Preface

It has become common sense that problems in software may have catastrophic consequences, ranging from economic to human safety aspects. This has led to the development of software norms that prescribe the use of particular development methods. These norms advise the use of *validation* techniques, and at the highest levels of certification, of *mathematical* validation.

Program verification is the area of computer science that studies mathematical methods for checking that a program conforms to its specification. It is part of the broader area of *formal methods*, which groups together very heterogeneous rigorous approaches to systems and software development. Program verification is concerned with properties of code, which can be studied in more than one way. From methods based on logic, in particular the combined use of a *program logic* and *first-order theories*, to other approaches like *software model checking*; *abstract interpretation*; and *symbolic execution*.

This book is a self-contained introduction to program verification using logic-based methods, assuming only basic knowledge of standard mathematical concepts that should be familiar to any computer science student. It includes a self-contained introduction to propositional logic and first-order reasoning with theories, followed by a study of program verification that combines theoretical and practical aspects—from a program logic to the use of a realistic tool for the verification of C programs, through the generation of verification conditions and the treatment of errors.

More specifically, we start with an overview of propositional (Chap. 3) and first-order (Chap. 4) logic, and then go on (Chap. 5) to establish a setting for the verification of sequential programs, building on the axiomatic semantics (based on Hoare logic) of a generic *While* language, whose behaviour is specified using *preconditions* and *postconditions*.

It is then shown how a set of first-order proof obligations, usually known as *verification conditions*, can be mechanically generated from a program and a specification that the program is required to satisfy (Chap. 6). Concrete programming languages can be obtained by instantiating the language of program expressions, which we illustrate with a language of integer expressions and then a language of integer-type arrays.

This setting is then adapted to cover the treatment of *safety properties* of programs, by explicitly incorporating in the semantics the possibility of runtime errors occurring. The set of generated verification conditions is extended to guard against that possibility (Chap. 7). This is illustrated in the concrete languages with arithmetic errors and out-of-bounds array accesses.

Finally the setting is extended to allow for the specification and verification of programs consisting of mutually-recursive procedures, based on annotations included in the code, usually known as *contracts* (Chap. 8). The idea here is that each procedure has its own public specification, called a contract in this context, and when reasoning individually about the correctness of an individual procedure one can assume that all procedures are correct with respect to their respective specifications.

The last part of the book illustrates the specification (Chap. 9) and verification (Chap. 10) of programs of a real-world programming language. We have chosen to use the ACSL specification language for C programs, and the Frama-C/Jessie tool for their verification. It is important to understand that these are being actively developed as this book is written, but we do believe that the core of the specification language and verification tools will remain unchanged.

Our emphasis is on the integrated presentation, and on making explicit the bits of the story that may be harder to grasp. Program verification already has a long history but remains a very active research field, and this book contains some material that as far as we can tell can only be found in research papers. The approach followed has logic at its core, and the theories that support the reasoning (which may include user-supplied parts) are always explicitly referred. The generation of verification conditions is performed by an algorithm that is synthesized from Hoare logic in a step-by-step fashion. The important topic of *specification adaptation* is also covered, since it is essential to the treatment of procedures. Finally, we develop a general framework for the verification of contract-based mutually-recursive procedures, which is an important step towards understanding the use of modern verification tools based on contracts.

The book will prepare readers to use verification methods in practice, which we find cannot be done correctly with only a superficial understanding of what is going on. Although verification methods tend to be progressively easier to use, software engineers can greatly benefit from an understanding of why the verification methods and tools are sound, as well as what their limitations are, and how they are implemented. Readers will also be capable of constructing their own verification platforms for specific languages—but it must be said at this point that one of the most challenging aspects, the treatment of heap-based dynamic data structures based on a memory model—is completely left out.

As mentioned before, program verification belongs to what are usually described as *formal methods* for the development of software. Chapter 1 motivates the importance of program verification, and explains how formal approaches to software development and validation are more and more important in the software industry. Chapter 2 offers an overview of formal methods that will hopefully give the reader a more general context for understanding the program verification techniques covered in the rest of the book.

Notes to the Instructor The book assumes knowledge of the basic mathematical concepts (like sets, functions, and relations) that are usually taught in introductory courses on mathematics for computer science students. No knowledge of logical concepts is assumed.

The accompanying website for this book is http://www.di.uminho.pt/rsd-book, where teaching material, solutions to selected exercises, source code, and links to useful online resources can be found.

Acknowledgments The authors are indebted to their students and colleagues at the departments of Informatics of the universities of Minho and Beira Interior, who have provided feedback and comments on drafts of several chapters. The idea of writing this book came up when the third author was on leave at the Department of Computer Science of the University of Porto (Faculty of Science), which kindly provided a welcoming environment for writing the first draft chapters.