ :=\ x-y\ ;$$
$$x\ :=\ x-y$$
$$\{\ x = Y\ \wedge\ y = X\ \}\ .$$

Note that, in practise, we have to take into account the size of the variables to avoid overflow problems. (Overflow occurs when an operation attempts to create a value that is larger than the maximum value that can be represented within the available storage space.) We omit considerations on overflows for brevity and simplicity.

8.5.1 Questions that the teacher should ask

Model the problem and annotate the program

- *Do you understand the problem? Do you understand the trick based on the exclusive-or operator?*

  The teacher has to be sure that the students understand what is required. Going through one or two examples may be enough for most students.

- *What is the weakest precondition that, after execution of the last assignment, establishes the postcondition?*

  The goal of this question is to help introducing the assignment axiom. It is important that the students learn how to use the assignment axiom to calculate weakest preconditions (at a later stage, the teacher may want to show them how to use it to calculate assignments).

- *Do you understand the assignment axiom?*

  The assignment axiom is a fundamental rule to reason about programs. Therefore, the teacher has to be sure that the students understand it before proceeding.

Investigate properties

- *What can we conclude if the two properties* $(x \otimes y) \otimes ((x \otimes y) \otimes y) = y$ *and* $(x \otimes y) \otimes y = x$ *are satisfied?*

  The students should understand that if these two conditions are true initially, the values of the variables will be exchanged on termination. The teacher has to be sure that they understand the goal: to investigate properties of the operator that guarantee these two conditions.

- *We want to prove that* $(x \otimes y) \otimes y = x$*. How can we isolate the* x *in* $(x \otimes y) \otimes y$*?*

  We have almost reduced the problem to a syntactic problem. The teacher should ask what properties allow the isolation of x. The goal is to help the students reach the conclusion that we can use associativity.

- *How can we eliminate the subexpression* $y \otimes y$ *in* $x \otimes (y \otimes y)$*?*

  Again, the goal of this question is to help the students reach the conclusion that, if we assume that the operator is unitpotent, we can eliminate the subexpression $y \otimes y$.

- *When is the program shown correct?*

  The students should understand that the program is correct if the conditions that we have assumed are true. So, any operator that satisfies these conditions, can be used to exchange the values of two variables without using any additional variables.

Generalise the operator

- *Do you notice any symmetry between the operators* $\oplus$ *and* $\ominus$ *?*

  The goal of this question is to help the students realise that the operators $\oplus$ and $\ominus$ are identical with respect to the conditions shown. The teacher should stress that this suggests the presence of unnecessary detail and then help the students prove that they are indeed the same.

A simple refinement

- *Can you think of any known operators that can be used as instances of* $\otimes$ *and* $\oplus$*?*

  We suggest the teacher to let the students suggest instances for the operators, rather than showing them which operators can be used. We believe that most students will realise that addition and subtraction can be used (the same for multiplication and division).

8.5.2 Questions that the teacher should not ask

Investigate properties

- *Can you see that if we use associativity in the expression $(x \otimes y) \otimes y$, we isolate the x?*

  We recommend the teacher to let the students think about properties that allow the isolation of the variable x, rather than disclosing it.

- *Can we use unitpotency to eliminate the subexpression $y \otimes y$ in $x \otimes (y \otimes y)$?*

  For the same reason as above, this question is not recommended, because it discloses the property that has to be postulated.

A simple refinement

- *Can we use addition and subtraction (resp. multiplication and division) as instances of $\otimes$ and $\oplus$?*

  We recommend the teacher to let the students suggest instances for $\otimes$ and $\oplus$. If they do not suggest any operators, the teacher may provide some clues. It can also be useful to test the students' suggestions with particular examples.

8.5.3 Concepts that the teacher should introduce

Assignment axiom

Invariant

Non-determinism

Postcondition

Weakest precondition

8.6 Extensions and exercises

Exercise 8.6.1 (Eliminating the symmetry requirement) In page 256, we prove that

$$(x \otimes y) \ominus ((x \otimes y) \oplus y) = y \quad ,$$

using the fact that $\otimes$ is symmetric. Can you prove it without assuming that $\otimes$ is symmetric?

□

8.7 Solutions to extensions and exercises

8.6.1 Another possible calculation is:

$$
\begin{array}{ll}
 & (x \otimes y) \ominus ((x \otimes y) \oplus y) \\
= & \quad \{ \quad \ominus \text{ associates with } \oplus \quad \} \\
 & ((x \otimes y) \ominus (x \otimes y)) \oplus y \\
= & \quad \{ \quad \ominus \text{ is unitpotent with respect to } \oplus \quad \} \\
 & y \ .
\end{array}
$$

We then conclude that $\otimes$ does not need to be symmetric and that the program with three operations is correct if

$\otimes$ associates with $\oplus$,

$\ominus$ associates with $\oplus$,

$\oplus$ is unitpotent with respect to $\otimes$, and

$\ominus$ is unitpotent with respect to $\oplus$.

□

8.8 Further reading

We recommend [Bac03, Chapter 9] for more details about the assignment axiom. In the page 124 of the same book, the reader may also find a short discussion on weakest preconditions.

SCENARIO 9

The Chameleons of Camelot

9.1 Brief description and goals

This scenario presents a generalisation of the problem "The Chameleons of Camelot", found in [Hon97, p. 140] (a more recent and accessible reference is [Win09]). Its goal is to help students recognise, model, and solve algorithmic problems. The solution is goal-oriented and explores an invariant of the underlying non-deterministic algorithm. It is also an example of problem decomposition and it can be used to convey the notions of loop, guard, postcondition, and non-determinism. We also show *how* we can achieve the goal, rather than just showing it is possible to achieve it. The constructive argument involves a discussion on program termination that can be used to introduce the concept of bound function.

9.2 Problem

On the island of Camelot there are three different types of chameleons: grey chameleons, brown chameleons, and crimson chameleons. Whenever two chameleons of different colours meet, they both change colour to the third colour.

For which number of grey, brown, and crimson chameleons is it possible to arrange a succession of meetings that results in all the chameleons displaying the same colour?

For example, if the number of the three different types of chameleons is 4, 7, and 19 (irrespective of the colour), we can arrange a succession of meetings that results in a monochromatic state (e.g. $(4\,,7\,,19)\to(6\,,6\,,18)\to(5\,,5\,,20)\to\cdots\to(0\,,0\,,30)$). On the other hand, if the number of chameleons is 1, 2, and 3, it is impossible to make them all display the same colour.

9.3 Prerequisites

Elementary knowledge of invariants, postconditions, and congruences.

9.4 Resolution and notes

This problem is clearly of algorithmic nature, since there is a process that makes the chameleons change colours. The algorithm behind it is non-deterministic and easily expressible; denoting the number of grey, brown, and crimson chameleons by g, b, and c, respectively, we can formalise it as follows:

$$
\begin{aligned}
&\mathsf{do}\quad 0<g\wedge 0<b \;\to\; g,b,c \;:=\; g{-}1,b{-}1,c{+}2\\
&\Box\quad 0<g\wedge 0<c \;\to\; g,b,c \;:=\; g{-}1,b{+}2,c{-}1\\
&\Box\quad 0<b\wedge 0<c \;\to\; g,b,c \;:=\; g{+}2,b{-}1,c{-}1\\
&\mathsf{od}\\
&\{\ P\ \}\quad .
\end{aligned}
$$

The algorithm consists of a single loop (enclosed between the keywords do and od) that executes while at least one of the three guards (the conditions at the left of the arrow $\to$) is satisfied. If more than one guard is satisfied, the block operator ($\Box$) ensures that only one of the three assignments is chosen non-deterministically. The first assignment, for example, corresponds to a meeting between a grey chameleon and a brown chameleon: provided that there are chameleons of both these colours, the number of grey chameleons (g) and brown chameleons (b) both decrease by 1, whilst the number of crimson chameleons (c) increases by 2. P is the postcondition and it should express that all chameleons display the same colour. A candidate for P is

$$(g=0\wedge b=0)\vee(g=0\wedge c=0)\vee(b=0\wedge c=0)\quad . \tag{9.4.1}$$

Although (9.4.1) is simple to express, it does not need to be guessed, since it corresponds to to the negation of the disjunction of the guards. In other words, if the loop stops executing it is because it reached a state satisfying (9.4.1). However, the algorithm is non-deterministic and is not guaranteed to terminate. Our goal is thus to make the algorithm more deterministic in a way that guarantees termination.

Now, instead of working directly with the final goal (9.4.1), we can think of how to get to intermediate states from which the problem is easy to solve. Towards that end, we need to determine from which states we can easily arrange meetings such that the

resulting states satisfy (9.4.1). For example, the simplest states we can think of are the ones where there is only one type of chameleon; in this case the goal is trivially satisfied (e.g. $(0\,,0\,,30)$). Similarly, the problem is easy to solve when the number of chameleons of different colours is the same (e.g. $(30\,,30\,,30)$); in this case, we can choose two colours and organise a meeting between all the chameleons of these two colours. More generally, for the states where at least two types of chameleons are equally numbered, we can arrange a meeting between all the chameleons of these two types. Formally, we can express these states as:

(9.4.2) $g = b \;\vee\; g = c \;\vee\; b = c$.

If the algorithm above reaches a state that satisfies this expression, it remains to arrange a meeting between all the chameleons of two equally numbered classes. Now that the problem is decomposed into two simpler parts, we can change and annotate the algorithm to reflect the decomposition:

do $g \neq b \wedge g \neq c \wedge b \neq c \;\rightarrow$

if $0 < g \wedge 0 < b \;\rightarrow\; g\,,b\,,c \;:=\; g{-}1\,,b{-}1\,,c{+}2$

□ $0 < g \wedge 0 < c \;\rightarrow\; g\,,b\,,c \;:=\; g{-}1\,,b{+}2\,,c{-}1$

□ $0 < b \wedge 0 < c \;\rightarrow\; g\,,b\,,c \;:=\; g{+}2\,,b{-}1\,,c{-}1$

fi

od

$\{\; g = b \;\vee\; g = c \;\vee\; b = c \;\}$;

Two classes of chameleons are equally numbered, so we can arrange a meeting between all the chameleons of these two classes. As a consequence, their colour changes to the third one.

(If there are only chameleons of one colour, there is nothing left to do.)

$\{\; (g = 0 \wedge b = 0) \vee (g = 0 \wedge c = 0) \vee (b = 0 \wedge c = 0) \;\}$.

The introduction of the if-statement is necessary, because we want to guarantee that the loop stops when it reaches a state satisfying (9.4.2). Also, we do not want the loop to execute when two classes of chameleons are equally numbered.

Having nothing else to play with, we can now investigate how (9.4.2) behaves under the three loop assignments. Taking the first disjunct, we calculate:

$(g = b)[g\,,b\,,c \;:=\; g{-}1\,,b{-}1\,,c{+}2]$

$$
\begin{array}{ll}
= & \{ \quad \text{substitution} \quad \} \\
& g-1 = b-1 \\
= & \{ \quad \text{cancellation} \quad \} \\
& g = b \quad ;
\end{array}
$$

$$
\begin{array}{ll}
& (g = b)[g\,, b\,, c \;:=\; g-1\,, b+2\,, c-1] \\
= & \{ \quad \text{substitution} \quad \} \\
& g-1 = b+2 \\
= & \{ \quad \text{cancellation} \quad \} \\
& g = b+3 \quad ;
\end{array}
$$

Finally,

$$
\begin{array}{ll}
& (g = b)[g\,, b\,, c \;:=\; g+2\,, b-1\,, c-1] \\
= & \{ \quad \text{substitution} \quad \} \\
& g+2 = b-1 \\
= & \{ \quad \text{cancellation} \quad \} \\
& g+3 = b \quad .
\end{array}
$$

From these calculations we can conclude that the number of grey and brown chameleons after any meeting is either the same, or it differs by 3. Consequently, after several meetings, the number of chameleons differs by a multiple of 3 (note that 0 is a multiple of 3). A concise way of expressing this fact is by using congruences[1]:

$$g \cong b \pmod 3 \quad .$$

Using the same reasoning for the other two disjuncts (see exercise 9.6.1), we conclude that an invariant of the loop is

$$g \cong b \pmod 3 \;\vee\; g \cong c \pmod 3 \;\vee\; b \cong c \pmod 3 \quad .$$

We can now extend the algorithm presented above with the invariant:

$$\{ \;\textbf{Invariant:}\; g \cong b \pmod 3 \;\vee\; g \cong c \pmod 3 \;\vee\; b \cong c \pmod 3 \;\}$$

[1] $g \cong b \pmod 3$ means that g and b differ by a multiple of 3, i.e., $3 \setminus g-b$.

do $g \neq b \wedge g \neq c \wedge b \neq c \rightarrow$

if $0 < g \wedge 0 < b \rightarrow g, b, c := g-1, b-1, c+2$

□ $0 < g \wedge 0 < c \rightarrow g, b, c := g-1, b+2, c-1$

□ $0 < b \wedge 0 < c \rightarrow g, b, c := g+2, b-1, c-1$

fi

od

$\{ g = b \vee g = c \vee b = c \}$;

Two classes of chameleons are equally numbered, so we can arrange a meeting between all the chameleons of these two classes. As a consequence, their colour changes to the third one.

(If there are only chameleons of one colour, there is nothing left to do.)

$\{ (g = 0 \wedge b = 0) \vee (g = 0 \wedge c = 0) \vee (b = 0 \wedge c = 0) \}$.

We can also immediately conclude that if the initial numbers of chameleons do not satisfy the invariant, that is, if no two initial numbers are congruent modulo 3, it is impossible to organise a succession of meetings that results in all the chameleons displaying the same colour (see exercise 9.6.2).

But we still have to show that from a state satisfying the invariant, it is possible to reach a state where two classes of chameleons are equally numbered. That is, we have to make it more deterministic in a way that guarantees termination. Recall that after each meeting, the difference between the number of chameleons is either the same or it differs by 3. Therefore, if the difference between the numbers of two types of chameleons is a multiple of 3, and if we can iteratively reduce that difference by 3, it is possible to make them equal. We now give an algorithm that shows *how* to do it. Suppose, without loss of generality, that the difference between the number of grey and brown chameleons is a multiple of 3 and that $g < b$. Then, while there are crimson chameleons and $g \neq b$, we organise successive meetings between brown and crimson chameleons. This reduces the difference between the number of grey and brown chameleons by a multiple of 3. If there are no crimson chameleons and $g \neq b$, we organise a meeting between one grey and one brown chameleon. This increases the number of crimson chameleons by 2 and we can now organise two meetings between brown and crimson chameleons. Formally, the algorithm that organises a succession of meetings that allow to reach a state where $g = b$ is:

$\{\ g \cong b \pmod 3\ \wedge\ g < b\ \}$

$\{$ **Bound-function:** $b-g$ $\}$

do $g \neq b \rightarrow$ if $c \neq 0 \rightarrow g, b, c := g+2, b-1, c-1$

$\square\ c = 0 \rightarrow g, b, c := g-1, b-1, c+2$;

$g, b, c := g+2, b-1, c-1$

fi

od

$\{\ g = b\ \}$

The algorithm executes while $g \neq b$ and it terminates, because the function $b-g$, which is initially a positive multiple of 3, decreases by 3 at each iteration. This guarantees that the algorithm reaches a state where $b-g = 0$, that is, where $g = b$.

In conclusion, any initial values g, b and c that satisfy the invariant

$$g \cong b \pmod 3\ \vee\ g \cong c \pmod 3\ \vee\ b \cong c \pmod 3\ ,$$

allow a succession of meetings that results in all the chameleons displaying the same colour.

9.5 Notes for the teacher

Model the problem This problem is clearly of algorithmic nature, since there is a process that makes the chameleons change colours. To be sure that the students understand the problem, we recommend the teacher to go through one or two examples (the examples shown in section 9.2 can be useful).

The next step is to model the underlying algorithm. The teacher should emphasise that we are interested in the number of existing chameleons and in how that number evolves over time. This initial discussion should naturally lead to the introduction of variables to represent the number of the different types of chameleons; and of assignments to model the way in which these variables can change. It is a good idea to accompany an assignment like the following

$$g, b, c := g-1, b-1, c+2\ ,$$

with words explaining its informal meaning (e.g. "This assignment represents the meeting between a grey chameleon and a brown chameleon. The number of grey

chameleons and brown chameleons are both decreased by 1 and the number of crimson chameleons is increased by 2.").

At this point, the teacher can recall that the variables are natural numbers (because we can not have a negative number of chameleons) and ask if the assignment above is entirely correct. The discussion should lead to the notion of guard and to the introduction of the guarded assignment:

$$0 < g \wedge 0 < b \;\rightarrow\; g, b, c \;:=\; g{-}1, b{-}1, c{+}2 \quad .$$

Again, an informal explanation can be useful (e.g. "We can only arrange a meeting between two brown and crimson chameleons when there is at least one of each.").

Now, the teacher can ask the students to write down formally the other two meetings and introduce the *block* operator ($\Box$) for non-deterministic choice:

$$\begin{aligned} & 0 < g \wedge 0 < b \;\rightarrow\; g, b, c \;:=\; g{-}1, b{-}1, c{+}2 \\ \Box\; & 0 < g \wedge 0 < c \;\rightarrow\; g, b, c \;:=\; g{-}1, b{+}2, c{-}1 \\ \Box\; & 0 < b \wedge 0 < c \;\rightarrow\; g, b, c \;:=\; g{+}2, b{-}1, c{-}1 \end{aligned}$$

This expression represents a non-deterministic choice of one of three assignments: if more than one guard is satisfied, then the operator $\Box$ selects one at random to execute. So, if there are, for example, 4 green chameleons, 7 brown chameleons, and 19 crimson chameleons, all the three assignments can be executed (but only one will).

The teacher should explain that the above expression executes only once, i.e., once an assignment is selected, it is executed, and the process stops. This motivates the introduction of loops:

$$\begin{aligned} & \mathsf{do}\;\; 0 < g \wedge 0 < b \;\rightarrow\; g, b, c \;:=\; g{-}1, b{-}1, c{+}2 \\ \Box\;\; & 0 < g \wedge 0 < c \;\rightarrow\; g, b, c \;:=\; g{-}1, b{+}2, c{-}1 \\ \Box\;\; & 0 < b \wedge 0 < c \;\rightarrow\; g, b, c \;:=\; g{+}2, b{-}1, c{-}1 \\ & \mathsf{od} \quad . \end{aligned}$$

An informal explanation can be useful (e.g. "The $\mathsf{do} \cdots \mathsf{od}$ means that one of the assignments will be repeatedly chosen to be executed until all the guards evaluate to false.").

Determine the postcondition and decompose the problem Now that we have modelled the underlying algorithm, we have to express our goal. The answer comes directly from the problem statement: "For which number of grey, brown, and crimson

chameleons is it possible to arrange a succession of meetings that results in all the chameleons displaying the same colour?". So we need to express formally that all the chameleons display the same colour. One alternative is:

(9.5.1) $(g = 0 \wedge b = 0) \vee (g = 0 \wedge c = 0) \vee (b = 0 \wedge c = 0)$.

Again, an informal description can be useful (e.g. "The first disjunct means that there are only crimson chameleons, the second means that there only brown chameleons, and the third means that there are only green chameleons. Their disjunction means that at least one of these statements is true.").

Note that (9.5.1) does not need to be guessed, since it corresponds to to the negation of the disjunction of the guards. In other words, the loop stops executing when it reaches a state satisfying (9.5.1). Depending on the level of the students, the teacher may calculate (9.5.1) from the guards. One way of doing it is:

$$
\begin{array}{ll}
 & \neg((0 < g \wedge 0 < b) \vee (0 < g \wedge 0 < b) \vee (0 < g \wedge 0 < b)) \\
= & \quad \{ \quad \text{De Morgan's rule and the variables are natural numbers} \quad \} \\
 & (g = 0 \vee b = 0) \wedge (b = 0 \vee c = 0) \wedge (c = 0 \vee g = 0) \\
= & \quad \{ \quad \text{distributivity (several times)} \quad \} \\
 & (g = 0 \wedge b = 0 \wedge c = 0) \vee (g = 0 \wedge b = 0) \vee (g = 0 \wedge c = 0) \vee (b = 0 \wedge c = 0) \\
= & \quad \{ \quad \text{the first disjunct implies the others} \quad \} \\
 & (g = 0 \wedge b = 0) \vee (g = 0 \wedge c = 0) \vee (b = 0 \wedge c = 0) \ .
\end{array}
$$

Now, instead of working directly with the final goal (9.5.1), we can think of how to get to intermediate states from which the problem is easy to solve. The teacher should lead the students to the observation that if two types of chameleons are equally numbered, we can arrange a meeting between all the chameleons of these two types. We suggest the teacher to start with some concrete examples until the students get there (e.g. $(0\,,0\,,0)$, $(5\,,3\,,5)$, and $(10\,,171\,,10)$). Formally, we can express these states as:

(9.5.2) $g = b \ \vee \ g = c \ \vee \ b = c$.

If the algorithm above reaches a state that satisfies this expression, it remains to arrange a meeting between all the chameleons of two equally numbered classes. Now that the problem is decomposed into two simpler parts, we can change and annotate the algorithm to reflect the decomposition:

$$\mathsf{do} \quad g \neq b \wedge g \neq c \wedge b \neq c \ \rightarrow$$

$$
\begin{array}{l}
\quad \textsf{if} \quad 0<g \wedge 0<b \;\rightarrow\; g,b,c \;:=\; g{-}1,b{-}1,c{+}2 \\
\quad \Box \quad 0<g \wedge 0<c \;\rightarrow\; g,b,c \;:=\; g{-}1,b{+}2,c{-}1 \\
\quad \Box \quad 0<b \wedge 0<c \;\rightarrow\; g,b,c \;:=\; g{+}2,b{-}1,c{-}1 \\
\quad \textsf{fi} \\
\textsf{od}
\end{array}
$$

$\{\; g=b \;\vee\; g=c \;\vee\; b=c \;\};$

Two classes of chameleons are equally numbered, so we can arrange a meeting between all the chameleons of these two classes. As a consequence, their colour changes to the third one.

(If there are only chameleons of one colour, there is nothing left to do.)

$\{\; (g=0 \wedge b=0) \vee (g=0 \wedge c=0) \vee (b=0 \wedge c=0) \;\}$.

The introduction of the if-statement is necessary, because we want to guarantee that the loop stops when it reaches a state satisfying (9.5.2). (We can say that (9.5.2) is a postcondition of the loop, but not of the algorithm.) Also, we do not want the loop to execute when two classes of chameleons are equally numbered.

The teacher should make clear that the non-deterministic loop is not guaranteed to terminate. The goal is thus to make the algorithm more deterministic in a way that guarantees termination.

Determine appropriate invariants Now that we have formalised our algorithm and goal, there is not much left to do other than to investigate how (9.5.2) behaves under the three loop assignments. A standard technique is to work from the postcondition, using the assignment axiom:

$$\{\; P[v := e] \;\}\; v \;:=\; e \;\{\; P \;\} \quad ,$$

where $v := e$ represents an assignment and P is the postcondition. Taking the first disjunct of (9.5.2), we calculate how it behaves under the three assignments:

$$
\begin{array}{ll}
 & (g=b)[g,b,c \;:=\; g{-}1,b{-}1,c{+}2] \\
= & \quad \{ \quad \text{substitution} \quad \} \\
 & g{-}1=b{-}1 \\
= & \quad \{ \quad \text{cancellation} \quad \}
\end{array}
$$

$$g = b \quad ;$$

$$\begin{array}{ll} & (g = b)[g, b, c \;:=\; g{-}1, b{+}2, c{-}1] \\ = & \quad \{ \quad \text{substitution} \quad \} \\ & g{-}1 = b{+}2 \\ = & \quad \{ \quad \text{cancellation} \quad \} \\ & g = b{+}3 \quad ; \end{array}$$

Finally,

$$\begin{array}{ll} & (g = b)[g, b, c \;:=\; g{+}2, b{-}1, c{-}1] \\ = & \quad \{ \quad \text{substitution} \quad \} \\ & g{+}2 = b{-}1 \\ = & \quad \{ \quad \text{cancellation} \quad \} \\ & g{+}3 = b \quad . \end{array}$$

The teacher should ask the students if they see any pattern in the calculations. The discussion should lead to the fact that the number of grey and brown chameleons after any meeting is either the same, or it differs by 3. Consequently, after several meetings, the number of chameleons differs by a multiple of 3 (note that 0 is a multiple of 3). A concise way of expressing this fact is by using congruences:

$$g \cong b \pmod 3 \quad .$$

Using the same reasoning for the other two disjuncts (see exercise 9.6.1), we conclude that an invariant of the loop is

$$g \cong b \pmod 3 \;\vee\; g \cong c \pmod 3 \;\vee\; b \cong c \pmod 3 \quad .$$

We can now extend the algorithm presented above with the invariant:

$$\begin{array}{l} \{ \;\textbf{Invariant:}\; g{\cong}b \pmod 3 \;\vee\; g{\cong}c \pmod 3 \;\vee\; b{\cong}c \pmod 3 \;\} \\ \textsf{do} \quad g \neq b \wedge g \neq c \wedge b \neq c \;\rightarrow \\ \qquad \textsf{if} \quad 0 < g \wedge 0 < b \;\rightarrow\; g, b, c \;:=\; g{-}1, b{-}1, c{+}2 \\ \qquad \Box \quad 0 < g \wedge 0 < c \;\rightarrow\; g, b, c \;:=\; g{-}1, b{+}2, c{-}1 \\ \qquad \Box \quad 0 < b \wedge 0 < c \;\rightarrow\; g, b, c \;:=\; g{+}2, b{-}1, c{-}1 \end{array}$$

fi

od

$\{\ g = b \ \vee\ g = c \ \vee\ b = c\ \}$;

Two classes of chameleons are equally numbered, so we can arrange a meeting between all the chameleons of these two classes. As a consequence, their colour changes to the third one.

(If there are only chameleons of one colour, there is nothing left to do.)

$\{\ (g = 0 \wedge b = 0) \vee (g = 0 \wedge c = 0) \vee (b = 0 \wedge c = 0)\ \}$.

Use the invariant to solve the problem We can also immediately conclude that if the initial numbers of chameleons do not satisfy the invariant, that is, if no two initial numbers are congruent modulo 3, it is impossible to organise a succession of meetings that results in all the chameleons displaying the same colour. It is important that the students understand what an invariant is and how it is being used here to make this conclusion. The exercise 9.6.2 can help the teacher to determine if the students understand how to use the invariant as a necessary condition.

But we still have to show that from a state satisfying the invariant, it is possible to reach a state where two classes of chameleons are equally numbered. That is, we have to make it more deterministic in a way that guarantees termination. We suggest the teacher to emphasise that, so far, we have only proved that the invariant is a necessary condition. The teacher could also ask the students if it is sufficient. If the students say no, the teacher can ask for a counter-example (which has to be invalid, since the invariant is a sufficient condition). If the students say yes, the teacher can ask how they can organise a succession of meetings to achieve the goal.

We now prove that the invariant is sufficient by constructing an algorithm that organises meetings to achieve the goal. Recall that after each meeting, the difference between the number of chameleons is either the same or it differs by 3. Therefore, if the difference between the numbers of two types of chameleons is a multiple of 3, and if we can iteratively reduce that difference by 3, it is possible to make them equal. We now give an algorithm that shows *how* to do it. Suppose, without loss of generality, that the difference between the number of grey and brown chameleons is a multiple of 3 and that $g < b$. Then, while there are crimson chameleons and $g \neq b$, we organise successive meetings between brown and crimson chameleons. This reduces the difference between the number of grey and brown chameleons by a multiple of 3. If there are

no crimson chameleons and $g \neq b$, we organise a meeting between one grey and one brown chameleon. This increases the number of crimson chameleons by 2 and we can now organise two meetings between brown and crimson chameleons. Formally, the algorithm that organises a succession of meetings that allow to reach a state where $g = b$ is:

$\{\ g \cong b \pmod 3\ \wedge\ g < b\ \}$

$\{$ **Bound-function:** $b-g$ $\}$

do $g \neq b \rightarrow$ if $c \neq 0 \rightarrow g, b, c := g+2, b-1, c-1$

$\square\ c = 0 \rightarrow g, b, c := g-1, b-1, c+2$;

$g, b, c := g+2, b-1, c-1$

fi

od

$\{\ g = b\ \}$

This algorithm can be used to introduce the concept of bound function, which is usually a natural-valued function on the program variables that is bounded below and that decreases at each iteration. In this case, the algorithm executes while $g \neq b$ and it terminates, because the function $b-g$, which is initially a positive multiple of 3, decreases by 3 at each iteration. This guarantees that the algorithm reaches a state where $b-g = 0$, that is, where $g = b$.

Depending on the goals and on the students, the teacher can formally prove that $b-g$ is indeed decreasing by 3. In general, we introduce an auxiliary variable C that denotes the value of the bound function before an iteration, and we prove that the value of the bound function after the iteration is less than C. In this case, we have to prove the two following properties:

$$(b-g)[g, b, c := g+2, b-1, c-1] < C \ \Leftarrow\ b-g = C \quad \text{, and}$$

$$(b-g)[g, b, c := g-1, b-1, c+2\ ;\ g, b, c := g+2, b-1, c-1] < C \ \Leftarrow\ b-g = C \quad .$$

The proof of the first property is:

$$\begin{array}{ll} & (b-g)[g, b, c := g+2, b-1, c-1] < C \\ = & \quad \{\quad \text{substitution} \quad \} \\ & b-g-3 < C \end{array}$$

$$
\begin{array}{ll}
= & \{ \quad b-g=C \quad \} \\
& C-3<C \\
= & \{ \quad \text{inequality} \quad \} \\
& \mathsf{true} \ .
\end{array}
$$

We leave the proof of the other property for the reader. (Note that the composition of the two assignments $g\,,b\,,c := g-1\,,b-1\,,c+2$ and $g\,,b\,,c := g+2\,,b-1\,,c-1$ corresponds to the assignment $g\,,b\,,c := g+1\,,b-2\,,c+1$.) We suggest the teacher to ask what happens to the algorithm when $g=b$. The students should understand why the algorithm stops.

In conclusion, any initial values g, b and c that satisfy the invariant

$$g \cong b \ (\text{mod } 3) \ \vee \ g \cong c \ (\text{mod } 3) \ \vee \ b \cong c \ (\text{mod } 3) \quad ,$$

allow a succession of meetings that results in all the chameleons displaying the same colour. We suggest the teacher to go through the initial examples once again to see which ones satisfy the invariant.

Discuss generalisations If the teacher has the opportunity to discuss generalisations, then he could start by asking the students how we can generalise the problem. In particular, we have two generalisations in view:

- **Generalise the number of colours:** Instead of three different types of chameleons, we can have n different types. This generalisation would increase the number of variables and assignments.
- **Generalise the number of chameleons that change colour:** Whenever two chameleons of different colours meet, we can reduce the number of chameleons of these two colours by m and increase the number of chameleons of the third colour by p. This generalisation is considered in exercise 9.6.4.

We advise the teacher to discuss some generalisations (even if they are not solved in the classroom), because the students get the perception that a problem is never really solved. Furthermore, the teacher can set some generalisations as projects or homework.

9.5.1 Questions that the teacher should ask

Model the problem

- *Do you understand the problem?*

  The teacher has to be sure that the students understand what is required. Going through one or two examples may be enough for most students.

- *Is this an algorithmic problem?*

  One of the basic and most important skills in algorithmic problem solving is to be able to identify problems of algorithmic nature. We suggest the teacher to ask this question explicitly, so that in subsequent problems students ask the same question to themselves.

- *What changes after each meeting and how can we represent that change?*

  The purpose of this question is to introduce variables and assignments. The students should realise by themselves that what changes after each meeting is the number of chameleons of certain colours. Once they get there, the teacher can introduce the names for the variables and ask how they change. The discussion should naturally lead to the introduction of the assignments.

- *Is the assignment*

  $$g, b, c \;:=\; g-1, b-1, c+2$$

  entirely correct? Remember: the variables are natural numbers!

  If the students do not see what is wrong with the assignment, we suggest the teacher to show some examples (e.g. "What happens if there is only one crimson chameleon?"). Once the students understand that we can only perform such an assignment when there are at least one green chameleon and one brown chameleon, the teacher can ask how would they express that restriction. The discussion should lead to the introduction of guards.

- *Are there any other possible assignments? Which assignments are these?*

  It is important to let the students work! So, the other two assignments should be done by the students, not the teacher.

- *If it is possible to arrange more than one meeting at one time, which one should we choose?*

  The students should understand that if it is possible to execute more than one assignment, there is no preferred one. In terms of reasoning, it is better to avoid distinctions. The teacher should introduce the block operator to express this non-determinism (depending on the students, it may be necessary to explain what non-determinism means).

Determine and discuss the postcondition

- *What is our goal? What do we want to prove? How do we express it formally?*

  Every time we are working in a goal-oriented fashion, this question should be asked explicitly. The teacher may need to help the students formalising the states where there are chameleons of only one colour; if that is the case, we suggest him to help with the first disjunct and let the students do the other two.

- *Can you think of any state for which the problem is easy to solve?*

  One way of simplifying the problem is to think of states from which it is easy to attain the goal. To help the students, we recommend the teacher to go through some examples for which the problem is easy to solve. Clearly, the simplest is when there are chameleons of only one colour: there is nothing left to do. If the chameleons of the three types are equally numbered, it is also easy to solve. More generally, if two classes of chameleons are equally numbered the problem is easy to solve.

- *Do you understand the structure of our algorithm?*

  The teacher should not proceed until the students understand how the argument is structured. In particular, it is important to understand the role of the intermediate state.

Determine appropriate invariants

- *Do you see any pattern in these three calculations? What is the relation between the number of grey and brown chameleons?*

  The goal of this question is to get to the fact that the number of grey and brown chameleons after any meeting is either the same or it differs by 3. The teacher can ask the students to calculate how the other two disjuncts behave under the three assignments (see exercise 9.6.1).

Use the invariant to solve the problem

- *Do you understand how the invariant is being used?*

  The teacher should be sure that most students understand what an invariant is and how the notion of invariance is being used to solve this problem. We recommend the teacher to do a final explanation of how the argument is structured and how it achieves our goal.

- *How can we use our argument to deal with the examples we have seen initially?*

  It is important to let the students see by themselves which examples satisfy the invariant and which don't. The teacher can also invent new examples to test their understanding..

Discuss generalisations

- *Can we generalise this problem? How?*

  Asking this question explicitly helps cultivating inquisitive minds. The students should realise that a problem is never really solved, as we can always raise new questions. Some generalisations can be set as homework.

9.5.2 Questions that the teacher should not ask

Model the problem

- *Can you see that we have to introduce variables for the number of chameleons as their number changes after each meeting?*

  The students should be lead to the introduction of the right variables. In general, it is better they feel that they find the answers by themselves.

- *Why is the assignment*

  $$g\,,b\,,c \;:=\; g{-}1\,,b{-}1\,,c{+}2$$

  wrong for natural numbers?

  Again, this question is obtrusive. The discussion should naturally lead to the fact that the variables represent natural numbers.

- *[Assuming the teacher wrote the other two assignments] Do these two assignments correctly model the other two types of meetings?*

  To learn how to solve problems, the students have to practise and write down their solutions. Once the first assignment is explained, the teacher should ask the students to write down the other two.

Determine and discuss the postcondition

- *[Assuming the teacher wrote the goal with no explanation] Can you see why this is our goal?*

Expressing the goal of the problem is one of the most important tasks of the solution. Therefore, we suggest the teacher to take some time here and to formalise the goal together with the students.

- *Can you see that if two different types of chameleons are equally numbered, the problem is easy to solve?*

  The teacher should start by asking the students which states make the problem easy to solve. With the help of some examples, we believe that most students will get to the fact that if two different types of chameleons are equally numbered, the problem is easy to solve.

Determine appropriate invariants

- *Can you see that the number of grey and brown chameleons after any meeting is either the same or it differs by* 3?

  Again, we suggest the teacher to let the students reach the answer by themselves. Perhaps going through each calculation independently and asking the students to express the relation between the variables helps.

Use the invariant to solve the problem

- *Can you see that this example satisfies the invariant and this one does not?*

  The teacher should let the students test the argument with the examples shown previously. This way, the teacher can assess if the students really understand the argument developed.

9.5.3 Concepts that the teacher should introduce

Assignment axiom

Invariant

Non-determinism

Postcondition

9.6 Extensions and exercises

Exercise 9.6.1 (Invariants) Prove that $g \cong c \pmod 3$ and $b \cong c \pmod 3$ are also invariants of the algorithm.

□

Exercise 9.6.2 (Subatomic particles) A bubble chamber contains three types of subatomic particles: 10 particles of type X, 11 particles of type Y, 111 particles of type Z. Whenever an X- and Y-particle collide, they both become Z-particles. Likewise, Y- and Z-particles collide and become X-particles and X- and Z-particles become $Y-$particles upon collision. Can the particles in the bubble chamber evolve so that only one type is present?

□

Exercise 9.6.3 (Total number of chameleons) What can you say about the total number of chameleons? That is, how does the value of $g+b+c$ change after each meeting?

□

Exercise 9.6.4 (Generalisation) On the island of Camelot there are three different types of chameleons: grey chameleons, brown chameleons, and crimson chameleons. Whenever two chameleons of different colours meet, the number of chameleons of these two colours is reduced by m, and the number of chameleons of the third colour is increased by p.

For which number of grey, brown, and crimson chameleons is it possible to arrange a succession of meetings that results in all the chameleons displaying the same colour?

□

9.7 Solutions to extensions and exercises

9.6.1 To determine if $g \cong c \pmod 3$ is an invariant, we have to determine how it behaves under the three loop assignments:

$$
\begin{array}{ll}
 & (g \cong c \pmod 3)[g\,,b\,,c \;:=\; g-1\,,b-1\,,c+2] \\
= & \quad \{ \quad \text{substitution} \quad \} \\
 & g-1 \cong c+2 \pmod 3
\end{array}
$$

$$
\begin{array}{ll}
= & \{ \quad \text{modular arithmetic} \quad \} \\
& g \cong c+3 \pmod 3 \\
= & \{ \quad \text{modular arithmetic} \quad \} \\
& g \cong c \pmod 3 \quad ;
\end{array}
$$

$$
\begin{array}{ll}
& (g \cong c \pmod 3)[g\,,b\,,c \;:=\; g-1\,,b+2\,,c-1] \\
= & \{ \quad \text{substitution} \quad \} \\
& g-1 \cong c-1 \pmod 3 \\
= & \{ \quad \text{modular arithmetic} \quad \} \\
& g \cong c \pmod 3 \quad ;
\end{array}
$$

Finally,

$$
\begin{array}{ll}
& (g \cong c \pmod 3)[g\,,b\,,c \;:=\; g+2\,,b-1\,,c-1] \\
= & \{ \quad \text{substitution} \quad \} \\
& g+2 \cong c-1 \pmod 3 \\
= & \{ \quad \text{modular arithmetic} \quad \} \\
& g+3 \cong c \pmod 3 \\
= & \{ \quad \text{modular arithmetic} \quad \} \\
& g \cong c \pmod 3 \quad .
\end{array}
$$

This means that the number of grey and crimson chameleons after any meeting is either the same, or it differs by 3. The other calculation is exactly the same, but with g replaced by b.

□

9.6.2 This problem is an instance of the one that we solved, but, instead of chameleons, we have subatomic particles. Because $10 \not\cong 11 \pmod 3$, $10 \not\cong 111 \pmod 3$, and $11 \not\cong 111 \pmod 3$, it is impossible to attain a configuration where only one type of particles is present.

We have found this problem in [Zei06, p. 100].

□

9.6.3 The total number of chameleons remains constant after each iteration of the loop

body. To prove it, we calculate how the value of $g{+}b{+}c$ behaves under the three loop assignments:

$$
\begin{array}{ll}
 & (g{+}b{+}c)[g\,,b\,,c\ :=\ g{-}1\,,b{-}1\,,c{+}2] \\
= & \quad \{\quad \text{substitution} \quad \} \\
 & (g{-}1){+}(b{-}1){+}(c{+}2) \\
= & \quad \{\quad \text{arithmetic} \quad \} \\
 & g{+}b{+}c \quad ;
\end{array}
$$

$$
\begin{array}{ll}
 & (g{+}b{+}c)[g\,,b\,,c\ :=\ g{-}1\,,b{+}2\,,c{-}1] \\
= & \quad \{\quad \text{substitution} \quad \} \\
 & (g{-}1){+}(b{+}2){+}(c{-}1) \\
= & \quad \{\quad \text{arithmetic} \quad \} \\
 & g{+}b{+}c \quad ;
\end{array}
$$

Finally,

$$
\begin{array}{ll}
 & (g{+}b{+}c)[g\,,b\,,c\ :=\ g{+}2\,,b{-}1\,,c{-}1] \\
= & \quad \{\quad \text{substitution} \quad \} \\
 & (g{+}2){+}(b{-}1){+}(c{-}1) \\
= & \quad \{\quad \text{arithmetic} \quad \} \\
 & g{+}b{+}c \quad .
\end{array}
$$

□

9.6.4 The algorithm behind this generalisation is similar to the one shown in section 9.4; denoting the number of grey, brown, and crimson chameleons by g, b, and c, respectively, we can formalise it as follows:

$$
\begin{array}{l}
\mathsf{do}\ \ g \neq b \wedge g \neq c \wedge b \neq c\ \rightarrow \\
\qquad \mathsf{if}\ \ m \leq g \wedge m \leq b\ \rightarrow\ g\,,b\,,c\ :=\ g{-}m\,,b{-}m\,,c{+}p \\
\qquad \Box\ \ m \leq g \wedge m \leq c\ \rightarrow\ g\,,b\,,c\ :=\ g{-}m\,,b{+}p\,,c{-}m \\
\qquad \Box\ \ m \leq b \wedge m \leq c\ \rightarrow\ g\,,b\,,c\ :=\ g{+}p\,,b{-}m\,,c{-}m \\
\qquad \mathsf{fi}
\end{array}
$$

od

$\{$ (9.4.2), i.e., $g = b \ \vee \ g = c \ \vee \ b = c$ $\}$;

Two classes of chameleons are equally numbered, so we can arrange a meeting between all the chameleons of these two classes. As a consequence, their colour changes to the third one.

(If there are only chameleons of one colour, there is nothing left to do.)

$\{ \ (g = 0 \wedge b = 0) \vee (g = 0 \wedge c = 0) \vee (b = 0 \wedge c = 0) \ \}$.

Repeating the analysis of how (9.5.2) behaves under the three assignments, we take the first disjunct, and we calculate:

$$
\begin{array}{ll}
 & (g = b)[g, b, c := g-m, b-m, c+p] \\
= & \quad \{ \quad \text{substitution} \quad \} \\
 & g-m = b-m \\
= & \quad \{ \quad \text{cancellation} \quad \} \\
 & g = b \quad ;
\end{array}
$$

$$
\begin{array}{ll}
 & (g = b)[g, b, c := g-m, b+p, c-m] \\
= & \quad \{ \quad \text{substitution} \quad \} \\
 & g-m = b+p \\
= & \quad \{ \quad \text{cancellation, associativity} \quad \} \\
 & g = b+(m+p) \quad ;
\end{array}
$$

Finally,

$$
\begin{array}{ll}
 & (g = b)[g, b, c := g+p, b-m, c-m] \\
= & \quad \{ \quad \text{substitution} \quad \} \\
 & g+p = b-m \\
= & \quad \{ \quad \text{cancellation, associativity} \quad \} \\
 & g+(m+p) = b \quad .
\end{array}
$$

This means that the number of grey and brown chameleons after any meeting is either the same, or it differs by $m+p$. Consequently, after several meetings, the number of chameleons differs by a multiple of $m+p$ (note that 0 is a multiple of $m+p$).

A concise way of expressing this fact is by using congruences:

$$g \cong b \pmod{(m+p)} \quad .$$

Using the same reasoning for the other two disjuncts (exercise left to the reader), we conclude that an invariant of the loop is

$$g \cong b \pmod{(m+p)} \ \vee\ g \cong c \pmod{(m+p)} \ \vee\ b \cong c \pmod{(m+p)} \quad .$$

We can now extend the algorithm presented above with the invariant:

$\{$ **Invariant:** $g \cong b \pmod{(m+p)} \ \vee\ g \cong c \pmod{(m+p)} \ \vee\ b \cong c \pmod{(m+p)}$ $\}$

do $g \neq b \wedge g \neq c \wedge b \neq c \ \rightarrow$

if $m \leq g \wedge m \leq b \ \rightarrow\ g, b, c \ :=\ g-m, b-m, c+p$

$\Box$ $m \leq g \wedge m \leq c \ \rightarrow\ g, b, c \ :=\ g-m, b+p, c-m$

$\Box$ $m \leq b \wedge m \leq c \ \rightarrow\ g, b, c \ :=\ g+p, b-m, c-m$

fi

od

$\{\ g = b \ \vee\ g = c \ \vee\ b = c\ \}$;

Two classes of chameleons are equally numbered, so we can arrange a meeting between all the chameleons of these two classes. As a consequence, their colour changes to the third one.

(If there are only chameleons of one colour, there is nothing left to do.)

$\{\ (g = 0 \wedge b = 0) \vee (g = 0 \wedge c = 0) \vee (b = 0 \wedge c = 0)\ \}$.

We can also immediately conclude that if the initial numbers of chameleons do not satisfy the invariant, that is, if no two initial numbers are congruent modulo $m+p$, it is impossible to organise a succession of meetings that results in all the chameleons displaying the same colour.

But we still have to show that from a state satisfying the invariant, it is possible to reach a state where two classes of chameleons are equally numbered. Recall that after each meeting, the difference between the number of chameleons is either the same or it differs by $m+p$. Therefore, if the difference between the numbers of two types of chameleons is a multiple of $m+p$, and if we can iteratively reduce that difference by $m+p$, it is possible to make them equal. We now give an algorithm that shows *how* to do it. Suppose, without loss of generality, that the difference between the number of

grey and brown chameleons is a multiple of $m+p$ and that $g < b$. Then, while there are crimson chameleons and $g \neq b$, we organise successive meetings between brown and crimson chameleons. This reduces the difference between the number of grey and brown chameleons by a multiple of $m+p$. If there are no crimson chameleons and $g \neq b$, we organise a meeting between one grey and one brown chameleon. This increases the number of crimson chameleons by p and we can now organise p meetings between brown and crimson chameleons. Formally, the algorithm that organises a succession of meetings that allow to reach a state where $g = b$ is:

$\{\ g \cong b \pmod{m+p}\ \wedge\ g < b\ \}$

$\{$ **Bound-function:** $b-g$ $\}$

do $g \neq b \rightarrow$ if $c \neq 0 \rightarrow g, b, c := g+p, b-m, c-m$

$\square\ c = 0 \rightarrow g, b, c := g-m, b-m, c+p$;

$g, b, c := g+p, b-m, c-m$

fi

od

$\{\ g = b\ \}$

The algorithm executes while $g \neq b$ and it terminates, because the function $b-g$, which is initially a positive multiple of $m+p$, decreases by $m+p$ at each iteration. This guarantees that the algorithm reaches a state where $b-g = 0$, that is, where $g = b$.

In conclusion, any initial values g, b and c that satisfy the invariant

$$g \cong b \pmod{(m+p)} \vee g \cong c \pmod{(m+p)} \vee b \cong c \pmod{(m+p)} \quad ,$$

allow a succession of meetings that results in all the chameleons displaying the same colour. Note that the main example of this scenario is an instance of this problem with m replaced by 1 and p replaced by 2.

□

9.8 Further reading

The problem presented in this scenario is a generalisation of the one found in [Hon97, page 140] (a more recent and accessible reference is [Win09]). The original statement is:

> On the island of Camelot there are 45 chameleons. At one time 13 of them are grey, 15 brown, and 17 crimson. However, whenever two chameleons

of different colors meet, they both change color to the third color. Thus, for example, if a grey and a brown chameleon were to be the first to meet, the count would change to 12 grey, 14 brown, and 19 crimson.

Is it possible to arrange a succession of meetings that would result in all the chameleons displaying the same color?

We recommend the teacher to illustrate the use of the invariant found in section 9.4 with these particular values. Also, it may be beneficial to read and compare Honsberger's solution with our solution.

SCENARIO 10

Will This Algorithm Terminate?

10.1 Brief description and goals

This scenario presents a problem from the St. Petersburg City Olympiad 1996, whose goal is to prove that a given algorithm terminates. It can be used to introduce the topic of program termination, the concept of bound function, and to help the students practise termination proofs.

10.2 Problem

Several positive integers are written on a blackboard. One can erase any two distinct integers and write their greatest common divisor and least common multiple instead. Prove that eventually the numbers will stop changing.

10.3 Prerequisites

Familiarity with the notions of divisibility, greatest common divisor, and least common multiple can be helpful. Some properties of conjunction, disjunction, and implication are used (De Morgan's rules, contrapositive, weakening, distributivity, and symmetry).

10.4 Resolution and notes

Our argument explores the algorithmic nature of the problem: we prove that the underlying algorithm terminates by finding a natural-valued function on the algorithm's variables that decreases at each iteration.

The first step is to model the problem and to formalise the underlying algorithm. Naming the blackboard as B, the formal transcription of the replacement process is:

$$\mathsf{do}\ \ \langle a,b \in B ::\ \ a \neq b \wedge \{a,b\} \neq \{a \triangle b\,,a \triangledown b\}\ \ \rightarrow\ \ a\,,b\ :=\ a \triangle b\,,a \triangledown b\ \rangle$$

$$\mathsf{od}$$

We write $a \triangledown b$ and $a \triangle b$ to denote the greatest common divisor and the least common multiple of a and b, respectively. The expression $\mathsf{do}\ g \rightarrow e\ \mathsf{od}$ represents a loop that executes e while the guard g evaluates to true. In words, the formal transcription means: given two distinct numbers written on the blackboard, a and b, replace them by $a \triangle b$ and $a \triangledown b$ as long as the set of numbers change (i.e., as long as $\{a,b\} \neq \{a \triangle b\,,a \triangledown b\}$). The loop terminates when there are no such numbers on the blackboard. Also, $a \backslash b$ means that a divides b, that is, there is an integer k such that $b = k \times a$. If a does not divide b, we write $a \not\backslash b$.

The shape of the second conjunct in the guard of the algorithm brings to our attention a somewhat known theorem:

(10.4.1) $\{a,b\} = \{a \triangle b\,,a \triangledown b\} \quad \equiv \quad a \backslash b \vee b \backslash a$.

This theorem is relevant because it means that if we choose any two numbers a and b such that $a \backslash b \vee b \backslash a$, then the collection of numbers remains unchanged. For example, if the two numbers to be changed are 10 and 5 , we replace them by the numbers $10 \triangledown 5$ and $10 \triangle 5$, which are, respectively, 5 and 10. Clearly, the collection of numbers remains unchanged. The theorem is also relevant because, in its contrapositive form, allows us to refine the guard of the algorithm:

(10.4.2) $\{a,b\} \neq \{a \triangle b\,,a \triangledown b\} \quad \equiv \quad a \not\backslash b \wedge b \not\backslash a$.

This means that we can safely focus only on the numbers a and b such that $a \not\backslash b \wedge b \not\backslash a$. For brevity, we write $a \# b$ instead of $a \not\backslash b \wedge b \not\backslash a$. We also call the set $\{a,b\}$ a *couple* whenever $a \# b$.

Also, note that we can remove the expression $a \neq b$ from the guard of the algorithm, since

$$a \neq b \quad \Leftarrow \quad \{a,b\} \neq \{a \triangle b\,,a \triangledown b\}\ .$$

Now, to help with our analysis, suppose S is the bag of all couples present in the blackboard, that is, for a and b in B,

(10.4.3) $\{a,b\} \in S \equiv a \# b$.

As an example, if the collection of numbers written on the blackboard is {7,7,10,15}, we have

$$S = \{\{7,10\},\{7,15\},\{7,10\},\{7,15\},\{10,15\}\} \quad .$$

Using this terminology, one way of formalising the algorithm is

$$\mathsf{do} \quad \{a,b\} \in S \quad \rightarrow \quad a\,,b := a \triangledown b\,, a \triangle b \quad \mathsf{od} \ .$$

Note that a and b are local names that have no meaning outside the loop. This formulation clearly shows that the loop body will execute while S is non-empty. This immediately suggests that to prove termination we have to prove that the size of S decreases at each iteration. To avoid confusion, we use S to denote the bag before an iteration and S' to denote the bag after the same iteration. The goal is to prove that $|S'| < |S|$. And this is indeed the case. The substitution of a and b by the numbers $a \triangledown b$ and $a \triangle b$ means that there is one couple $\{a,b\}$ in S that does not belong to S'. Moreover, the set $\{a \triangledown b\,, a \triangle b\}$ is not included in S', since $a \triangledown b \backslash a \triangle b$. Note, however, that S' may include couples of the shapes $\{a \triangledown b\,, c\}$ and $\{a \triangle b\,, c\}$, for some c, that are not elements of S. (We shall call these "new couples".) Our goal is thus to prove that, even with the inclusion of these new couples in S', $|S'|$ is smaller than $|S|$; in other words, we have to prove that the removal of the numbers a and b compensates the addition of new couples to S'. More precisely, we prove the following propositions:

(10.4.4) $\{a \triangledown b\,, c\} \in S' \ \wedge \ \{a \triangle b\,, c\} \in S' \ \Rightarrow \ \{a,c\} \in S \ \wedge \ \{b,c\} \in S \quad ;$

(10.4.5) $\{a \triangledown b\,, c\} \in S' \ \Rightarrow \ \{a,c\} \in S \ \vee \ \{b,c\} \in S \quad ;$

(10.4.6) $\{a \triangle b\,, c\} \in S' \ \Rightarrow \ \{a,c\} \in S \ \vee \ \{b,c\} \in S \quad .$

Proposition (10.4.4) means that if two new couples $\{a \triangledown b\,, c\}$ and $\{a \triangle b\,, c\}$ are elements of S', then $\{a,c\}$ and $\{b,c\}$ are elements of S. But because we have replaced a and b, $\{a,c\}$ and $\{b,c\}$ can not be elements of S'. Hence, the removal of a and b compensates the addition of the new couples to S'. Similarly, propositions (10.4.5) and (10.4.6) mean that for each new couple added to S', there is at least one couple that is an element of S but not of S'.

Now, before proving the propositions (10.4.4), (10.4.5), and (10.4.6), we investigate calculational rules that relate the operators $\triangledown$ and $\triangle$ with the relation $\backslash$. First, we have the contrapositives of the definitions of $\triangledown$ and $\triangle$:

(10.4.7) $a \triangle b \backslash c \ \equiv \ (a \backslash c) \vee (b \backslash c)$, and

(10.4.8) $(c \setminus a \triangledown b) \equiv (c \setminus a) \vee (c \setminus b)$.

Note that in these rules $\triangle$ and $\triangledown$ occur on the left and right of the $\setminus$ symbol, respectively. The rules where $\triangle$ and $\triangledown$ occur on the right and left, respectively, are:

(10.4.9) $(c \setminus a \triangle b) \Rightarrow (c \setminus a) \wedge (c \setminus b)$, and

(10.4.10) $(a \triangledown b \setminus c) \Rightarrow (a \setminus c) \wedge (b \setminus c)$.

(Note the use of implication, rather than equality, in these last two properties. We leave their proofs for the reader.)

Finally, using these calculational rules, the proofs of (10.4.4), (10.4.5), and (10.4.6) become simple exercises in calculational logic. Proposition (10.4.4) can be proved as:

$$
\begin{array}{ll}
 & \{a \triangledown b\,, c\} \in S' \ \wedge\ \{a \triangle b\,, c\} \in S' \\
= & \quad \{ \quad \text{definitions (10.4.3) and } \# \quad \} \\
 & (a \triangledown b \setminus c) \wedge (c \setminus a \triangledown b) \wedge (a \triangle b \setminus c) \wedge (c \setminus a \triangle b) \\
\Rightarrow & \quad \{ \quad \text{weakening} \quad \} \\
 & (a \triangledown b \setminus c) \wedge (c \setminus a \triangle b) \\
\Rightarrow & \quad \{ \quad \text{(10.4.10) and (10.4.9)} \quad \} \\
 & (a \setminus c) \wedge (b \setminus c) \wedge (c \setminus a) \wedge (c \setminus b) \\
= & \quad \{ \quad \text{symmetry and definitions (10.4.3) and } \# \quad \} \\
 & \{a,c\} \in S \ \wedge\ \{b,c\} \in S \ .
\end{array}
$$

Also, we prove (10.4.5) as follows:

$$
\begin{array}{ll}
 & \{a \triangledown b\,, c\} \in S' \\
= & \quad \{ \quad \text{definitions (10.4.3) and } \# \quad \} \\
 & (a \triangledown b \setminus c) \wedge (c \setminus a \triangledown b) \\
\Rightarrow & \quad \{ \quad \text{(10.4.10) and (10.4.8)} \quad \} \\
 & (a \setminus c) \wedge (b \setminus c) \wedge ((c \setminus a) \vee (c \setminus b)) \\
= & \quad \{ \quad \text{distributivity} \quad \} \\
 & ((a \setminus c) \wedge (b \setminus c) \wedge (c \setminus a)) \vee ((a \setminus c) \wedge (b \setminus c) \wedge (c \setminus b))
\end{array}
$$

$$
\begin{array}{ll}
\Rightarrow & \{ \quad \text{weakening} \quad \} \\
& ((a \backslash c) \wedge (c \backslash a)) \vee ((b \backslash c) \wedge (c \backslash b)) \\
= & \{ \quad \text{definitions (10.4.3) and } \sharp \quad \} \\
& \{a,c\} \in S \ \vee \ \{b,c\} \in S \ .
\end{array}
$$

The proof of (10.4.6) is very similar:

$$
\begin{array}{ll}
& \{a \triangle b \, , c\} \in S' \\
= & \{ \quad \text{definitions (10.4.3) and } \sharp \quad \} \\
& (a \triangle b \setminus c) \wedge (c \setminus a \triangle b) \\
\Rightarrow & \{ \quad \text{(10.4.9) and (10.4.7)} \quad \} \\
& ((a \backslash c) \vee (b \backslash c)) \wedge (c \backslash a) \wedge (c \backslash b) \\
= & \{ \quad \text{distributivity} \quad \} \\
& ((a \backslash c) \wedge (c \backslash a) \wedge (c \backslash b)) \vee ((b \backslash c) \wedge (c \backslash a) \wedge (c \backslash b)) \\
\Rightarrow & \{ \quad \text{weakening} \quad \} \\
& ((a \backslash c) \wedge (c \backslash a)) \vee ((b \backslash c) \wedge (c \backslash b)) \\
= & \{ \quad \text{definitions (10.4.3) and } \sharp \quad \} \\
& \{a,c\} \in S \ \vee \ \{b,c\} \in S \ .
\end{array}
$$

We can thus conclude that the size of S decreases at each iteration, which means that the algorithm will eventually terminate.

(This solution was jointly developed with Alexandra Mendes.)

10.5 Notes for the teacher

Short introduction to program termination Program termination is a fundamental topic in Computing and an active research area. Therefore, we recommend the teacher to start with a short introduction to the topic. In particular, we suggest the introduction to the concept of *bound function*. To ensure that we are making progress towards a termination condition, we usually define a bound function, which is a natural-valued function of the program variables that measures the size of the problem to be solved. A guarantee that the value of such a bound function is always decreased at each iteration is a guarantee that the number of times the algorithm (or loop) is executed is at most

the initial value of the bound function. In this scenario, we will use as a bound function the size of a given bag.

Model the problem Now, the first step in our solution is to model the problem and to formalise the underlying algorithm. Naming the blackboard as B, the formal transcription of the replacement process is:

$$\mathsf{do}\ \ \langle a, b \in B ::\ \ a \neq b\ \ \rightarrow\ \ a\,, b\ :=\ a \triangle b\,, a \triangledown b\ \rangle$$

$$\mathsf{od}$$

The teacher may use different notations, but we recommend the use of $a \triangledown b$ and $a \triangle b$ to denote the greatest common divisor and the least common multiple of a and b, respectively (these notations also help with a generalisation discussed later). It is important that the teacher makes sure that the students understand the formal transcription (it helps if each part of the algorithm is written as the several aspects of the problem statement are discussed).

We suggest the teacher to ask what would happen if we have the numbers 1 and 3. The goal is to help the students notice that in certain situations, the collection of numbers will not change. This suggests that we refine the model to allow replacements only when the set of numbers change. One way of achieving that is as follows:

$$\mathsf{do}\ \ \langle a, b \in B ::\ \ a \neq b \wedge \{a,b\} \neq \{a \triangle b\,, a \triangledown b\}\ \ \rightarrow\ \ a\,, b\ :=\ a \triangle b\,, a \triangledown b\ \rangle$$

$$\mathsf{od}$$

It may help the students to describe it in words: given two distinct numbers written on the blackboard, a and b, replace them by $a \triangle b$ and $a \triangledown b$ as long as the set of numbers change (i.e., as long as $\{a,b\} \neq \{a \triangle b\,, a \triangledown b\}$). The loop terminates when there are no such numbers on the blackboard.

Refine the model Now that we have a model, we can try to simplify it. We recommend the teacher to remark that the shape of the second conjunct in the guard of the algorithm is related with the following known theorem:

$$(10.5.1)\ \ \{a,b\} = \{a \triangle b\,, a \triangledown b\}\ \ \equiv\ \ a \backslash b \vee b \backslash a\ \ .$$

The students should understand that this theorem is relevant because it means that if we choose any two numbers a and b such that $a \backslash b \vee b \backslash a$, then the collection of numbers remains unchanged. For example, if the two numbers to be changed are 10 and 5 ,

we replace them by the numbers $10\triangledown 5$ and $10\triangle 5$, which are, respectively, 5 and 10. Clearly, the collection of numbers remains unchanged. The theorem is also relevant because, in its contrapositive form, allows us to refine the guard of the algorithm:

(10.5.2) $\{a,b\} \neq \{a\triangle b\,,a\triangledown b\} \quad \equiv \quad a \backslash b \wedge b \backslash a \quad .$

This means that we can safely focus only on the numbers a and b such that $a\backslash b \wedge b\backslash a$. For brevity, we recommend the teacher to write $a\sharp b$ instead of $a\backslash b \wedge b\backslash a$. We also call the set $\{a,b\}$ a *couple* whenever $a\sharp b$.

Also, note that we can remove the expression $a \neq b$ from the guard of the algorithm, since

$$a \neq b \quad \Leftarrow \quad \{a,b\} \neq \{a\triangle b\,,a\triangledown b\} \quad .$$

At this stage, we have the following model:

$$\mathsf{do}\ \ \langle a,b \in B :: \ \ a\sharp b \ \ \rightarrow \ \ a\,,b \ := \ a\triangle b\,,a\triangledown b\,\rangle$$

$$\mathsf{od}$$

Now, to help with our termination analysis, suppose S is the bag of all couples present in the blackboard, that is, for a and b in B,

(10.5.3) $\{a,b\} \in S \equiv a\sharp b \quad .$

Using this terminology, one way of formalising the algorithm is

$$\mathsf{do}\ \ \{a,b\} \in S \ \ \rightarrow \ \ a\,,b \ := \ a\triangledown b\,,a\triangle b \ \ \mathsf{od}\ .$$

A bag is sometimes called a multiset and it is a set in which elements may occur more than once (and the number of occurrences is significant). To help the students grasp the concept of bag and definition (10.5.3), we suggest the teacher to work out the value of S, when the collection of numbers written on the blackboard is, for example, $\{7,7,10,15\}$:

$$S = \{\{7,10\},\{7,15\},\{7,10\},\{7,15\},\{10,15\}\} \quad .$$

(Also, note that the bags $\{7,7,10,15\}$ and $\{7,10,15\}$ are different.)

The strategy follows from the model This new formulation clearly shows that the loop body will execute while S is non-empty. This immediately suggests that to prove termination, we have to prove that the size of S decreases at each iteration. The teacher should make clear that when S is empty, the loop terminates. Also, we recommend

the teacher to point out that the function giving the size of S is the bound function (see the short introduction) chosen for this algorithm. To avoid confusion, we shall use S to denote the bag before an iteration and S' to denote the bag after the same iteration. The goal is to prove that $|S'| < |S|$. We suggest the teacher to ask the students how we can prove it. Most students will probably remark that the substitution of a and b by the numbers $a \bigtriangledown b$ and $a \bigtriangleup b$ means that there is one couple $\{a,b\}$ in S that does not belong to S'. Also, it is easy to see that the set $\{a \bigtriangledown b\,, a \bigtriangleup b\}$ is not included in S', since $a \bigtriangledown b \backslash a \bigtriangleup b$. (If they do not make any of these remarks, we suggest the teacher to ask questions that can lead the students to them.)

Note, however, that S' may include couples of the shapes $\{a \bigtriangledown b\,, c\}$ and $\{a \bigtriangleup b\,, c\}$, for some c, that are not elements of S. (We shall call these "new couples".) The teacher should make sure that the students understand that new couples can form.

Our goal is thus to prove that, even with the inclusion of these new couples in S', $|S'|$ is smaller than $|S|$; in other words, we have to prove that the removal of the numbers a and b compensates the addition of new couples to S'. More precisely, we have to prove the following propositions:

(10.5.4) $\{a \bigtriangledown b\,, c\} \in S' \;\wedge\; \{a \bigtriangleup b\,, c\} \in S' \;\Rightarrow\; \{a,c\} \in S \;\wedge\; \{b,c\} \in S$;

(10.5.5) $\{a \bigtriangledown b\,, c\} \in S' \;\Rightarrow\; \{a,c\} \in S \;\vee\; \{b,c\} \in S$;

(10.5.6) $\{a \bigtriangleup b\,, c\} \in S' \;\Rightarrow\; \{a,c\} \in S \;\vee\; \{b,c\} \in S$.

Informal explanations of these properties can help the students. Proposition (10.5.4) means that if two new couples $\{a \bigtriangledown b\,, c\}$ and $\{a \bigtriangleup b\,, c\}$ are elements of S', then $\{a,c\}$ and $\{b,c\}$ are elements of S. But because we have replaced a and b, $\{a,c\}$ and $\{b,c\}$ can not be elements of S'. Hence, the removal of a and b compensates the addition of the new couples to S'. Similarly, propositions (10.5.5) and (10.5.6) mean that for each new couple added to S', there is at least one couple that is an element of S but not of S'.

Calculating the solution Now, before proving the propositions (10.5.4), (10.5.5), and (10.5.6), we investigate calculational rules that relate the operators $\bigtriangledown$ and $\bigtriangleup$ with the relation $\backslash$. First, we have the contrapositives of the definitions of $\bigtriangledown$ and $\bigtriangleup$:

(10.5.7) $a \bigtriangleup b \backslash c \;\equiv\; (a \backslash c) \vee (b \backslash c)$, and

(10.5.8) $(c \backslash a \bigtriangledown b) \equiv (c \backslash a) \vee (c \backslash b)$.

Note that in these rules $\triangle$ and $\triangledown$ occur on the left and right of the $\backslash$ symbol, respectively. The rules where $\triangle$ and $\triangledown$ occur on the right and left, respectively, are:

(10.5.9) $(c \setminus a \triangle b) \Rightarrow (c \setminus a) \wedge (c \setminus b)$, and

(10.5.10) $(a \triangledown b \setminus c) \Rightarrow (a \setminus c) \wedge (b \setminus c)$.

(Note the use of implication, rather than equality, in these last two properties. The teacher may want to prove them with the students; if we apply the contrapositive, the proofs are quite simple.)

Finally, using these calculational rules, the proofs of (10.5.4), (10.5.5), and (10.5.6) become simple exercises in calculational logic (the teacher should emphasise syntactic manipulation; given the four previous rules, the students do not need to interpret the expressions at all!). Proposition (10.5.4) can be proved as:

$$
\begin{array}{ll}
 & \{a \triangledown b\,, c\} \in S' \ \wedge\ \{a \triangle b\,, c\} \in S' \\
= & \quad \{ \quad \text{definitions (10.5.3) and } \sharp \quad \} \\
 & (a \triangledown b \setminus c) \wedge (c \setminus a \triangledown b) \wedge (a \triangle b \setminus c) \wedge (c \setminus a \triangle b) \\
\Rightarrow & \quad \{ \quad \text{weakening} \quad \} \\
 & (a \triangledown b \setminus c) \wedge (c \setminus a \triangle b) \\
\Rightarrow & \quad \{ \quad \text{(10.5.10) and (10.5.9)} \quad \} \\
 & (a \setminus c) \wedge (b \setminus c) \wedge (c \setminus a) \wedge (c \setminus b) \\
= & \quad \{ \quad \text{symmetry and definitions (10.5.3) and } \sharp \quad \} \\
 & \{a,c\} \in S \ \wedge\ \{b,c\} \in S \ .
\end{array}
$$

Also, we prove (10.5.5) as follows:

$$
\begin{array}{ll}
 & \{a \triangledown b\,, c\} \in S' \\
= & \quad \{ \quad \text{definitions (10.5.3) and } \sharp \quad \} \\
 & (a \triangledown b \setminus c) \wedge (c \setminus a \triangledown b) \\
\Rightarrow & \quad \{ \quad \text{(10.5.10) and (10.5.8)} \quad \} \\
 & (a \setminus c) \wedge (b \setminus c) \wedge ((c \setminus a) \vee (c \setminus b)) \\
= & \quad \{ \quad \text{distributivity} \quad \} \\
 & ((a \setminus c) \wedge (b \setminus c) \wedge (c \setminus a)) \vee ((a \setminus c) \wedge (b \setminus c) \wedge (c \setminus b))
\end{array}
$$

$$
\begin{array}{ll}
\Rightarrow & \{ \quad \text{weakening} \quad \} \\
& ((a \backslant c) \wedge (c \backslant a)) \vee ((b \backslant c) \wedge (c \backslant b)) \\
= & \{ \quad \text{definitions (10.5.3) and } \# \quad \} \\
& \{a,c\} \in S \ \vee \ \{b,c\} \in S \ .
\end{array}
$$

The proof of (10.5.6) is very similar:

$$
\begin{array}{ll}
& \{a \triangle b \, , c\} \in S' \\
= & \{ \quad \text{definitions (10.5.3) and } \# \quad \} \\
& (a \triangle b \backslant c) \wedge (c \backslant a \triangle b) \\
\Rightarrow & \{ \quad \text{(10.5.9) and (10.5.7)} \quad \} \\
& ((a \backslant c) \vee (b \backslant c)) \wedge (c \backslant a) \wedge (c \backslant b) \\
= & \{ \quad \text{distributivity} \quad \} \\
& ((a \backslant c) \wedge (c \backslant a) \wedge (c \backslant b)) \vee ((b \backslant c) \wedge (c \backslant a) \wedge (c \backslant b)) \\
\Rightarrow & \{ \quad \text{weakening} \quad \} \\
& ((a \backslant c) \wedge (c \backslant a)) \vee ((b \backslant c) \wedge (c \backslant b)) \\
= & \{ \quad \text{definitions (10.5.3) and } \# \quad \} \\
& \{a,c\} \in S \ \vee \ \{b,c\} \in S \ .
\end{array}
$$

We can thus conclude that the size of S is decreasing at each iteration, which means that the algorithm will eventually terminate.

Discuss generalisations and other properties We have just proved that the bag S will eventually become empty, which means that on termination, we have $a \backslash b \vee b \backslash a$ for all numbers a and b that are written in the blackboard B. An immediate conclusion is that the numbers left on the blackboard form a *chain* (alternative names for a chain are *linearly ordered set* and *totally ordered set*).

We have not used anywhere the assumption that the numbers written on the blackboard are positive, which means that the problem statement contains unnecessary detail. Also, even more interesting, in the solution presented above there is nothing special about the operators $\triangle$ and $\triangledown$, and about the relation $\backslant$. In fact, we can generalise the problem to the following:

> Suppose B is a bag of elements of some lattice $(A, \sqsubseteq)$. One can replace any

two distinct values of B by their infimum and supremum instead. Prove that eventually B will stop changing.

The solution to this generalisation is essentially the same, but with the relation $\setminus$ replaced by $\sqsubseteq$ and $\triangle$ and $\triangledown$ representing the supremum and infimum in that order, respectively.

10.5.1 Questions that the teacher should ask

Model the problem

- *Do you understand the problem?*

  The teacher has to be sure that the students understand what is required. Going through one or two examples may be enough for most students.

- *Is this an algorithmic problem?*

  One of the basic and most important skills in algorithmic problem solving is to be able to identify problems of algorithmic nature. We suggest the teacher to ask this question explicitly, so that in subsequent problems students ask the same question to themselves.

- *How can we model the problem?*

  The goal of this question is to start the discussion on the modelling of the problem. This question can be gradually decomposed into other questions, such as *Which variables should we introduce?*, *How can we model the replacement?*, *How can we model the restriction that the numbers have to be different?*, and *How can we model the repetitive nature of the problem?*. All these questions refer to different components of the model.

- *What would happen if the blackboard had the numbers* 1 *and* 2*?*

  The goal of this question is to help the students understand that in certain situations, the collection of numbers will not change. This question prepares the first refinement of the guard.

Refine the model

- *Can we simplify or rewrite the guard* $\{a,b\} \neq \{a \triangle b\, , a \triangledown b\}$*?*

  This question can be used to introduce the discussion on relevant properties for the problem. In particular, the teacher can ask the students if they know any condition on a and b such that $\{a,b\} = \{a \triangle b\, , a \triangledown b\}$.

- *Why is theorem (10.5.1) relevant?* or *What happens if we replace two numbers a and b such that $a \backslash b \vee b \backslash a$?*

  The students have to understand that if we replace two numbers a and b such that $a \backslash b \vee b \backslash a$, then the collection of numbers does not change.

- *How can we use (10.5.1) to change the guard?*

  We assume that the students are familiar with the contrapositive rule. This question can be used to introduce the notion of couple and the new notation that we suggest above.

- *How can we use prove that the algorithm based on the bag S terminates?*

  It is very important that the students understand how they can prove the termination of the algorithm based on the bag S.

The strategy follows from the model

- *[After the strategy is defined] What is the bound function that we are using?*

  The goal of this question is to relate the strategy with the initial discussion on bound functions. The students should understand that, in essence, what they are doing is defining as bound function the function that returns the size of bag S.

- *How can we prove that $|S'| < |S|$?*

  This question can be used to introduce the discussion on how we can prove that the size of S is decreasing.

- *What happens to S when the numbers a and b are replaced by $a \bigtriangledown b$ and $a \bigtriangleup b$?*

  This question is related with the previous one. The first crucial observation is that $a \bigtriangledown b$ and $a \bigtriangleup b$ are not in S'.

- *If $a \bigtriangledown b$ and $a \bigtriangleup b$ are not in S', can we conclude immediately that $|S'| < |S|$?* or *Can S' include new couples?*

  The students have to understand that new couples can form and that the proof reduces to prove that, even with new couples, the size of S' is smaller than the size of S.

- *If new couples are added to S', how can its size be smaller than the size of S? How can we express that formally?*

  This question can be used to introduce the three proof obligations. It is important that the students understand their formalisation. We recommend

the teacher to help them model the proof obligations and to give informal explanations of each.

Calculating the solution

- *Do you know any properties relating the operators $\bigtriangledown$ and $\bigtriangleup$ with the relation $\backslash$?*

  This question can be used to introduce the calculational rules that we use in the final part of the solution..

10.5.2 Questions that the teacher should not ask

Refine the model

- *Can we use the theorem*

$$\{a,b\} = \{a \bigtriangleup b\,,\, a \bigtriangledown b\} \quad \equiv \quad a \backslash b \vee b \backslash a$$

  to simplify the guard?

  We recommend the teacher to work out with the students the theorem, rather than giving it away. The questions in the previous section can help.

10.5.3 Concepts that the teacher should introduce

Bound function

10.6 Extensions and exercises

Exercise 10.6.1 (Euclid's algorithm) Assuming that x and y are both positive integers, does the following loop terminate?

do $\quad y < x \;\rightarrow\; x := x - y$

$\Box \quad x < y \;\rightarrow\; y := y - x$

od

If so, give a bound function on x and y that decreases at each iteration.

□

Exercise 10.6.2 (Binary Strings) Suppose that we have a finite bit string (i.e. a string of zeroes and ones) that is repeatedly transformed by replacing

a pattern 00 by 01 , or
a pattern 11 by 10 , wherever in the string and as long as such transformations are possible.

(This problem was taken from Netty van Gasteren's thesis [vG90, p. 14].)

□

10.7 Solutions to extensions and exercises

10.6.1 At each step, one of the variables is being decreased, which means that the loop terminates and the function $x+y$ can be used as a bound function. To formally prove that the loop terminates, we can assume that the bound function has a certain value, say Z, and verify that that value decreases after each iteration. More formally, we verify the following:

$$\begin{array}{ll} & \{\ x+y = Z\ \} \\ & y < x \ \rightarrow\ x\ :=\ x-y \\ \Box\ & x < y \ \rightarrow\ y\ :=\ y-x \\ & \{\ x+y < Z\ \} \end{array}$$

We have two calculations, one for each assignment. If the first assignment is executed, the formal requirement that we have to prove is:

$$(x-y)+y < Z \ \Leftarrow\ x+y = Z \quad ,$$

which is the same as

$$x < Z \ \Leftarrow\ x+y = Z \quad ,$$

We can easily prove it as follows:

$$\begin{array}{lll} & x+y = Z & \\ = & \{ & \text{monotonicity, i.e., subtract } y \text{ from both sides;} \\ & & \text{we can safely do this, because } y \text{ is positive} \quad \} \end{array}$$

$$
\begin{array}{ll}
 & x = Z - y \\
\Rightarrow & \{ \quad y \text{ is a positive number, so } Z - y < Z \quad \} \\
 & x < Z \ .
\end{array}
$$

If the second assignment is executed, the formal requirement and the proof are symmetric on y. So, we conclude that $x+y$ is decreasing. Furthermore, since x and y are both positive, $x+y$ is also positive (i.e., it is bounded below). Therefore, the loop has to terminate.
□

10.6.2 The solution we show here is Netty van Gasteren's solution. Since the pair of transformations is invariant under an interchange of 0 and 1, only equality and difference of bits matter. Exploiting this observation, we record the succession of neighbour equalities and differences in the bit string as a string of y's and x's, with

y standing for a pair of equal neighbour bits, and
x standing for a pair of different neighbour bits

(which given the first bit precisely determines the bit string).

In this terminology, a transformation changes a y in the "code string" into an x, while leaving all elements to the left of that y unchanged. Thus the code string decreases lexically at each transformation. Since it furthermore is lexically bounded from below — by the string of appropriate length consisting of x's only — the game terminates.

(The shape of the bit string upon termination follows from the observation that the leftmost bit of the bit string does not change in the game and that upon termination the code string consists of x's only.)
□

10.8 Further reading

We have found this problem in [Zei06, p. 77] as an example of the extreme principle. In there, Zeitz starts with some examples and with an informal discussion that, according to him, "leads easily" to the following conjecture:

> Eventually, the sequence will form a chain where each element will divide the next (when arranged in order). Moreover, the least element and the greatest element of this chain are respectively the greatest common divisor and least common multiple of all the original numbers.

He then presents an informal argument, followed by a formal solution that verifies the conjecture by induction. Our main criticism of his solution is that we are not convinced that working out some examples "leads easily" to the conjecture. Once it is stated, it is easy to see it; but how can we teach students to guess this sort of conjecture? In our solution, once we formalise the algorithm, we clearly see that the loop body will run while S is non-empty; this immediately suggests that we need to prove that the size of S decreases at each iteration. The fact that a chain is formed follows immediately from our definition of S. So we think that the amount of guessing in our solution is much more reasonable and teachable.

Also, we think that our calculational solution helps to generalise the problem. It is very easy to see that all the properties used apply to any lattice.

SCENARIO 11

Constructing Euclid's Algorithm

11.1 Brief description and goals

The goal of this scenario is to derive Euclid's algorithm to compute the greatest common divisor of two positive natural numbers. We also show how to use the algorithm to verify theorems related with the greatest common divisor. The scenario can be used to introduce the construction of programs from their formal specifications and the use of invariance to prove theorems.

11.2 Problem

The greatest common divisor (gcd) of two numbers is the largest natural number that divides both numbers. For example, the greatest common divisor of 8 and 12, denoted in this scenario as $8 \bigtriangledown 12$, is 4. (We use the symbol $\bigtriangledown$ to denote the gcd operator and we call it "nabla".)

The gcd is a fundamental concept in number theory, so it is important to know some of its properties. For example, a property that is commonly used to simplify calculations is that multiplication by a natural number distributes over $\bigtriangledown$:

(11.2.1) $[\ (c \times m) \bigtriangledown (c \times n) = c \times (m \bigtriangledown n)\ ]$.

The square so-called "everywhere" brackets are used to indicate that a Boolean statement is "everywhere" true. That is, the statement has the value true for all instantiations of its free variables. Such statements are often called "facts", or "laws", or "theorems". When using the everywhere brackets, the domain of the free variables has to be made clear. This is particularly important here because sometimes the domain of a variable is the integers and sometimes it is the natural numbers. Usually, we rely on a conven-

tion for naming the variables, but sometimes we preface a law with a reminder of the domain.

Note that confusion and ambiguity occur when we define the gcd as the *largest* natural number that divides both numbers, because the natural numbers can be ordered in more than one way: they can be ordered by the usual size ordering (denoted by the symbol $\leq$), but they can also be ordered by the division relation (denoted by the symbol $\backslash$). To avoid ambiguity, we define the gcd of two numbers as the largest number that divides both numbers *with respect to the division ordering*. More formally, given two numbers m and n, $m \bigtriangledown n$ is defined as

(11.2.2) $\langle \forall k :: \; k \backslash m \; \wedge \; k \backslash n \;\; \equiv \;\; k \; \backslash \; m \bigtriangledown n \rangle$.

The goal of this scenario is to construct an algorithm that computes the gcd of two positive natural numbers. That is, we want to construct an algorithm that solves the following equation:

(11.2.3) $x ::$ $\qquad \langle \forall k \;\; :: \;\; k \backslash m \; \wedge \; k \backslash n \;\; \equiv \;\; k \backslash x \rangle$.

Also, can we use the derived algorithm to prove property (11.2.1)?

11.3 Prerequisites

Knowledge of division properties, invariants, guarded-command language, and indirect equality may be useful.

11.4 Resolution and notes

The goal is to solve the two following problems:

1. construct an algorithm that solves equation (11.2.3), where m and n are positive naturals;
2. use the derived algorithm to prove property (11.2.1).

We solve them in the two following sections.

11.4.1 Constructing the algorithm

Equation (11.2.3) does not directly suggest any algorithm, but the germ of an algorithm is suggested by observing that it is equivalent to

(11.4.1) $x,y::\quad x=y \;\wedge\; \langle\forall k::\; k\backslash m \wedge k\backslash n \;\equiv\; k\backslash x \wedge k\backslash y\rangle$.

This new shape strongly suggests an algorithm that, initially, establishes the truth of

$$\langle\forall k::\; k\backslash m \wedge k\backslash n \;\equiv\; k\backslash x \wedge k\backslash y\rangle$$

— which is trivially achieved by the assignment $x,y := m,n$ — and then, reduces x and y in such a way that the property is kept invariant whilst making progress to a state satisfying $x=y$. When such a state is reached, we have found a solution to the equation (11.4.1), and the value of x (or y since they are equal) is a solution of (11.2.3). Thus, the structure of the algorithm we are trying to develop is as follows[1]:

$\{\; 0<m \;\wedge\; 0<n \;\}$

$x\,,y \;:=\; m\,,n \;;$

$\{$ **Invariant**: $\langle\forall k::\; k\backslash m \wedge k\backslash n \;\equiv\; k\backslash x \wedge k\backslash y\rangle \;\}$

do $x\neq y \;\rightarrow\; x\,,y \;:=\; A\,,B$

od

$\{\; x=y \;\wedge\; \langle\forall k::\; k\backslash m \wedge k\backslash n \;\equiv\; k\backslash x \wedge k\backslash y\rangle \;\}$

Now we only have to define A and B in such a way that the assignment in the loop body leads to a state where $x=y$ is satisfied while maintaining the invariant. Exploiting the transitivity of equality, the invariant is maintained by choosing A and B so that

(11.4.2) $\langle\forall k::\; k\backslash x \wedge k\backslash y \;\equiv\; k\backslash A \wedge k\backslash B\rangle$.

To ensure that we are making progress towards the termination condition, we have to define a *bound function*, which is a natural-valued function of the variables x and y that measures the size of the problem to be solved. A guarantee that the value of such a bound function is always decreased at each iteration is a guarantee that the number of times the loop body is executed is at most the initial value of the bound function. The definition of the bound function depends on the assignments we choose for A and B.

[1] We use the Guarded Command Language (GCL), a very simple programming language with just four programming constructs — assignment, sequential composition, conditionals, and loops. The GCL was introduced by Dijkstra [Dij75]. The statement do S od is a loop that executes S repeatedly while at least one of S's guards is true. Expressions in curly brackets are assertions.

At this point, we need to exploit properties specific to division. Inspecting the shape of (11.4.2), we see that it is similar to the shape of the following property:

$$[\ k\backslash x \wedge k\backslash y \ \equiv\ k\backslash(x+a\times y) \wedge k\backslash y\ ]\ .$$

A corollary of this property is:

(11.4.3) $[\ k\backslash x \wedge k\backslash y \ \equiv\ k\backslash(x-y) \wedge k\backslash y\ ]\ .$

The relevance of this corollary is that our invariant is preserved by the assignment $x := x-y$ (leaving the value of y unchanged). (Compare (11.4.3) with (11.4.2).) Note that this also reduces the value of x when y is positive. This suggests that we strengthen the invariant by requiring that x and y remain positive; the assignment $x := x-y$ is executed when x is greater than y and, symmetrically, the assignment $y := y-x$ is executed when y is greater than x. As bound function we can take $x+y$ (see exercise 11.6.2). The algorithm becomes

$\{\ 0<m \ \wedge\ 0<n\ \}$

$x,y := m,n\ ;$

$\{$ **Invariant**: $0<x \ \wedge\ 0<y \ \wedge\ \langle\forall k::\ k\backslash m \wedge k\backslash n \ \equiv\ k\backslash x \wedge k\backslash y\rangle$

Bound function: $x+y\ \}$

do $x\neq y\ \rightarrow$

if $\ y<x \rightarrow x := x-y$

$\square\ \ x<y \rightarrow y := y-x$

fi

od

$\{\ 0<x \ \wedge\ 0<y \ \wedge\ x=y \ \wedge\ \langle\forall k::\ k\backslash m \wedge k\backslash n \ \equiv\ k\backslash x \wedge k\backslash y\rangle\ \}$

(Exercise 11.6.1 is about formally verifying the correctness of the algorithm.) Finally, since

$$(x<y \vee y<x) \ \equiv\ x\neq y\ ,$$

we can safely remove the outer guard and simplify the algorithm, as shown below.

$\{\ 0<m \ \wedge\ 0<n\ \}$

$x,y := m,n\ ;$

$\{$ **Invariant**: $0<x \;\wedge\; 0<y \;\wedge\; \langle\forall k::\; k\backslash m \wedge k\backslash n \;\equiv\; k\backslash x \wedge k\backslash y\rangle$

Bound function: $x+y$ $\}$

do $y<x \;\rightarrow\; x := x-y$

$\Box \quad x<y \;\rightarrow\; y := y-x$

od

$\{\; 0<x \;\wedge\; 0<y \;\wedge\; x=y \;\wedge\; \langle\forall k::\; k\backslash m \wedge k\backslash n \;\equiv\; k\backslash x \wedge k\backslash y\rangle \;\}$

The algorithm that we have constructed is Euclid's algorithm for computing the greatest common divisor of two positive natural numbers, the oldest nontrivial algorithm that has survived to the present day! (Please note that our formulation of the algorithm differs from most versions found in number-theory books. While they use the property $[\; m\triangledown n \;=\; n\triangledown(m \bmod n) \;]$, we use (11.4.3), i.e., $[\; m\triangledown n \;=\; (m-n)\triangledown n \;]$. For an encyclopedic account of Euclid's algorithm, we recommend [Knu97, p. 334].)

11.4.2 Proving the distributivity property

makeThe invariant that we use in the previous section rests on the validity of the theorem

$$[\; k\backslash m \wedge k\backslash n \;\equiv\; k\backslash(m-n) \wedge k\backslash n \;] \;.$$

But, as Van Gasteren observed in [vG90, Chapter 11], we can use the more general and equally valid theorem

$$[\; k\setminus(c\times m) \wedge k\setminus(c\times n) \;\equiv\; k\setminus(c\times(m-n)) \wedge k\setminus(c\times n) \;]$$

to conclude that the following property is an invariant of Euclid's algorithm:

$$\langle\forall k,c::\; k\setminus(c\times m) \wedge k\setminus(c\times n) \;\equiv\; k\setminus(c\times x) \wedge k\setminus(c\times y)\rangle \;.$$

In particular, the property is true on termination of the algorithm, at which point x and y both equal $m\triangledown n$. That is, for all m and n, such that $0<m$ and $0<n$,

(11.4.4) $[\; k\setminus(c\times m) \wedge k\setminus(c\times n) \;\equiv\; k\setminus(c\times(m\triangledown n)) \;] \;.$

In addition, theorem (11.4.4) holds when $m<0$, since

$$[\; (-m)\triangledown n = m\triangledown n \;] \;\wedge\; [\; k\setminus(c\times(-m)) \;\equiv\; k\setminus(c\times m) \;] \;,$$

and it holds when m equals 0, since $[\; k\backslash 0 \;]$. Hence, using the symmetry between m and n we conclude that (11.4.4) is indeed valid for all integers m and n. (In Van Gasteren's presentation, this theorem only holds for all $(m,n)\neq(0,0)$.)

Theorem (11.4.4) can be used to prove property (11.2.1) by indirect equality. That is, we prove:

$$[\ k \setminus (c \times (m \nabla n)) \quad \equiv \quad k \setminus ((c \times m) \nabla (c \times n))\ ] \ .$$

The proof is simple:

$$\begin{array}{ll} & k \setminus (c \times (m \nabla n)) \\ = & \quad \{ \quad (11.4.4) \quad \} \\ & k \setminus (c \times m) \ \wedge \ k \setminus (c \times n) \\ = & \quad \{ \quad (11.2.2) \quad \} \\ & k \setminus ((c \times m) \nabla (c \times n)) \ . \end{array}$$

Property (11.2.1) is an important property that can be used to simplify arguments where both multiplication and the greatest common divisor are involved. An example is Van Gasteren's proof of the theorem

(11.4.5) $[\ (m \times p) \nabla n = m \nabla n \ \Leftarrow \ p \nabla n = 1\ ]$,

which is as follows:

$$\begin{array}{ll} & m \nabla n \\ = & \quad \{ \quad p \nabla n = 1 \text{ and 1 is the unit of multiplication} \quad \} \\ & (m \times (p \nabla n)) \nabla n \\ = & \quad \{ \quad (11.2.1) \quad \} \\ & (m \times p) \ \nabla \ (m \times n) \ \nabla \ n \\ = & \quad \{ \quad (m \times n) \nabla n = n \quad \} \\ & (m \times p) \nabla n \ . \end{array}$$

11.5 Notes for the teacher

Manipulating the specification It is important that the students understand that the goal is to construct an algorithm satisfying

$$\{\ 0 < m \ \wedge \ 0 < n\ \}$$

Compute x

$$\{\ \langle \forall k \ :: \ k \setminus m \ \wedge \ k \setminus n \ \equiv \ k \setminus x \rangle\ \} \ .$$

Equation (11.2.3) does not directly suggest any algorithm nor an invariant. We suggest the teacher to ask the students how they can make (11.2.3) more symmetric; introducing symmetry can make it easier to initialise and may suggest an invariant. The goal is to introduce the following equivalent equation:

(11.5.1) $x,y::\quad x=y \;\wedge\; \langle\forall k::\; k\backslash m \wedge k\backslash n \;\equiv\; k\backslash x \wedge k\backslash y\rangle$.

This new equation strongly suggests an algorithm that, initially, establishes the truth of

$$\langle\forall k::\; k\backslash m \wedge k\backslash n \;\equiv\; k\backslash x \wedge k\backslash y\rangle$$

— which is trivially achieved by the assignment $x,y := m,n$ — and then, reduces x and y in such a way that the property is kept invariant whilst making progress to a state satisfying $x=y$. When such a state is reached, we have found a solution to the equation (11.5.1), and the value of x (or y since they are equal) is a solution of (11.2.3). Thus, the structure of the algorithm we are trying to develop is as follows:

$\{\; 0<m \;\wedge\; 0<n \;\}$

$x\,,y \;:=\; m\,,n \;;$

$\{$ **Invariant**: $\langle\forall k::\; k\backslash m \wedge k\backslash n \;\equiv\; k\backslash x \wedge k\backslash y\rangle \;\}$

do $x\neq y \;\rightarrow\; x\,,y \;:=\; A\,,B$

od

$\{\; x=y \;\wedge\; \langle\forall k::\; k\backslash m \wedge k\backslash n \;\equiv\; k\backslash x \wedge k\backslash y\rangle \;\}$

This technique is very common. Suppose that we have a postcondition of the form $P\wedge Q$, and we know how to initialise one of the conjuncts, say P. Then, a possible shape for an algorithm that establishes the postcondition using a loop has P as an invariant and $\neg Q$ as the guard of the loop. Property P is true on termination (because it is an invariant) and the loop terminates when Q is true; as a result, the algorithm establishes $P\wedge Q$. It is important that the students understand this technique.

Also, the teacher may have to explain the notations used to express the algorithm. We use the Guarded Command Language (GCL), a very simple programming language with just four programming constructs — assignment, sequential composition, conditionals, and loops. The GCL was introduced by Dijkstra [Dij75]. The statement do S od is a loop that executes S repeatedly while at least one of S's guards is true. Expressions in curly brackets are assertions.

Calculating the assignments Now we only have to define A and B in such a way that the assignment in the loop body leads to a state where $x=y$ is satisfied while maintain-

ing the invariant. Exploiting the transitivity of equality, the invariant is maintained by choosing A and B so that

(11.5.2) $\langle \forall k ::\ k \backslash x \wedge k \backslash y \ \equiv \ k \backslash A \wedge k \backslash B \rangle$.

At this point, we need to exploit properties specific to division. We suggest the teacher to ask if the students know any properties similar to (11.5.2). If they do not know any properties, the teacher may introduce the following theorem:

$$[\ k \backslash x \wedge k \backslash y \ \equiv \ k \backslash (x + a \times y) \wedge k \backslash y\ ] \ .$$

(The teacher may have to explain that the square brackets mean that the theorem is true for all possible values of k, x, y, and a.) Now, we recommend the teacher asks why this theorem is relevant. The students should understand that if A is chosen to be $x + a \times y$ and B to be y, the invariant is preserved. Moreover, a corollary of this property is:

(11.5.3) $[\ k \backslash x \wedge k \backslash y \ \equiv \ k \backslash (x - y) \wedge k \backslash y\ ]$.

The relevance of this corollary is that our invariant is preserved by the assignment $x := x - y$ (leaving the value of y unchanged). It is important that the students understand this. Note that this also reduces the value of x when y is positive. This suggests that we strengthen the invariant by requiring that x and y remain positive; the assignment $x := x - y$ is executed when x is greater than y and, symmetrically, the assignment $y := y - x$ is executed when y is greater than x. The algorithm becomes

$\{\ 0 < m \ \wedge \ 0 < n\ \}$

$x, y := m, n\ ;$

$\{$ **Invariant**: $0 < x \ \wedge \ 0 < y \ \wedge \ \langle \forall k ::\ k \backslash m \wedge k \backslash n \ \equiv \ k \backslash x \wedge k \backslash y \rangle\ \}$

do $x \neq y \ \rightarrow$

if $\ y < x \ \rightarrow \ x := x - y$

$\square \ \ x < y \ \rightarrow \ y := y - x$

fi

od

$\{\ 0 < x \ \wedge \ 0 < y \ \wedge \ x = y \ \wedge \ \langle \forall k ::\ k \backslash m \wedge k \backslash n \ \equiv \ k \backslash x \wedge k \backslash y \rangle\ \}$

The teacher may want to prove the correctness of the assignments more formally; alternatively, it can be given as an exercise (see exercise 11.6.1).

Simplifying the guards Finally, we recommend the teacher to ask if the algorithm can be simplified. The goal is to observe that, since

$$(x<y \;\vee\; y<x) \;\equiv\; x\neq y \quad ,$$

we can safely remove the outer guard and simplify the algorithm, as shown below.

$\{\; 0<m \;\wedge\; 0<n \;\}$

$x,y \;:=\; m,n \;;$

$\{$ **Invariant**: $\;0{<}x \;\wedge\; 0{<}y \;\wedge\; \langle\forall k{::}\; k\backslash m \wedge k\backslash n \;\equiv\; k\backslash x \wedge k\backslash y\rangle \;\}$

do $\;y<x \;\rightarrow\; x \;:=\; x{-}y$

$\Box \quad x<y \;\rightarrow\; y \;:=\; y{-}x$

od

$\{\; 0{<}x \;\wedge\; 0{<}y \;\wedge\; x=y \;\wedge\; \langle\forall k{::}\; k\backslash m \wedge k\backslash n \;\equiv\; k\backslash x \wedge k\backslash y\rangle \;\}$

This algorithm is the so-called Euclid's algorithm for computing the greatest common divisor of two positive natural numbers, the oldest nontrivial algorithm that has survived to the present day! (The teacher should say that our formulation of the algorithm differs from most versions found in number-theory books. While they use the property $[\; m\triangledown n \;=\; n\triangledown(m \bmod n) \;]$, we use (11.5.3), i.e., $[\; m\triangledown n \;=\; (m{-}n)\triangledown n \;]$.)

Proving termination We recommend the teacher to ask the students if the algorithm above terminates. The students should understand that the postcondition is only established if termination is proved. To ensure that we are making progress towards the termination condition, we have to define a *bound function*, which is a natural-valued function of the variables x and y that measures the size of the problem to be solved. A guarantee that the value of such a bound function is always decreased at each iteration is a guarantee that the number of times the loop body is executed is at most the initial value of the bound function. The definition of the bound function depends on the assignments we choose for A and B.

We recommend the teacher asks if the students can think of any function on variables x and y that decreases at each iteration and is bounded below. A possibility is the function $x{+}y$; the teacher may want to prove that it can be chosen as a bound function (see exercise 11.6.2).

We can also annotate the algorithm with the bound function:

$\{\; 0<m \;\wedge\; 0<n \;\}$

$x, y := m, n$;

{ **Invariant**: $0<x \ \wedge \ 0<y \ \wedge \ \langle \forall k::\ k\backslash m \wedge k\backslash n \ \equiv \ k\backslash x \wedge k\backslash y \rangle$

Bound function: $x+y$ }

do $y < x \rightarrow x := x-y$

$\Box \quad x < y \rightarrow y := y-x$

od

{ $0<x \ \wedge \ 0<y \ \wedge \ x=y \ \wedge \ \langle \forall k::\ k\backslash m \wedge k\backslash n \ \equiv \ k\backslash x \wedge k\backslash y \rangle$ }

Generalising the invariant The second part of the problem is about proving property (11.2.1), using the derived algorithm. Before we tackle the proof, we suggest the teacher to introduce a generalisation of the invariant that we have used. The invariant that we use in the previous section rests on the validity of the theorem

$$[\ k\backslash m \wedge k\backslash n \ \equiv \ k\backslash(m-n) \wedge k\backslash n\ ] \ .$$

But we can use the more general and equally valid theorem

$$[\ k \setminus (c\times m) \wedge k \setminus (c\times n) \ \equiv \ k \setminus (c \times (m-n)) \wedge k \setminus (c\times n)\ ]$$

to conclude that the following property is an invariant of Euclid's algorithm:

$$\langle \forall k, c::\ k \setminus (c\times m) \wedge k \setminus (c\times n) \ \equiv \ k \setminus (c\times x) \wedge k \setminus (c\times y) \rangle \ .$$

(The teacher may want to prove that it is indeed an invariant.) In particular, the property is true on termination of the algorithm, at which point x and y both equal $m \triangledown n$. That is, for all m and n, such that $0 < m$ and $0 < n$,

(11.5.4) $$[\ k \setminus (c\times m) \wedge k \setminus (c\times n) \ \equiv \ k \setminus (c \times (m \triangledown n))\ ] \ .$$

In addition, theorem (11.5.4) holds when $m < 0$, since

$$[\ (-m) \triangledown n = m \triangledown n\ ] \ \wedge \ [\ k \setminus (c\times(-m)) \ \equiv \ k \setminus (c\times m)\ ] \ ,$$

and it holds when m equals 0, since $[\ k\backslash 0\ ]$. Hence, using the symmetry between m and n we conclude that (11.5.4) is indeed valid for all integers m and n.

The teacher may omit the generalisation to the integer domain and restrict all the proofs to natural numbers.

Proving the distributivity property Theorem (11.5.4) can be used to prove property (11.2.1) by indirect equality. That is, we prove:

(11.5.5) $[\ k \setminus (c \times (m \nabla n)) \quad \equiv \quad k \setminus ((c \times m) \nabla (c \times n))\ ]$.

We suggest the teacher to explain what is indirect equality and to show why (11.5.5) establishes (11.2.1). If property (11.5.5) is true, it is true for all possible values of k. In particular, it is valid for $k = c \times (m \nabla n)$; replacing k by $c \times (m \nabla n)$, the left-hand side simplifies to true and we conclude that $(c \times (m \nabla n)) \setminus ((c \times m) \nabla (c \times n))$. But it is also valid for $k = (c \times m) \nabla (c \times n)$, so if we replace k, the right-hand side simplifies to true and we conclude $(c \times m) \nabla (c \times n) \setminus (c \times (m \nabla n))$. Therefore, by anti-symmetry of the division relation (we assume that c is a natural), we have $(c \times m) \nabla (c \times n) = c \times (m \nabla n)$.

The proof of (11.5.5) is simple:

$$
\begin{array}{ll}
 & k \setminus (c \times (m \nabla n)) \\
= & \quad \{ \quad (11.5.4) \quad \} \\
 & k \setminus (c \times m) \ \wedge \ k \setminus (c \times n) \\
= & \quad \{ \quad (11.2.2) \quad \} \\
 & k \setminus ((c \times m) \nabla (c \times n)) \ .
\end{array}
$$

Property (11.2.1) is an important property that can be used to simplify arguments where both multiplication and the greatest common divisor are involved. An example is Van Gasteren's proof of the theorem

$$[\ (m \times p) \nabla n = m \nabla n \ \Leftarrow \ p \nabla n = 1\ ] \ ,$$

which is as follows:

$$
\begin{array}{ll}
 & m \nabla n \\
= & \quad \{ \quad p \nabla n = 1 \text{ and 1 is the unit of multiplication} \quad \} \\
 & (m \times (p \nabla n)) \nabla n \\
= & \quad \{ \quad (11.2.1) \quad \} \\
 & (m \times p) \nabla (m \times n) \nabla n \\
= & \quad \{ \quad (m \times n) \nabla n = n \quad \} \\
 & (m \times p) \nabla n \ .
\end{array}
$$

We suggest the teacher to prove this property with the students.

Discuss generalisations and exercises Euclid's algorithm can be used to prove other theorems on the greatest common divisor. For two non-trivial examples, see exercises 11.6.3 and 11.6.4. We believe these exercises can be set as project assignments.

11.5.1 Questions that the teacher should ask

Manipulating the specification

- *Do you understand the problem?*

  The teacher has to be sure that the students understand what is required. We recommend the teacher to show the formal specification to the students.

- *Sometimes, it helps to make expressions more symmetric. How can we make (11.2.3) more symmetric?*

  The goal of this question is to introduce (11.5.1). It is important to let the students think about the question.

- *Does (11.5.1) suggest any algorithm?*

  The goal of this question is to introduce the discussion on using one conjunct as the invariant and the negation of the other as the guard of the loop. It is important that the students understand the technique.

Calculating the assignments

- *Do you know any property similar to (11.5.2)?*

  The goal of this question is to start the discussion on relevant properties of division. The goal is to introduce (11.5.3). It is important that the students understand the relevance of (11.5.3).

- *Is (11.5.3) relevant to determine the assignments?*

  It is important that the students understand the relevance of (11.5.3).

Simplifying the guards

- *Can we simplify the algorithm?*

  The algorithm can be simplified by removing the if statement. The teacher may have to explain informally why both versions are semantically the same.

Proving termination

- *Does the algorithm always terminate?*

The goal is to make the students realise that the postcondition is only established if termination is proved. We suggest the teacher asks the students how would they prove that it always terminates. After discussing the concept of bound function, the teacher may ask if they can think of any function on variables x and y that decreases at each iteration and is bounded below.

Proving the distributivity property

- *Can you see why (11.5.5) establishes the distributivity property?*

  The goal is to start the discussion on indirect equality. We suggest the teacher to show the instantiations that establish the distributivity property and to highlight the relevant properties (reflexivity and anti-symmetry).

11.5.2 Questions that the teacher should not ask

Proving termination

- *Can you see that $x := x-y$ (or $y := y-x$) is a valid assignment?*

  We think it is better to let the students choose the assignments based on the properties of division discussed before.

Proving termination

- *Can $x+y$ be used as a bound function?*

  This question should not be asked because it gives away a possible bound function. It is better to let the students suggest functions and verify if they can be used.

11.5.3 Concepts that the teacher should introduce

Bound function

Invariant

Indirect equality

11.6 Extensions and exercises

Exercise 11.6.1 (Correctness) Prove the correctness of the assignments in the loop of Euclid's algorithm.

□

Exercise 11.6.2 (Termination) We have stated, but not proved, that $x+y$ can be used as a bound function. Show that $x+y$ can indeed be used.

□

Exercise 11.6.3 (Geometric property) Using Euclid's algorithm, prove that in a Cartesian coordinate system, $m \bigtriangledown n$ can be interpreted as the number of points with integral coordinates on the straight line joining the points $(0,0)$ and (m,n), excluding $(0,0)$. More formally, prove the following (with dummies s and t ranging over integers):

$$[\ \langle \Sigma s,t\ :\ m\times t = n\times s\ \wedge\ s\leq m\ \wedge\ t\leq n\ \wedge\ (0<s\vee 0<t)\ :\ 1\rangle\ =\ m\bigtriangledown n\ ]\ .$$

□

Exercise 11.6.4 (Distributivity) In addition to multiplication by a natural number, there are other functions that distribute over $\bigtriangledown$. Based on Euclid's algorithm, investigate and determine reasonable sufficient conditions for a natural-valued function f to distribute over $\bigtriangledown$, i.e., for the following property to hold:

$$[\ f.(m\bigtriangledown n)\ =\ f.m\ \bigtriangledown\ f.n\ ]\ .$$

□

11.7 Solutions to extensions and exercises

11.6.1 The goal is to prove that

$$\langle \forall k::\ k\backslash m\ \wedge\ k\backslash n\ \equiv\ k\backslash x\ \wedge\ k\backslash y\rangle$$

is an invariant of the assignments in the loop body. It surely is an invariant with respect to the assignment $x := x-y$, since we have:

$$\begin{array}{ll} & \langle \forall k::\ k\backslash m\ \wedge\ k\backslash n\ \equiv\ k\backslash (x-y)\ \wedge\ k\backslash y\rangle \\ = & \quad \{\quad [\ k\backslash (x-y)\ \wedge\ k\backslash y\ \equiv\ k\backslash x\ \wedge\ k\backslash y\ ]\ \} \\ & \langle \forall k::\ k\backslash m\ \wedge\ k\backslash n\ \equiv\ k\backslash x\ \wedge\ k\backslash y\rangle\ . \end{array}$$

The proof for the second assignment is similar and left to the reader.
□

11.6.2 To prove that $x+y$ can be used as a bound function, we have to prove that it decreases at each iteration and that it is bounded below. Because x and y are positive numbers, $x+y$ is also positive; this means it is bounded below. As a result, the formal requirements for it to be a bound function are:

$$(x+y)[x := x-y] < C \Leftarrow x+y = C \quad \text{, and}$$

$$(x+y)[y := y-x] < C \Leftarrow x+y = C \quad .$$

The first can be proved as:

$$\begin{array}{ll} & (x+y)[y := y-x] < C \\ = & \quad \{ \quad \text{substitution} \quad \} \\ & x+(y-x) < C \\ = & \quad \{ \quad \text{associativity} \quad \} \\ & (x+y)-x < C \\ = & \quad \{ \quad x+y = C \quad \} \\ & C-x < C \\ = & \quad \{ \quad 0 < x \quad \} \\ & \mathsf{true} \ . \end{array}$$

The second is similar and left to the reader.
□

11.6.3 We want to prove the following property:

(11.7.1)

$$[\ \langle \Sigma s,t \ : \ m\times t = n\times s \ \wedge \ s \le m \ \wedge \ t \le n \ \wedge \ (0<s \vee 0<t) \ : \ 1\rangle \ = \ m \triangledown n \ ] \ .$$

We begin by observing that (11.7.1) holds when $m = 0$ or when $n = 0$ (we leave the proof to the reader). When $0 < m$ and $0 < n$, we can simplify the range of (11.7.1). First, we observe that

$$(0 < s \le m \ \equiv \ 0 < t \le n) \Leftarrow m\times t = n\times s \quad ,$$

since

$$
\begin{array}{ll}
 & 0 < t \leq n \\
= & \quad \{ \quad 0 < m \quad \} \\
 & 0 < m \times t \leq m \times n \\
= & \quad \{ \quad m \times t = n \times s \quad \} \\
 & 0 < n \times s \leq m \times n \\
= & \quad \{ \quad 0 < n, \text{cancellation} \quad \} \\
 & 0 < s \leq m \ .
\end{array}
$$

As a result, (11.7.1) can be written as

(11.7.2) $[\ \langle \Sigma s, t \ : \ m \times t = n \times s \ \wedge \ 0 < t \leq n \ : \ 1 \rangle = m \bigtriangledown n \ ]$.

In order to use Euclid's algorithm, we need to find an invariant that allows us to conclude (11.7.2). If we use as invariant

(11.7.3) $\langle \Sigma s, t \ : \ x \times t = y \times s \ \wedge \ 0 < t \leq y \ : \ 1 \rangle = x \bigtriangledown y$,

its initial value is the property that we want to prove:

$$\langle \Sigma s, t \ : \ m \times t = n \times s \ \wedge \ 0 < t \leq n \ : \ 1 \rangle = m \bigtriangledown n \ .$$

Its value upon termination is

$$\langle \Sigma s, t \ : \ (m \bigtriangledown n) \times t = (m \bigtriangledown n) \times s \ \wedge \ 0 < t \leq m \bigtriangledown n \ : \ 1 \rangle = (m \bigtriangledown n) \bigtriangledown (m \bigtriangledown n) \ ,$$

which is equivalent (by cancellation of multiplication and idempotence of $\bigtriangledown$) to

$$\langle \Sigma s, t \ : \ t = s \ \wedge \ 0 < t \leq m \bigtriangledown n \ : \ 1 \rangle = m \bigtriangledown n \ .$$

It is easy to see that the invariant reduces to true on termination (because the sum on the left equals $m \bigtriangledown n$), making its initial value also true.

It is also easy to see that the right-hand side of the invariant is unnecessary as it is the same initially and on termination. Therefore, we can simplify (11.7.3) and use as invariant

(11.7.4) $\langle \Sigma s, t \ : \ x \times t = y \times s \ \wedge \ 0 < t \leq y \ : \ 1 \rangle$.

Its value on termination is

$$\langle \Sigma s, t \ : \ (m \bigtriangledown n) \times t = (m \bigtriangledown n) \times s \ \wedge \ 0 < t \leq m \bigtriangledown n \ : \ 1 \rangle \ ,$$

which is equivalent to

$$\langle \Sigma s,t \ : \ t = s \ \wedge \ 0 < t \leq m \triangledown n \ : \ 1 \rangle \quad .$$

As said above, this sum equals $m \triangledown n$.

Now, since the invariant (11.7.4) equals the left-hand side of (11.7.2) for the initial values of x and y, we only have to check if it remains constant after each iteration. This means that we have to prove (for $y < x \wedge 0 < y$):

$$\begin{array}{ll} & \langle \Sigma s,t \ : \ x \times t = y \times s \ \wedge \ 0 < t \leq y \ : \ 1 \rangle \\ = & \langle \Sigma s,t \ : \ (x-y) \times t = y \times s \ \wedge \ 0 < t \leq y \ : \ 1 \rangle \ , \end{array}$$

which can be rewritten, for positive x and y, as:

$$\begin{array}{ll} & \langle \Sigma s,t \ : \ (x+y) \times t = y \times s \ \wedge \ 0 < t \leq y \ : \ 1 \rangle \\ = & \langle \Sigma s,t \ : \ x \times t = y \times s \ \wedge \ 0 < t \leq y \ : \ 1 \rangle \ . \end{array}$$

The proof is as follows:

$$\begin{array}{ll} & \langle \Sigma s,t \ : \ (x+y) \times t = y \times s \ \wedge \ 0 < t \leq y \ : \ 1 \rangle \\ = & \quad \{ \quad \text{distributivity and cancellation} \quad \} \\ & \langle \Sigma s,t \ : \ x \times t = y \times (s-t) \ \wedge \ 0 < t \leq y \ : \ 1 \rangle \\ = & \quad \{ \quad \text{range translation: } s := s+t \quad \} \\ & \langle \Sigma s,t \ : \ x \times t = y \times s \ \wedge \ 0 < t \leq y \ : \ 1 \rangle \ . \end{array}$$

Note that the simplification done in (11.7.2) allows us to apply the range translation rule in the last step without having to relate the range of variable s with the possible values for variable t.

□

11.6.4 The goal is to determine reasonable sufficient conditions for a natural-valued function f to distribute over $\triangledown$, i.e., for the following property to hold:

(11.7.5) $[\ f.(m \triangledown n) = f.m \triangledown f.n \ ]$.

For simplicity, we restrict all variables to natural numbers. This implies that the domain of f is also restricted to the natural numbers.

We explore (11.7.5) by identifying invariants of Euclid's algorithm involving the function f. To determine an appropriate loop invariant, we take the right-hand side of (11.7.5) and calculate:

$$
\begin{array}{ll}
 & f.m \mathbin{\triangledown} f.n \\
= & \quad \{ \quad \text{the initial values of } x \text{ and } y \text{ are } m \text{ and } n, \text{ respectively} \quad \} \\
 & f.x \mathbin{\triangledown} f.y \\
= & \quad \{ \quad \text{suppose that } f.x \mathbin{\triangledown} f.y \text{ is invariant;} \\
 & \qquad \text{on termination: } x = m \triangledown n \ \wedge \ y = m \triangledown n \quad \} \\
 & f.(m \triangledown n) \mathbin{\triangledown} f.(m \triangledown n) \\
= & \quad \{ \quad \triangledown \text{ is idempotent} \quad \} \\
 & f.(m \triangledown n) \ .
\end{array}
$$

Property (11.7.5) is thus established under the assumption that $f.x \triangledown f.y$ is an invariant of the loop body. The next step is to determine what condition on f guarantees that $f.x \triangledown f.y$ is indeed invariant. Noting the symmetry in the loop body between x and y, the condition is easily calculated to be

$$[\ f.(x-y) \mathbin{\triangledown} f.y \ = \ f.x \mathbin{\triangledown} f.y \ \Leftarrow \ 0 < y < x \ ] \ .$$

Equivalently, by the rule of range translation ($x \ := \ x+y$), the condition can be written as

(11.7.6) $[\ f.x \mathbin{\triangledown} f.y \ = \ f.(x+y) \mathbin{\triangledown} f.y \ \Leftarrow \ 0 < x \ \wedge \ 0 < y \ ] \ .$

Formally, this means that

$$\text{“ } f \text{ distributes over } \triangledown \text{ ”} \ \Leftarrow \ (11.7.6) \ .$$

Incidentally, the converse of this property is also valid:

$$(11.7.6) \ \Leftarrow \ \text{“ } f \text{ distributes over } \triangledown \text{ ”} \ .$$

The simple calculation proceeds as follows:

$$
\begin{array}{ll}
 & f.(x+y) \mathbin{\triangledown} f.y \\
= & \quad \{ \quad f \text{ distributes over } \triangledown \quad \} \\
 & f.((x+y) \triangledown y) \\
= & \quad \{ \quad [\ (m+n) \triangledown n = m \triangledown n \ ] \quad \} \\
 & f.(x \triangledown y) \\
= & \quad \{ \quad f \text{ distributes over } \triangledown \quad \} \\
 & f.x \mathbin{\triangledown} f.y \ .
\end{array}
$$

By mutual implication we conclude that

$$\text{“ } f \text{ distributes over } \triangledown \text{ ”} \quad \equiv \quad (11.7.6) \quad .$$

We have now reached a point where we can determine if a function distributes over $\triangledown$. However, since (11.7.6) still has two occurrences of $\triangledown$, we want to refine it into simpler properties. Towards that end we turn our attention to the condition

$$f.x \triangledown f.y = f.(x+y) \triangledown f.y \quad ,$$

and we explore simple ways of guaranteeing that it is everywhere true. For instance, it is immediately obvious that any function that distributes over addition distributes over $\triangledown$. (Note that multiplication by a natural number is such a function.) The proof is very simple:

$$\begin{array}{ll} & f.(x+y) \triangledown f.y \\ = & \quad \{ \quad f \text{ distributes over addition} \quad \} \\ & (f.x+f.y) \triangledown f.y \\ = & \quad \{ \quad [\ (m+n)\triangledown n = m\triangledown n\ ] \quad \} \\ & f.x \triangledown f.y \quad . \end{array}$$

We can formulate the following lemma, which is a more general requirement:

Lemma 11.7.7 All functions f that satisfy

$$\langle \forall x,y:: \quad \langle \exists a,b \ : \ a\triangledown f.y = 1 \ : \ f.(x+y) \ = \ a \times f.x \ + \ b \times f.y \rangle \rangle$$

distribute over $\triangledown$.

Proof

$$\begin{array}{ll} & f.(x+y) \triangledown f.y \\ = & \quad \{ \quad f.(x+y) \ = \ a \times f.x \ + \ b \times f.y \quad \} \\ & (a \times f.x \ + \ b \times f.y) \triangledown f.y \\ = & \quad \{ \quad [\ (m+a\times n)\triangledown n = m\triangledown n\ ] \quad \} \\ & (a \times f.x) \triangledown f.y \\ = & \quad \{ \quad a \triangledown f.y = 1 \quad \text{and} \quad [\ (m\times p)\triangledown n = m\triangledown n \Leftarrow p\triangledown n = 1\ ] \quad \} \\ & f.x \triangledown f.y \quad . \end{array}$$

□

Note that since the discussion above is based on Euclid's algorithm, lemma 11.7.7 only applies to positive arguments. We now investigate the case where m or n is 0. We have, for $m = 0$:

$$
\begin{array}{ll}
& f.(0\triangledown n) \;=\; f.0 \triangledown f.n \\
= & \quad \{ \quad [\; 0\triangledown m = m \;] \quad \} \\
& f.n \;=\; f.0 \triangledown f.n \\
= & \quad \{ \quad [\; a\backslash b \;\equiv\; a = b\triangledown a \;] \quad \} \\
& f.n \setminus f.0 \\
\Leftarrow & \quad \{ \quad \text{obvious possibilities that make the expression valid} \\
& \qquad\quad \text{are } f.0 = 0,\ f.n = 1, \text{ or } f.n = f.0\text{; the first is the} \\
& \qquad\quad \text{interesting case} \quad \} \\
& f.0 = 0 \;\; .
\end{array}
$$

Hence, using the symmetry between m and n we have, for $m = 0$ or $n = 0$:

(11.7.8) $f.(m\triangledown n) \;=\; f.m \triangledown f.n \quad \Leftarrow \quad f.0 = 0 \;\; .$

The conclusion is that we can use (11.7.8) and lemma 11.7.7 to prove that a natural-valued function with domain $\mathbb{N}$ distributes over $\triangledown$. We do not know if the condition in lemma 11.7.7 is necessary for a function to distribute over $\triangledown$, but we do not know any function distributing over $\triangledown$ that does not satisfy the condition. This is a good problem to give to interested students.

□

11.8 Further reading

We recommend [BF10], which is on using Euclid's algorithm to prove and construct theorems. The main material of this scenario was taken from there.

SCENARIO 12

The King Who Loved Diagonals

12.1 Brief description and goals

This scenario is about a recreational problem that can be solved by using the solutions to two exercises from scenario 11. It can be used to practise formal modelling and to learn interesting properties related with the greatest common divisor of two numbers.

12.2 Problem

A very rich king wanted to thank one of his knights for leading his soldiers in a victorious battle. So he chose four large rooms of his castle that had the floor equally tiled. In each of these rooms, he drew a straight diagonal line connecting two opposite corners. Where the line crossed exactly four tiles, he placed one gold coin. (He actually ordered someone to draw the lines and place the coins. After all, *he was the king*!)

The four rooms were all of different sizes:

- Room 0: $(2^{11}-1)\times(2^{13}-1)$ tiles, i.e., 2047×8191 tiles
- Room 1: $(2^{15}-1)\times(2^{20}-1)$ tiles, i.e., 32767×1048575 tiles
- Room 2: $(2^{17}-1)\times(2^{21}-1)$ tiles, i.e., 131071×2097151 tiles
- Room 3: $(2^{20}-1)\times(2^{22}-1)$ tiles, i.e., 1048575×4194303 tiles

On the day that all coins were placed, he explained to the knight what he has done. He told him the sizes of the four rooms and he allowed him to collect all the gold coins from one of the rooms (and only one!).

Which room should the knight choose so that he collects a maximum number of gold coins?

12.3 Prerequisites

Scenario 11 (in particular, exercises 11.6.3 and 11.6.4).

12.4 Resolution and notes

The first step in our solution is to model the problem. A natural way to model it is by interpreting the floor as a Cartesian coordinate system, where the corners of the tiles represent integer coordinates. In that case, we can say that the diagonal of a room of size $p \times q$ is the straight line connecting the origin to the point (p,q). Moreover, using this model, we can say that the king placed one gold coin at each point of the diagonal line with positive integral coordinates.

But we know from exercise 11.6.3 of scenario 11, that the number of points with positive integral coordinates on the straight line joining the origin and the point (p,q) is the greatest common divisor of p and q, $p \bigtriangledown q$. Thus, the four rooms have the following number of gold coins:

- Room 0: $(2^{11}-1) \bigtriangledown (2^{13}-1)$ gold coins
- Room 1: $(2^{15}-1) \bigtriangledown (2^{20}-1)$ gold coins
- Room 2: $(2^{17}-1) \bigtriangledown (2^{21}-1)$ gold coins
- Room 3: $(2^{20}-1) \bigtriangledown (2^{22}-1)$ gold coins

We cannot compute these values immediately, because the arguments are too large. This suggests that we investigate properties that allow to simplify the computation of these values. Looking at the shape of the numbers, we see that both arguments are one less than powers of 2. So, if we define the function M as

$$M.k = 2^k - 1 \quad ,$$

then the number of gold coins in room 0, for example, is $M.11 \bigtriangledown M.13$. Now, a property that is connected to problem decomposition is distributivity. If function M distributes over $\bigtriangledown$, we have

$$M.k \bigtriangledown M.j \;=\; M.(k \bigtriangledown j) \quad .$$

This would help, because it is much easier to calculate the gcd of the exponents than to calculate the gcd of the powers. From exercise 11.6.4 of scenario 11, we know that a

function f that satisfies

$$\langle \forall x, y:: \quad \langle \exists a, b \ : \ a \bigtriangledown f.y \ = \ 1 \ : \ f.(x+y) \ = \ a \times f.x \ + \ b \times f.y \rangle \rangle$$

distributes over $\bigtriangledown$.

So, the function M distributes over $\bigtriangledown$, if there are integers a and b such that

$$M.(k+j) \ = \ a \times M.k \ + \ b \times M.j \quad \wedge \quad a \bigtriangledown M.j \ = \ 1 \quad ,$$

that is

$$2^{k+j}-1 \ = \ a \times (2^k-1) + b \times (2^j-1) \quad \wedge \quad a \bigtriangledown (2^j-1) = 1 \quad .$$

An obvious instantiation for a is 1 (because it makes the second conjunct trivially true). Choosing $a = 1$, we calculate b:

$$\begin{array}{ll} & 2^{k+j}-1 \ = \ (2^k-1) + b \times (2^j-1) \\ = & \quad \{ \quad \text{arithmetic} \quad \} \\ & 2^{k+j}-2^k = b \times (2^j-1) \\ = & \quad \{ \quad \text{multiplication distributes over addition} \quad \} \\ & 2^k \times (2^j-1) = b \times (2^j-1) \\ \Leftarrow & \quad \{ \quad \text{Leibniz} \quad \} \\ & b = 2^k \quad . \end{array}$$

We thus have

$$2^{k+j}-1 \ = \ 1 \times (2^k-1) + 2^k \times (2^j-1) \quad \wedge \quad 1 \bigtriangledown (2^j-1) = 1 \quad ,$$

and we can conclude that function M distributes over $\bigtriangledown$:

$$(2^k-1) \bigtriangledown (2^j-1) = 2^{k \bigtriangledown j}-1 \quad .$$

Thus, the four rooms have the following number of gold coins:

- Room 0: $2^{11 \bigtriangledown 13}-1$ gold coins
- Room 1: $2^{15 \bigtriangledown 20}-1$ gold coins
- Room 2: $2^{17 \bigtriangledown 21}-1$ gold coins
- Room 3: $2^{20 \bigtriangledown 22}-1$ gold coins

Clearly, the knight should choose room 1! The value of $15 \bigtriangledown 20$ is 5, whilst $11 \bigtriangledown 13 = 17 \bigtriangledown 21 = 1$ and $20 \bigtriangledown 22 = 2$. Choosing room 1, the knight would collect 31 gold coins.

12.5 Notes for the teacher

Model the problem The first step in our solution is to model the problem. We suggest the teacher to ask the students how would they model it. A natural way to model it is by interpreting the floor as a Cartesian coordinate system, where the corners of the tiles represent integer coordinates. In that case, we can say that the diagonal of a room of size $p \times q$ is the straight line connecting the origin to the point (p,q). Moreover, using this model, we can say that the king placed one gold coin at each point of the diagonal line with positive integral coordinates. It is very important that the students understand how the model reflects the data from the problem statement.

Use of relevant properties It is assumed (see the prerequisites) that the students have solved exercises 11.6.3 and 11.6.4 from scenario 11. Therefore, they should know from exercise 11.6.3 that the number of points with positive integral coordinates on the straight line joining the origin and the point (p,q) is the greatest common divisor of p and q, $p \bigtriangledown q$. This means that the four rooms have the following number of gold coins:

- Room 0: $(2^{11}-1) \bigtriangledown (2^{13}-1)$ gold coins
- Room 1: $(2^{15}-1) \bigtriangledown (2^{20}-1)$ gold coins
- Room 2: $(2^{17}-1) \bigtriangledown (2^{21}-1)$ gold coins
- Room 3: $(2^{20}-1) \bigtriangledown (2^{22}-1)$ gold coins

We recommend the teacher to ask the students if they know how to compute this numbers easily. The goal is to introduce the investigation of properties that allow to simplify the computation of these values. We recommend the teacher to ask the students if they notice any similarity in the arguments of $\bigtriangledown$. Looking at the shape of the numbers, we see that both arguments are one less than powers of 2. So, if the teacher defines the function M as

$$M.k = 2^k - 1 \quad ,$$

then the number of gold coins in room 0, for example, is $M.11 \bigtriangledown M.13$. Now, a property that is connected to problem decomposition is distributivity. And if function M distributes over $\bigtriangledown$, we have

$$M.k \bigtriangledown M.j \;=\; M.(k \bigtriangledown j) \quad .$$

We suggest the teacher to ask the students if they think distributivity helps. The goal is to understand that it helps, since it is much easier to calculate the gcd of the exponents than to calculate the gcd of the powers.

Distributivity proof From exercise 11.6.4 of scenario 11, the students should know that a function f that satisfies

$$\langle \forall x,y:: \quad \langle \exists a,b \;:\; a \bigtriangledown f.y \;=\; 1 \;:\; f.(x+y) \;=\; a \times f.x \;+\; b \times f.y \rangle \rangle$$

distributes over $\bigtriangledown$.

So, the teacher can ask the students how they can use this result with function M. Function M distributes over $\bigtriangledown$ if there are integers a and b such that

$$M.(k+j) \;=\; a \times M.k \;+\; b \times M.j \quad \wedge \quad a \bigtriangledown M.j \;=\; 1 \quad ,$$

that is

$$2^{k+j}-1 \;=\; a \times (2^k-1) + b \times (2^j-1) \quad \wedge \quad a \bigtriangledown (2^j-1) = 1 \quad .$$

We suggest the teacher to ask the students for possible values for the variable a. Obvious instantiations for a are 1, 2^j, and 2^j-2 (because they make the second conjunct trivially true). Choosing $a = 1$, we calculate b:

$$\begin{array}{ll} & 2^{k+j}-1 \;=\; (2^k-1) + b \times (2^j-1) \\ = & \quad \{ \quad \text{arithmetic} \quad \} \\ & 2^{k+j}-2^k = b \times (2^j-1) \\ = & \quad \{ \quad \text{multiplication distributes over addition} \quad \} \\ & 2^k \times (2^j-1) = b \times (2^j-1) \\ \Leftarrow & \quad \{ \quad \text{Leibniz} \quad \} \\ & b = 2^k \quad . \end{array}$$

(We recommend the teacher to leave all the steps for the students.) We thus have

$$2^{k+j}-1 \;=\; 1 \times (2^k-1) + 2^k \times (2^j-1) \quad \wedge \quad 1 \bigtriangledown (2^j-1) = 1 \quad ,$$

and we can conclude that function M distributes over $\bigtriangledown$:

$$(2^k-1) \bigtriangledown (2^j-1) = 2^{k \bigtriangledown j}-1 \quad .$$

Conclusion It should be easy for the students find out what is the room with the maximum number of gold coins:

- Room 0: $2^{11 \bigtriangledown 13}-1$ gold coins

- Room 1: $2^{15 \bigtriangledown 20} - 1$ gold coins
- Room 2: $2^{17 \bigtriangledown 21} - 1$ gold coins
- Room 3: $2^{20 \bigtriangledown 22} - 1$ gold coins

Clearly, the knight should choose room 1! The value of $15 \bigtriangledown 20$ is 5, whilst $11 \bigtriangledown 13 = 17 \bigtriangledown 21 = 1$ and $20 \bigtriangledown 22 = 2$. Choosing room 1, the knight would collect 31 gold coins.

Generalisations The teacher may want to generalise function M to the function of exercise 12.6.1.

12.5.1 Questions that the teacher should ask

Model the problem

- *Do you understand the problem?*

  The teacher has to be sure that the students understand what is required. Drawing some examples can help the students.

- *How can we model the problem?*

  A natural way to model the problem is by interpreting the floor as a Cartesian coordinate system, but we suggest the teacher to discuss other possibilities with the students. Once the model is presented, it is important that the students understand how it reflects the problem.

Use of relevant properties

- *Do you know any property that can be used to solve the problem?*

  The students should be aware that the number of points with integral positive coordinates on the straight line joining the origin and a point (p,q) is $p \bigtriangledown q$.

- *Can you easily compute these four values?*

  The goal of this question is to introduce the investigation of properties that allow to simplify the computation.

- *Do you notice any similarity between the arguments of* $\bigtriangledown$*?*

  The students should realise that both arguments are one less than powers of 2. This suggests the introduction of function M.

- *Do you think distributivity helps? Why?*

  The goal is to make sure that the students understand why distributivity can be used to solve the problem.

Distributivity proof

- *What values can we choose for variable a?*

  The goal of this question is to help the students simplify the proof. The second conjunct suggests three instantiations; the simplest can be used to calculate a value for b.

12.5.2 Questions that the teacher should not ask

Model the problem

- *Can we model the floor as a Cartesian coordinate system?*

  We think it is better to allow the students to think how they can model the problem, rather than suggesting ways of doing it.

Use of relevant properties

- *Can we use the result we have seen in scenario 11?*

  This question should only be asked if the students fail to recognise that the results they have seen in scenario 11 can be used.

- *Can you see that both arguments are one less than powers of 2?*

  We think this question reveals information that can easily be discovered by the students. Moreover, it is more valuable for the students to discover this by themselves.

12.5.3 Concepts that the teacher should introduce

Cartesian plane

Distributivity

12.6 Extensions and exercises

Exercise 12.6.1 (Generalisation) Prove that, for all integers k and m such that $0 < k^m$, the function f defined as

$$f.m = k^m - 1$$

distributes over the greatest common divisor.

□

12.7 Solutions to extensions and exercises

12.6.1 From exercise 11.6.4 of scenario 11, we know that a function f that satisfies

$$\langle \forall x, y :: \quad \langle \exists a, b \ : \ a \bigtriangledown f.y \ = \ 1 \ : \ f.(x+y) \ = \ a \times f.x \ + \ b \times f.y \rangle \rangle$$

distributes over $\bigtriangledown$.

So, the function f distributes over $\bigtriangledown$, if there are integers a and b such that

$$k^{m+n} - 1 \ = \ a \times (k^m - 1) + b \times (k^n - 1) \quad \wedge \quad a \bigtriangledown (k^n - 1) = 1 \quad .$$

An obvious instantiation for a is 1 (because it makes the second conjunct trivially true). Choosing $a = 1$, we calculate b:

$$\begin{array}{ll} & k^{m+n} - 1 \ = \ (k^m - 1) + b \times (k^n - 1) \\ = & \quad \{ \quad \text{arithmetic} \quad \} \\ & k^{m+n} - k^m = b \times (k^n - 1) \\ = & \quad \{ \quad \text{multiplication distributes over addition} \quad \} \\ & k^m \times (k^n - 1) = b \times (k^n - 1) \\ \Leftarrow & \quad \{ \quad \text{Leibniz} \quad \} \\ & b = k^m \quad . \end{array}$$

We thus have

$$k^{m+n} - 1 \ = \ 1 \times (k^m - 1) + k^m \times (k^n - 1) \quad \wedge \quad 1 \bigtriangledown (k^n - 1) = 1 \quad ,$$

and we can conclude that function f distributes over $\bigtriangledown$:

$$(k^m - 1) \bigtriangledown (k^n - 1) = k^{m \bigtriangledown n} - 1 \quad .$$

□

12.8 Further reading

We recommend [BF10], which is on using Euclid's algorithm to prove and construct theorems.

Index

□ (the block operator), 264
⇔ (if and only if), 19
≡ (equivales), 18, 69
≢ (inequivales), 216
abs (absolute value function), 75
▽ (greatest common divisor), 76
△ (least common multiple), 80
\ (division relation), 75
[ ] (everywhere brackets), 68
MATHIS, 9
Disquisitiones Arithmeticae, 102
Elements, 3, *see also* Euclid
How to Solve It, *see* Pólya, George

1945, 10

Abstraction, 34
Algebraic symmetry, *see* Symmetry
Algorithm, 14
    calculation of, 53
    definition of, 1
    formal manipulation of, 25, 247
    identifying algorithmic problems, 14
    inversion, 59
Algorithm inversion, 59
    alternative commands, 61
    definition, 60
    iterative commands, 62
    non-determinism, 63
    sequential composition, 60
Algorithmic problem solving, 1
    at Nottingham, 9
    education and research on, 8
    principles of, 13–35
    teaching of, 148
    techniques for, 36–64
Algorithmic proofs, 3, 124
Alloy specification language, 160
American Mathematical Monthly, 139
Assignment axiom, 26, 253
Associativity, 18
    associative reading, 18
    Boolean equality, 18, 194
Avoid guessing, 183
Avoid unnecessary detail, *see* Concision

Back, Ralph-Johan, 8
Backhouse, Roland Carl, 8, 9, 139, 246
Bag, *see* Multiset
Bell, Tim, 9
Bijection, 92, 93, 96
Binary tree, 44, 94
    binary search tree, 94
Bird, Richard, 8, 101
Bitwise exclusive-or, 248
Boolean equality, 18, 192
    continued equivalences, 195
Bound function, *see* Program termination
Bracket notation, 19, *see also* Notation, 224
Brilhart, John, 124
Brocot, Achille, 144
Brute-force, 154

Calculational method, 8, 17, *see also* Proof

format, 184
feedback from students, 158
symbol manipulation, 186, 255, 295
Calkin-Wilf tree, *see* Rational numbers
Cartesian coordinate system, 85
Case analysis, 22, 24, 32, 100, 154, 209
Chain, 296
Chess moves, 49
Chinese remainder theorem, *see* Congruences
Computational thinking, 9
PROBEs, 10
Concision
appropriate naming, 53, 59
importance of, 32
reducing the state space, 34
structural properties, 32
Congruences, 102, 266
basic properties, 103
cancellation properties, 112
Chinese remainder theorem, 117
congruent numbers, 102
continued congruences, 107
in practice, 111
modular exponentiation, 114
modulus, 102
noncongruent numbers, 102
nonresidue, 102
residue, 102
set of residues, 105
Conjunction, 22
Conjunctive reading, 18
Construction versus verification, 27
Constructive proof, 15, 136
Context-dependent parsing, 18
Contributions, 5
Control groups, 161
Conventional proofs, 2
Coprime numbers, 2, *see also* Greatest common divisor, 93

De Moor, Oege, 8
De Morgan rules, 208
Dedekind, Julius Wilhelm Richard, 124
Denumerable, 92
Determinant, *see* Matrices
Dickson, Leonard E., 124
Dijkstra, Edsger W., 7, 90
E. W. Dijkstra Archive, 8
Diophantus of Alexandria, 124
Disjunction, 22
Distributivity, 22, 48, 173
in logic puzzles, 216
relation with naming, 51
relation with problem decomposition, 48
Divide-and-conquer, 37
Divisibility theory, 68
division ordering, 76
division relation, 75
integer division, 68
number of divisors, 181
Division algorithm
invariant, 70
recursive algorithm, 73
specification, 69
termination proof, 70
Duncker, Karl, 10

Educational experiments, 158
Educational material, 148
Eindhoven quantifier notation, *see* Quantifiers
Eisenstein array, 140
Eisenstein, Ferdinand G. M., 93

Eisenstein-Stern tree, *see* Rational numbers, 140
Empty boxes, *see also* Invariants
  problem statement, 52
  solution, 53
Equational logic, 33
Equivalence, *see* Boolean equality
Euclid, 3, *see also* Euclid's algorithm
Euclid's algorithm
  as a construction interface, 87–102
  as a verification interface, 81–87
  construction of, 77, 303
  definition of, 3
  extended version, 84
  invariant, 77
  inversion of, 128
  termination proof, 57
Euclid's lemma, 82
Euler, Leonhard, 124
Existence theorems, 15, 53, 118
Extended Euclid's algorithm, 84
Extreme principle, 301

Feijen, Wim, 20
Fellows, Mike, 9
Fermat, Pierre de, 124
Fibonacci numbers
  definition of, 2
  relation with gcd, 90
Formalism
  formal modelling, 173
  on the use of, 17
Free information, *see* Symmetry
Functional specification, 25, 120
Future work, 161

Galois connection, 68, 163
Gardner, Martin, 11
Gauss, Carl Friedrich, 102, 124
gcd, *see* Greatest common divisor
Generating functions, 163
Gibbons, Jeremy, 101
Girard, Albert, 124
Go To statement, 7
Goal-oriented investigations, 27, 173, 184
Goat, Cabbage and Wolf
  naming the elements of the problem, 34
  problem statement, 14
Goldbach, Christian, 124
Graphs
  local degree of a vertex, 28
Greatest common divisor, 2, 79, 112, 151, 287, 303
  $\bigtriangledown$, 76, 288
  different orderings, 79
  distributivity properties, 82, 87, 307, 324
  geometrical property, 85, 316, 324
Greatest lower bound, *see* Infimum
Gries, David, 8
Guarded command language (GCL), 3, 247

Hadamard, Jacques, 10
Handshaking Lemma, 227
  problem definition, 29
Hardy, Godfrey Harold, 65
Hardy, Kenneth, 137
Hermite, Charles, 124
Hoare triples, 7, 25, 40, 247
Hoare, Charles Antony Richard, 7
Homomorphism, 98
Honsberger, Ross, 154, 226, 286

Indirect equality, 69
Infimum, 76, 79, 297

Infix notation, *see* Notation
Invariants, 4, 52, 81
    empty boxes, 244
    knockout tournament, 244
    moving a heavy armchair, 238
    mutilated chessboard, 244
    seen as constants, 86
    the chameleons of Camelot, 266
Investigative mathematics, 247
Isomorphism, 44

Knights and Knaves, *see* Logic puzzles
Knockout tournament, *see* Invariants
Knuth, Donald E., 3

Lagrange, Joseph-Louis, 124
Lattice, 296
Least common multiple, 80, 114, 287
    △, 80, 288
Legendre, Adrien-Marie, 102
Leibniz's rule, *see* Substitution of equals for equals
Lester, David, 101
Lexicographic ordering, 59, 98
Logic puzzles, 22, 154
    a logical race, 216
    distributivity, 216
    Knights and Knaves, 192
    Portia's casket, 23, 203
    simultaneous equations on Booleans, 22
Logical implication, 203, 207

Manipulation
    syntactic, 17, 295
    without interpretation, 17, 295
mathmeth.com - Discipline in Thought, 8
Matrices
    determinant, 94, 131
    transposition, 95
Mendes, Alexandra, 291
Mersenne function, 48, 151
    relation with gcd, 91
Mersenne, Marin, 124, *see also* Mersenne function
Method-oriented, 150, 152
Michalewicz, Matthew, 11
Michalewicz, Zbigniew, 11
Modular exponentiation, *see* Congruences
Monotonicity, 184, 186
Moving a heavy armchair, *see* Invariants
MPC, *see* Related work (mathematics of program construction)
Multiple assignments, *see* Simultaneous assignments
Multiset, 293
Muskat, Joseph B., 137
Mutilated chessboard, *see* Invariants
Mutual implication, 32

NetLogo, 160
Newman, Moshe, 93
noisrevni mhtiroglA, *see* Algorithm inversion
Nonconstructive proof, 15
Notation
    infix, 17
    introducing new, 19, 224
    relevance of, 17
Nuclear pennies game, 44
Nursery rhyme
    As I was going to St. Ives, 10

One-to-one correspondence, 92
Ordering relations, 184
Overflow, 253

Pólya, George, 28, 148, 153, 191

How to Solve It, 10, 153
Pappus, 28
Parity, 49
even, 49
standard solutions to problems on, 50
Partial order, 76
Perfect square, 173
Polynomials, 162
Portia's casket, *see* Logic puzzles
Postcondition, 25
Precondition, 25
Preorder, 75
Problem decomposition, 37
relation with distributivity, 48
Program inversion, *see* Algorithm inversion
Program termination, 57, 287
bound function, 57, 70, 287
Programming trick, 247
Proof format, 173
relevance of, 19
Puzzle-based learning, 11

Quantifiers, 29
Eindhoven quantifier notation, 19, 68, 227
unique existential quantifications, 216

Randomised trials, 161
Rational numbers
are denumerable, 96
Calkin-Wilf tree, 97
Eisenstein-Stern tree, 97, 140
enumerations, 92–102
lowest form, 93
mediant, 146
Newman's algorithm, 142
Stern-Brocot tree, 97, 142
Recreational problems, 148, 151
Regressive planning, *see* Goal-oriented investigations
Rejection
is part of academic life, 139
Related work, 6–11
Computer Science Unplugged, 9
classical problem solving, 10
mathematics of program construction, 6
structured derivations, 8
Rhind Mathematical Papyrus, 10

Safety properties, *see* Invariants
Schneider, Fred, 8
Schoenfeld, Alan H., 10
Sequential composition, 37
Serret, Joseph Alfred, 124
Seven-trees-in-one, 44
Simultaneous assignments, 27
Smith, Henry J. S., 124
Smith-Cornacchia algorithm, 137, *see also* Sum of two squares
Solution-oriented, 150, 152
Square numbers, 173
St. Petersburg City Olympiad, 287
Stern, Moritz A., 93, 140
Stern-Brocot tree, *see* Rational numbers
Subatomic particles, 280
Substitution of equals for equals, 19, 21, 24, 206
Sum of two squares, 16, 124, 136
main theorem, 136
Supremum, 297
Symmetry, 39
algebraic symmetry, 46, 173
free information, 39, 42
river-crossing problems, 39
the jealous couples, 40

Syntax-driven investigations, 255, 295
Synthesis, 27

Teaching scenarios, 148
    catalogue of, 149, 155, 173
    glossary, 162
    homework, 154
    how to create, 150
    project assignments, 154
    promoting self-discovery, 149, 153
    questions not to ask, 153
    questions to ask, 153
The chameleons of Camelot, 160, 263
    generalisation, 280
    invariant, 266
    non-determinism, 264
    problem decomposition, 38
    problem statement, 15
    removing the non-determinism, 267
    solution, 264
The jealous couples
    naming the elements of the problem, 35
    problem statement, 35
    solution, 40
The king who loved diagonals, 323, *see also* Greatest common divisor
The Risks Digest, 7
Theorem of Pythagoras, 153
Totally ordered set, 296
Turing, Alan Mathison, 57

Unitpotency, 249

van de Snepscheut, Jan, 8
van Gasteren, Antonetta J. M., 8, 59, 81

Wagon, Stan, 124, 137
Weakest precondition, 253
Wertheimer, Max, 10
Weyl, Hermann, 13
Whiteboards
    a problem for, 287
    use of, 178
Whitehead, Alfred North, 158
Wilf, Herbert, 161
Williams, Kenneth S., 137
Witten, Ian, 9
Working backwards, *see* Goal-oriented investigations

Zagier, Don, 124
Zeitz, Paul, 301