Preface

This book teaches two different sorts of things, woven together. It teaches you how to read and write mathematical proofs. It also provides a survey of basic mathematical objects, notation, and techniques which will be useful in later computer science courses. These include propositional and predicate logic, sets, functions, relations, modular arithmetic, counting, graphs, and trees.

Why learn formal mathematics?

Formal mathematics is relevant to computer science in several ways. First, it is used to create theoretical designs for algorithms and prove that they work correctly. This is especially important for methods that are used frequently and/or in applications where we don't want failures (aircraft design, Pentagon security, ecommerce). Only some people do these designs, but many people use them. The users need to be able to read and understand how the designs work.

Second, the skills you learn in doing formal mathematics correspond closely to those required to design and debug programs. Both require keeping track of what types your variables are. Both use inductive and/or recursive design. And both require careful proofreading skills. So what you learn in this class will also help you succeed in practical programming courses.

Third, when you come to design complex real software, you'll have to document what you've done and how it works. This is hard for many people to do well, but it's critical for the folks using your software. Learning how to describe mathematical objects clearly will translate into better skills

describing computational objects.

Everyone can do proofs

You probably think about proofs as something created by brilliant young theoreticians. Some of you **are** brilliant young theoreticians and you'll think this stuff is fun because it's what you naturally like to do. However, many of you are future software and hardware engineers. Some of you may never think of formal mathematics as "fun." That's ok. We understand. We're hoping to give you a sense of why it's pretty and useful, and enough fluency with it to communicate with the theoretical side of the field.

Many people think that proofs involve being incredibly clever. That is true for some steps in some proofs. That's where it helps to actually be a brilliant young theoretician. But many proofs are very routine. And many steps in clever proofs are routine. So there's quite a lot of proofs that all of us can do. And we can all read, understand, and appreciate the clever bits that we didn't think up ourselves.

In other words, don't be afraid. You can do proofs at the level required for this course, and future courses in theoretical computer science.

Is this book right for you?

This book is designed for students who have taken an introductory programming class of the sort intended for scientists or engineers. Algorithms will be presented in "pseudo-code," so it does not matter which programming language you have used. But you should have written programs that manipulate the contents of arrays e.g. sort an array of numbers. You should also have written programs that are recursive, i.e. in which a function (aka procedure aka method) calls itself.

We'll also assume that you have taken one semester of calculus. Though we will make little direct reference to calculus, we will assume a level of fluency with precalculus (algebra, geometry, logarithms, etc) that is typically developed during while taking calculus. If you already have significant expe-

rience writing proofs, including inductive proofs, this book may be too easy for you. You may wish to read some of the books listed as supplementary readings in Appendix B.

For instructors

This text is designed for a broad range of computer science majors, ranging from budding young theoreticians to capable programmers with very little interest in theory. It assumes only limited mathematical maturity, so that it can be used very early in the major. Therefore, a central goal is to explain the process of proof construction clearly to students who can't just pick it up by osmosis.

This book is designed to be succinct, so that students will read and absorb its contents. Therefore, it includes only core concepts and a selection of illustrative examples, with the expectation that the instructor will provide supplementary materials as needed (see Appendix B for useful follow-on books) and that students can look up a wider range of facts, definitions, and pictures, e.g. on the internet.

Although the core topics in this book are old and established, terminology and notation have changed over the years and vary somewhat between authors. To avoid overloading students, I have chosen one clean, modern version of notation, definitions, and terminology to use consistently in the main text. Common variations are documented at the end of each chapter. If students understand the underlying concepts, they should have no difficulty adapting when they encounter different conventions in other books.

Many traditional textbooks do a careful and exhaustive treatment of each topic from first principles, including foundational constructions and theorems which prove that the conceptual system is well-formed. However, the most abstract parts of many topics are hard for beginners to absorb and the importance of the foundational development is lost on most students at this level. The early parts of this textbook remove most of this clutter, to focus more clearly on the key concepts and constructs. At the end, we revisit certain topics to look at more abstract examples and selected foundational issues. See Appendix C for a quick guide to topic rearrangement.