HOW TO PROVE IT

A Structured Approach

Second Edition

Daniel J. Velleman

Department of Mathematics and Computer Science

Amherst College
To Shelley
Contents

Preface to the Second Edition  ix
Preface xi
Introduction 1

1 Sentential Logic 8
1.1 Deductive Reasoning and Logical Connectives 8
1.2 Truth Tables 14
1.3 Variables and Sets 26
1.4 Operations on Sets 34
1.5 The Conditional and Biconditional Connectives 43

2 Quantificational Logic 55
2.1 Quantifiers 55
2.2 Equivalences Involving Quantifiers 64
2.3 More Operations on Sets 73

3 Proofs 84
3.1 Proof Strategies 84
3.2 Proofs Involving Negations and Conditionals 95
3.3 Proofs Involving Quantifiers 108
3.4 Proofs Involving Conjunctions and Biconditionals 124
3.5 Proofs Involving Disjunctions 136
3.6 Existence and Uniqueness Proofs 146
3.7 More Examples of Proofs 155

4 Relations 163
4.1 Ordered Pairs and Cartesian Products 163
4.2 Relations 171
4.3 More About Relations 180
4.4 Ordering Relations 189
4.5 Closures 202
4.6 Equivalence Relations 213

5 Functions
5.1 Functions 226
5.2 One-to-one and Onto 236
5.3 Inverses of Functions 245
5.4 Images and Inverse Images: A Research Project 255

6 Mathematical Induction
6.1 Proof by Mathematical Induction 260
6.2 More Examples 267
6.3 Recursion 279
6.4 Strong Induction 288
6.5 Closures Again 300

7 Infinite Sets
7.1 Equinumerous Sets 306
7.2 Countable and Uncountable Sets 315
7.3 The Cantor–Schröder–Bernstein Theorem 322

Appendix 1: Solutions to Selected Exercises 329
Appendix 2: Proof Designer 373
Suggestions for Further Reading 375
Summary of Proof Techniques 376
Index 381
Preface to the Second Edition

I would like to thank all of those who have sent me comments about the first edition. Those comments have resulted in a number of small changes throughout the text. However, the biggest difference between the first edition and the second is the addition of more than 200 new exercises. There is also an appendix containing solutions to selected exercises. Exercises for which solutions are supplied are marked with an asterisk. In most cases, the solution supplied is a complete solution; in some cases, it is a sketch of a solution, or a hint.

Some exercises in Chapters 3 and 4 are also marked with the symbol \( \vdash \). This indicates that these exercises can be solved using Proof Designer. Proof Designer is computer software that helps the user write outlines of proofs in elementary set theory, using the methods discussed in this book. Further information about Proof Designer can be found in an appendix, and at the Proof Designer Web site: http://www.cs.amherst.edu/~djv/pd/pd.html.
Preface

Students of mathematics and computer science often have trouble the first time they’re asked to work seriously with mathematical proofs, because they don’t know the “rules of the game.” What is expected of you if you are asked to prove something? What distinguishes a correct proof from an incorrect one? This book is intended to help students learn the answers to these questions by spelling out the underlying principles involved in the construction of proofs.

Many students get their first exposure to mathematical proofs in a high school course on geometry. Unfortunately, students in high school geometry are usually taught to think of a proof as a numbered list of statements and reasons, a view of proofs that is too restrictive to be very useful. There is a parallel with computer science here that can be instructive. Early programming languages encouraged a similar restrictive view of computer programs as numbered lists of instructions. Now computer scientists have moved away from such languages and teach programming by using languages that encourage an approach called “structured programming.” The discussion of proofs in this book is inspired by the belief that many of the considerations that have led computer scientists to embrace the structured approach to programming apply to proof-writing as well. You might say that this book teaches “structured proving.”

In structured programming, a computer program is constructed, not by listing instructions one after another, but by combining certain basic structures such as the if-else construct and do-while loop of the Java programming language. These structures are combined, not only by listing them one after another, but also by nesting one within another. For example, a program constructed by
nisting an if-else construct within a do-while loop would look like this:

```java
do
  if [condition]
    [List of instructions goes here.]
  else
    [Alternate list of instructions goes here.]
  while [condition]
```

The indenting in this program outline is not absolutely necessary, but it is a convenient method often used in computer science to display the underlying structure of a program.

Mathematical proofs are also constructed by combining certain basic proof structures. For example, a proof of a statement of the form “if \( P \) then \( Q \)” often uses what might be called the “suppose-until” structure: We suppose that \( P \) is true until we are able to reach the conclusion that \( Q \) is true, at which point we retract this supposition and conclude that the statement “if \( P \) then \( Q \)” is true. Another example is the “for arbitrary \( x \) prove” structure: To prove a statement of the form “for all \( x, P(x) \),” we declare \( x \) to be an arbitrary object and then prove \( P(x) \). Once we reach the conclusion that \( P(x) \) is true we retract the declaration of \( x \) as arbitrary and conclude that the statement “for all \( x, P(x) \)” is true. Furthermore, to prove more complex statements these structures are often combined, not only by listing one after another, but also by nesting one within another. For example, to prove a statement of the form “for all \( x, \text{ if } P(x) \text{ then } Q(x) \)” we would probably nest a “suppose-until” structure within a “for arbitrary \( x \) prove” structure, getting a proof of this form:

```
Let \( x \) be arbitrary.
  Suppose \( P(x) \) is true.
    [Proof of \( Q(x) \) goes here.]
  Thus, if \( P(x) \) then \( Q(x) \).
Thus, for all \( x \), if \( P(x) \) then \( Q(x) \).
```

As before, we have used indenting to make the underlying structure of the proof clear.

Of course, mathematicians don’t ordinarily write their proofs in this indented form. Our aim in this book is to teach students to write proofs in ordinary English paragraphs, just as mathematicians do, and not in the indented form. Nevertheless, our approach is based on the belief that if students are to succeed at writing such proofs, they must understand the underlying structure that proofs have. They must learn, for example, that sentences like “Let \( x \) be arbitrary” and “Suppose \( P \)” are not isolated steps in proofs, but are used to introduce the “for arbitrary \( x \) prove” and “suppose-until” proof structures. It is not uncommon for beginning students to use these sentences inappropriately in other ways.
Such mistakes are analogous to the programming error of using a “do” with no matching “while.”

Note that in our examples, the choice of proof structure is guided by the logical form of the statement being proven. For this reason, the book begins with elementary logic to familiarize students with the various forms that mathematical statements take. Chapter 1 discusses logical connectives, and quantifiers are introduced in Chapter 2. These chapters also present the basics of set theory, because it is an important subject that is used in the rest of the book (and throughout mathematics), and also because it serves to illustrate many of the points of logic discussed in these chapters.

Chapter 3 covers structured proving techniques in a systematic way, running through the various forms that mathematical statements can take and discussing the proof structures appropriate for each form. The examples of proofs in this chapter are for the most part chosen, not for their mathematical content, but for the proof structures they illustrate. This is especially true early in the chapter, when only a few proof techniques have been discussed, and as a result many of the proofs in this part of the chapter are rather trivial. As the chapter progresses the proofs get more sophisticated and more interesting, mathematically.

Chapters 4 and 5, on relations and functions, serve two purposes. First, they provide subject matter on which students can practice the proof-writing techniques from Chapter 3. And second, they introduce students to some fundamental concepts used in all branches of mathematics.

Chapter 6 is devoted to a method of proof that is very important in both mathematics and computer science: mathematical induction. The presentation builds on the techniques from Chapter 3, which students should have mastered by this point in the book.

Finally, in Chapter 7 many ideas from throughout the rest of the book are brought together to prove some of the most difficult and most interesting theorems in the book.

I would like to thank all those who read earlier drafts of the manuscript and made many helpful suggestions for improvements, in particular Lauren Cowles at Cambridge University Press, my colleague Professor Duane Bailey and his Discrete Mathematics class, who tried out earlier versions of some chapters, and finally my wife, Shelley, without whose constant encouragement this book would never have been written.