Preface

Goals

Keeping up with new developments in most areas of computing requires familiarity with basic logical concepts. In particular, your success in most aspects of software development significantly depends on your ability to reason correctly, to communicate your reasoning, and to understand and evaluate the reasoning of others. These abilities are critical for anyone who does feasibility analysis, systems analysis, problem specification, database design or management, program design, coding, testing, verification, problem diagnosis, documentation, software maintenance, or research in any of these areas.

If you know little about how logic can be used in software development and if you want to know more, then this book may be of use to you. After reading it you should be better able to reason about software development, to communicate your reasoning, to distinguish between good and bad reasoning, and to read professional literature, which presumes knowledge of elementary logic.

On the other hand, if you think that your own logical abilities are good enough, but that many other people are sadly deficient in these abilities, then please give copies of this book to those who need it.

Overview and Features

Applications of logic to software development are emphasized throughout. Examples involving program instructions are expressed in pseudocode so that the book makes no use of any particular programming language. It is divided into three parts. Part I is about language and logical form. It explains how to find and represent the logical forms of statements expressed in English. It shows how to use a subset of English, here called *logical English*, to represent both the meanings and logical forms of statements. Logical English is intermediate between informal but meaningful English and the largely meaningless and severely abstract notations commonly used in formal logic, and used here in Part III. This intermediate role

resembles the role of pseudocode. Like pseudocode, logical English is still recognizable English. And like pseudocode, it is a helpful bridge between informal English and a highly formal notation. They differ in that logical English is used to express statements and conditions while pseudocode is used to express instructions. Part I ends by describing how to use logical English to clarify and express data structure definitions, problem specifications, and conditions in instructions.

Part II is about truth in the ordinary "material" sense of that term. It shows how to use truth tables to determine the truth or falsity of a complex statement built using connectives such as "not", "and", "or", and "if…then…" if you know the truth or falsity of its component statements. Truth conditions for statements involving the quantifiers "all" and "some" are also described. Following that, several computer-related applications of this material are discussed. The final chapter shows how to apply truth value calculations to forward and backward tracing of program execution.

Part III is about "logical" truth. Logical truth is a generalization of material truth. It involves ignoring the meanings and material truth values of individual statements and focusing only on their logical forms. Much of what is known about how to reason correctly can best be stated in terms of logical forms. For example, the statement form "P or not P" is logically true. As a result, no matter what statement is used in place of P, the resulting statement of the form "P or not P" is materially true.

This part also explains and shows how to test statements for logical equivalence, logical implication, and logical redundancy, and how to test arguments for validity and soundness. It also explains how to use rules of inference to make proofs. It then describes how to apply these concepts to problem specifications. This is followed by a proof that no computer program that solves the problem of determining whether any arbitrarily selected program will halt with any arbitrarily selected input can be written. That bad news is followed by the good news that it is possible, though difficult, to prove that a program is correct relative to a problem specification, without doing any testing. Examples showing how to do this in simple cases are given. The last chapter briefly discusses some topics not covered here, e.g. logic testing and quantum computing. It includes a few pointers to additional sources of information about these topics.

Suggested Uses

This book is designed to be used by computer professionals and students who want to study on their own without an instructor. It is also suitable as the primary text for instructor led introductory courses on logic for students who are studying any of the computing disciplines. Earlier versions of much of this material were class tested in a college level course I teach on logic and its applications to computing. In addition, the three parts of the book can be the basis for three or more short professional development courses.

Target Audiences

Students and professionals who expect to be involved with any aspect of software development are the target audiences for this book. No prior knowledge of formal logic is assumed. Some knowledge of software development is assumed.

Audience Resources

Many examples as well as many practice exercises, along with solutions to half of them, are included here. Solutions to all the exercises can be had by instructors at http://www.springer.com/978-1-84800-081-0

Readers can contact me at http://www.logicforsoftwaredevelopment.com. I intend to post corrections to newly discovered errors, pointers to additional resources, logic jokes, and other related information there.

Readers are also urged to use the email link there to send error reports and constructive suggestions for improving this book.

Acknowledgements

I am grateful to the monks, administrators, staff, and faculty at Belmont Abbey College for providing me an environment in which it was possible to write this book. Special thanks, and some commiseration, are due to my students in CS325 who suffered through earlier drafts of much of this material.

Gene Lauver and Bill Seltzer, who have many years of experience in real-world computing, found numerous errors, made many helpful suggestions, and even worked many of the exercises “for fun”. Without their help, this book would have been noticeably less useful.

Finally, several anonymous reviewers made helpful suggestions. Wayne Wheeler and Catherine Brett at Springer, UK, provided enthusiastic, cheerful, patient, and professional editorial support.

Robert Lover
Charlotte, NC