The Calculus of Computation

Aaron R. Bradley · Zohar Manna

The Calculus of Computation

Decision Procedures with Applications to Verification

With 60 Figures

Springer

Authors

Aaron R. Bradley
Zohar Manna

Gates Building, Room 481
Stanford University
Stanford, CA 94305
USA

arbrad@cs.stanford.edu
manna@cs.stanford.edu

Library of Congress Control Number: 2007932679

ACM Computing Classification (1998): B.8, D.1, D.2, E.1, F.1, F.3, F.4, G.2, I.1, I.2

ISBN 978-3-540-74112-1 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springer.com

Typesetting by the authors
Production: LE-TEX Jelonek, Schmidt & Vöckler GbR, Leipzig
Cover design: KünkelLopka Werbeagentur, Heidelberg

Printed on acid-free paper 45/3180/YL - 5 4 3 2 1 0

To my wife,

Sarah

A.R.B.

To my grandchildren,

Itai
Maya
Ori

Z.M.

Preface

Logic is the calculus of computation. Forty-five years ago, John McCarthy predicted in *A Basis for a Mathematical Theory of Computation* that "the relationship between computation and mathematical logic will be as fruitful in the next century as that between analysis and physics in the last". The field of *computational logic* emerged over the past few decades in partial fulfillment of that vision. Focusing on producing efficient and powerful algorithms for deciding the satisfiability of formulae in logical theories and fragments, it continues to push the frontiers of general computer science.

This book is about computational logic and its applications to *program verification*. Program verification is the task of analyzing the correctness of a program. It encompasses the formal specification of what a program should do and the formal proof that the program meets this specification. The reasoning power that computational logic offers revolutionized the field of verification. Ongoing research will make verification standard practice in software and hardware engineering in the next few decades. This acceptance into everyday engineering cannot come too soon: software and hardware are becoming ever more ubiquitous and thus ever more the source of failure.

We wrote this book with an undergraduate and beginning graduate audience in mind. However, any computer scientist or engineer who would like to enter the field of computational logic or apply its products should find this book useful.

Content

The book has two parts. Part I, *Foundations*, presents first-order logic, induction, and program verification. The methods are general. For example, Chapter 2 presents a complete proof system for first-order logic, while Chapter 5 describes a relatively complete verification methodology. Part II, *Algorithmic Reasoning*, focuses on specialized algorithms for reasoning about fragments of first-order logic and for deducing facts about programs. Part II trades generality for decidability and efficiency.

The first three chapters of Part I introduce first-order logic. Chapters 1 and 2 begin our presentation with a review of propositional and predicate logic. Much of the material will be familiar to the reader who previously studied logic. However, Chapter 3 on first-order theories will be new to many readers. It axiomatically defines the various first-order theories and fragments that we study and apply throughout the rest of the book. Chapter 4 reviews induction, introducing some forms of induction that may be new to the reader. Induction provides the mathematical basis for analyzing program correctness.

Chapter 5 turns to the primary motivating application of computational logic in this book, the task of verifying programs. It discusses *specification*, in which the programmer formalizes in logic the (sometimes surprisingly vague) understanding that he has about what functions should do; *partial correctness*, which requires proving that a program or function meets a given specification if it halts; and *total correctness*, which requires proving additionally that a program or function always halts. The presentation uses the simple programming language pi and is supported by the verifying compiler πVC (see **The πVC System**, below, for more information on πVC). Chapter 6 suggests strategies for applying the verification methodology.

Part II on *Algorithmic Reasoning* begins in Chapter 7 with quantifier-elimination methods for limited integer and rational arithmetic. It describes an algorithm for reducing a quantified formula in integer or rational arithmetic to an equivalent formula without quantifiers.

Chapter 8 begins a sequence of chapters on decision procedures for quantifier-free and other fragments of theories. These fragments of first-order theories are interesting for three reasons. First, they are sometimes decidable when the full theory is not (see Chapters 9, 10, and 11). Second, they are sometimes efficiently decidable when the full theory is not (compare Chapters 7 and 8). Finally, they are often useful; for example, proving the verification conditions that arise in the examples of Chapters 5 and 6 requires just the fragments of theories studied in Chapters 8–11. The simplex method for linear programming is presented in Chapter 8 as a decision procedure for deciding satisfiability in rational and real arithmetic without multiplication.

Chapters 9 and 11 turn to decision procedures for non-arithmetical theories. Chapter 9 discusses the classic congruence closure algorithm for equality with uninterpreted functions and extends it to reason about data structures like lists, trees, and arrays. These decision procedures are for quantifier-free fragments only. Chapter 11 presents decision procedures for larger fragments of theories that formalize array-like data structures.

Decision procedures are most useful when they are combined. For example, in program verification one must reason about arithmetic and data structures simultaneously. Chapter 10 presents the Nelson-Oppen method for combining decision procedures for quantifier-free fragments. The decision procedures of Chapters 8, 9, and 11 are all combinable using the Nelson-Oppen method.

Chapter 12 presents a methodology for constructing *invariant generation procedures*. These procedures reason inductively about programs to aid in

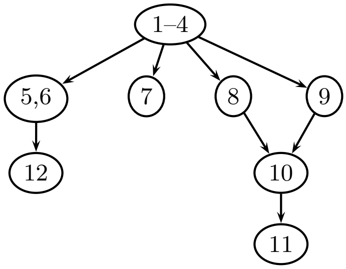

Fig. 0.1. The chapter dependency graph

verification. They relieve some of the burden on the programmer to provide program annotations for verification purposes. For now, developing a static analysis is one of the easiest ways of bringing formal methods into general usage, as a typical static analysis requires little or no input from the programmer. The chapter presents a general methodology and two instances of the method for deducing arithmetical properties of programs.

Finally, Chapter 13 suggests directions for further reading and research.

Teaching

This book can be used in various ways and taught at multiple levels. Figure 0.1 presents a dependency graph for the chapters. There are two main tracks: the *verification track*, which focuses on Chapters 1–4, 5, 6, and 12; and the *decision procedures track*, which focuses on Chapters 1–4 and 7–11. Within the decision procedures track, the reader can focus on the *quantifier-free decision procedures track*, which skips Chapters 7 and 11. The reader interested in quickly obtaining an understanding of modern combination decision procedures would prefer this final track.

We have annotated several sections with a * to indicate that they provide additional depth that is unnecessary for understanding subsequent material. Additionally, all proofs may be skipped without preventing a general understanding of the material.

Each chapter ends with a set of exercises. Some require just a mechanical understanding of the material, while others require a conceptual understanding or ask the reader to think beyond what is presented in the book. These latter exercises are annotated with a *. For certain audiences, additional exercises might include implementing decision procedures or invariant generation procedures and exploring certain topics in greater depth (see Chapter 13).

In our courses, we assign program verification exercises from Chapters 5 and 6 throughout the term to give students time to develop this important skill. Learning to verify programs is about as difficult for students as learning to program in the first place. Specifying and verifying programs also strengthens the students' facility with logic.

Bibliographic Remarks

Each chapter ends with a section entitled **Bibliographic Remarks** in which we attempt to provide a brief account of the historical context and development of the chapter's material. We have undoubtedly missed some important contributions, for which we apologize. We welcome corrections, comments, and historical anecdotes.

The πVC System

We implemented a *verifying compiler* called πVC to accompany this text. It allows users to write and verify annotated programs in the `pi` programming language. The system and a set of examples, including the programs listed in this book, are available for download from `http://theory.stanford.edu/~arbrad/pivc`. We plan to update this website regularly and welcome readers' comments, questions, and suggestions about πVC and the text.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Nos. CSR-0615449 and CNS-0411363 and by Navy/ONR contract N00014-03-1-0939. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the Navy/ONR. The first author received additional support from a Sang Samuel Wang Stanford Graduate Fellowship.

We thank the following people for their comments throughout the writing of this book: Miquel Bertran, Andrew Bradley, Susan Bradley, Chang-Seo Park, Caryn Sedloff, Henny Sipma, Matteo Slanina, Sarah Solter, Fabio Somenzi, Tomás Uribe, the students of CS156, and Alfred Hofmann and the reviewers and editors at Springer. Their suggestions helped us to improve the presentation substantially. Remaining errors and shortcomings are our responsibility.

Stanford University, June 2007

Aaron R. Bradley
Zohar Manna

Contents

Part I Foundations

1 Propositional Logic 3
1.1 Syntax 4
1.2 Semantics 6
1.3 Satisfiability and Validity 8
1.3.1 Truth Tables 9
1.3.2 Semantic Arguments 10
1.4 Equivalence and Implication 14
1.5 Substitution 16
1.6 Normal Forms 18
1.7 Decision Procedures for Satisfiability 21
1.7.1 Simple Decision Procedures 21
1.7.2 Reconsidering the Truth-Table Method 22
1.7.3 Conversion to an Equisatisfiable Formula in CNF 24
1.7.4 The Resolution Procedure 27
1.7.5 DPLL 28
1.8 Summary 31
Bibliographic Remarks 32
Exercises 32

2 First-Order Logic 35
2.1 Syntax 35
2.2 Semantics 39
2.3 Satisfiability and Validity 42
2.4 Substitution 45
2.4.1 Safe Substitution 47
2.4.2 Schema Substitution 48
2.5 Normal Forms 51
2.6 Decidability and Complexity 53
2.6.1 Satisfiability as a Formal Language 53

2.6.2 Decidability 54
2.6.3 ★Complexity 54
2.7 ★Meta-Theorems of First-Order Logic 56
2.7.1 Simplifying the Language of FOL 57
2.7.2 Semantic Argument Proof Rules 58
2.7.3 Soundness and Completeness 58
2.7.4 Additional Theorems 61
2.8 Summary 66
Bibliographic Remarks 67
Exercises 67

3 First-Order Theories 69
3.1 First-Order Theories 69
3.2 Equality 71
3.3 Natural Numbers and Integers 73
3.3.1 Peano Arithmetic 73
3.3.2 Presburger Arithmetic 75
3.3.3 Theory of Integers 76
3.4 Rationals and Reals 79
3.4.1 Theory of Reals 80
3.4.2 Theory of Rationals 82
3.5 Recursive Data Structures 84
3.6 Arrays 87
3.7 ★Survey of Decidability and Complexity 90
3.8 Combination Theories 91
3.9 Summary 92
Bibliographic Remarks 93
Exercises 93

4 Induction 95
4.1 Stepwise Induction 95
4.2 Complete Induction 99
4.3 Well-Founded Induction 102
4.4 Structural Induction 108
4.5 Summary 110
Bibliographic Remarks 111
Exercises 111

5 Program Correctness: Mechanics 113
5.1 pi: A Simple Imperative Language 114
5.1.1 The Language 115
5.1.2 Program Annotations 118
5.2 Partial Correctness 123
5.2.1 Basic Paths: Loops 125
5.2.2 Basic Paths: Function Calls 131

5.2.3 Program States . . . 135
5.2.4 Verification Conditions . . . 136
5.2.5 P-Invariant and P-Inductive . . . 142
5.3 Total Correctness . . . 143
5.4 Summary . . . 149
Bibliographic Remarks . . . 150
Exercises . . . 151

6 Program Correctness: Strategies . . . 153
6.1 Developing Inductive Annotations . . . 153
6.1.1 Basic Facts . . . 154
6.1.2 The Precondition Method . . . 156
6.1.3 A Strategy . . . 162
6.2 Extended Example: QuickSort . . . 164
6.2.1 Partial Correctness . . . 167
6.2.2 Total Correctness . . . 171
6.3 Summary . . . 172
Bibliographic Remarks . . . 173
Exercises . . . 173

Part II Algorithmic Reasoning

7 Quantified Linear Arithmetic . . . 183
7.1 Quantifier Elimination . . . 184
7.1.1 Quantifier Elimination . . . 184
7.1.2 A Simplification . . . 185
7.2 Quantifier Elimination over Integers . . . 185
7.2.1 Augmented Theory of Integers . . . 185
7.2.2 Cooper's Method . . . 187
7.2.3 A Symmetric Elimination . . . 194
7.2.4 Eliminating Blocks of Quantifiers . . . 195
7.2.5 ★Solving Divides Constraints . . . 196
7.3 Quantifier Elimination over Rationals . . . 200
7.3.1 Ferrante and Rackoff's Method . . . 200
7.4 ★Complexity . . . 204
7.5 Summary . . . 204
Bibliographic Remarks . . . 205
Exercises . . . 205

8 Quantifier-Free Linear Arithmetic . . . 207
8.1 Decision Procedures for Quantifier-Free Fragments . . . 207
8.2 Preliminary Concepts and Notation . . . 209
8.3 Linear Programs . . . 213
8.4 The Simplex Method . . . 218

8.4.1 From M to M_0 . . . 219
8.4.2 Vertex Traversal . . . 223
8.4.3 ★Complexity . . . 237
8.5 Summary . . . 237
Bibliographic Remarks . . . 238
Exercises . . . 238

9 Quantifier-Free Equality and Data Structures . . . 241
9.1 Theory of Equality . . . 242
9.2 Congruence Closure Algorithm . . . 244
9.2.1 Relations . . . 245
9.2.2 Congruence Closure Algorithm . . . 247
9.3 Congruence Closure with DAGs . . . 251
9.3.1 Directed Acyclic Graphs . . . 251
9.3.2 Basic Operations . . . 254
9.3.3 Congruence Closure Algorithm . . . 255
9.3.4 Decision Procedure for T_{E}-Satisfiability . . . 256
9.3.5 ★Complexity . . . 258
9.4 Recursive Data Structures . . . 259
9.5 Arrays . . . 263
9.6 Summary . . . 265
Bibliographic Remarks . . . 266
Exercises . . . 267

10 Combining Decision Procedures . . . 269
10.1 Combining Decision Procedures . . . 269
10.2 Nelson-Oppen Method: Nondeterministic Version . . . 271
10.2.1 Phase 1: Variable Abstraction . . . 271
10.2.2 Phase 2: Guess and Check . . . 273
10.2.3 Practical Efficiency . . . 274
10.3 Nelson-Oppen Method: Deterministic Version . . . 276
10.3.1 Convex Theories . . . 276
10.3.2 Phase 2: Equality Propagation . . . 278
10.3.3 Equality Propagation: Implementation . . . 282
10.4 ★Correctness of the Nelson-Oppen Method . . . 283
10.5 ★Complexity . . . 287
10.6 Summary . . . 288
Bibliographic Remarks . . . 288
Exercises . . . 288

11 Arrays . . . 291
11.1 Arrays with Uninterpreted Indices . . . 292
11.1.1 Array Property Fragment . . . 292
11.1.2 Decision Procedure . . . 294
11.2 Integer-Indexed Arrays . . . 299

11.2.1 Array Property Fragment 300
11.2.2 Decision Procedure 301
11.3 Hashtables 304
11.3.1 Hashtable Property Fragment 305
11.3.2 Decision Procedure 306
11.4 Larger Fragments 308
11.5 Summary 309
Bibliographic Remarks 310
Exercises 310

12 Invariant Generation 311
12.1 Invariant Generation 311
12.1.1 Weakest Precondition and Strongest Postcondition 312
12.1.2 ★General Definitions of wp and sp 315
12.1.3 Static Analysis 316
12.1.4 Abstraction 319
12.2 Interval Analysis 325
12.3 Karr's Analysis 333
12.4 ★Standard Notation and Concepts 341
12.5 Summary 344
Bibliographic Remarks 345
Exercises 345

13 Further Reading 347

References 351

Index 357