Texts in Theoretical Computer Science. An EATCS Series

Series editors

Monika Henzinger, Faculty of Science, Universität Wien, Wien, Austria Juraj Hromkovič, Department of Computer Science, ETH Zürich, Zürich, Switzerland Mogens Nielsen, Department of Computer Science, Aarhus Universitet, Denmark Grzegorz Rozenberg, Leiden Centre of Advanced Computer Science, Leiden, The Netherlands Arto Salomaa, Turku Centre of Computer Science, Turku, Finland More information about this series at http://www.springer.com/series/3214

Daniel Kroening · Ofer Strichman

Decision Procedures

An Algorithmic Point of View

Second Edition



Daniel Kroening Computing Laboratory University of Oxford Oxford UK Ofer Strichman Information Systems Engineering The William Davidson Faculty of Industrial Engineering and Management Technion – Israel Institute of Technology Haifa Israel

ISSN 1862-4499 Texts in Theoretical Computer Science. An EATCS Series ISBN 978-3-662-50496-3 ISBN 978-3-662-50497-0 (eBook) DOI 10.1007/978-3-662-50497-0

Library of Congress Control Number: 2016957285

© Springer-Verlag Berlin Heidelberg 2008, 2016

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made.

Printed on acid-free paper

This Springer imprint is published by Springer Nature The registered company is Springer-Verlag GmbH Germany The registered company address is: Heidelberger Platz 3, 14197 Berlin, Germany

By Randal E. Bryant

Research in decision procedures started several decades ago, but both their practical importance and the underlying technology have progressed rapidly in the last five years. Back in the 1970s, there was a flurry of activity in this area, mostly centered at Stanford and the Stanford Research Institute (SRI), motivated by a desire to apply formal logic to problems in artificial intelligence and software verification. This work laid foundations that are still in use today. Activity dropped off through the 1980s and 1990s, accompanied by a general pessimism about automated formal methods. A conventional wisdom arose that computer systems, especially software, were far too complex to reason about formally.

One notable exception to this conventional wisdom was the success of applying Boolean methods to hardware verification, beginning in the early 1990s. Tools such as model checkers demonstrated that useful properties could be proven about industrial-scale hardware systems, and that bugs that had otherwise escaped extensive simulation could be detected. These approaches improved on their predecessors by employing more efficient logical reasoning methods, namely ordered binary decision diagrams and Boolean satisfiability solvers. The importance of considering algorithmic efficiency, and even lowlevel concerns such as cache performance, became widely recognized as having a major impact on the size of problems that could be handled.

Representing systems at a detailed Boolean level limited the applicability of early model checkers to control-intensive hardware systems. Trying to model data operations, as well as the data and control structures found in software, leads to far too many states, when every bit of a state is viewed as a separate Boolean signal.

One way to raise the level of abstraction for verifying a system is to view data in more abstract terms. Rather than viewing a computer word as a collection of 32 Boolean values, it can be represented as an integer. Rather than viewing a floating-point multiplier as a complex collection of Boolean functions, many verification tasks can simply view it as an "uninterpreted function" computing some repeatable function over its inputs. From this approach came a renewed interest in decision procedures, automating the process of reasoning about different mathematical forms. Some of this work revived methods dating back many years, but alternative approaches also arose that made use of Boolean methods, exploiting the greatly improved performance of Boolean satisfiability (SAT) solvers. Most recently, decision procedures have become quite sophisticated, using the general framework of search-based SAT solvers, integrated with methods for handling the individual mathematical theories.

With the combination of algorithmic improvements and the improved performance of computer systems, modern decision procedures can readily handle problems that far exceed the capacity of their forebearers from the 1970s. This progress has made it possible to apply formal reasoning to both hardware and software in ways that disprove the earlier conventional wisdom. In addition, the many forms of malicious attacks on computer systems have created a program execution environment where seemingly minor bugs can yield serious vulnerabilities, and this has greatly increased the motivation to apply formal methods to software analysis.

Until now, learning the state of the art in decision procedures required assimilating a vast amount of literature, spread across journals and conferences in a variety of different disciplines and over multiple decades. Ideas are scattered throughout these publications, but with no standard terminology or notation. In addition some approaches have been shown to be unsound, and many have proven ineffective. I am therefore pleased that Daniel Kroening and Ofer Strichman have compiled the vast amount of information on decision procedures into a single volume. Enough progress has been made in the field that the results will be of interest to those wishing to apply decision procedures. At the same time, this is a fast-moving and active research community, making this work essential reading for the many researchers in the field.

Foreword to the second edition

By Leonardo de Moura¹, Microsoft Research

Decision procedures are already making a profound impact on a number of application areas, and had become so efficient in practice in the last 15 years (mostly since the introduction of a new generation of SAT solvers, and in particular the introduction of Chaff in 2001) that numerous practical problems that were beyond our reach beforehand are now routinely solved in seconds. Yet, they draw on a combination of some of the most fundamental areas in computer science as well as discoveries from the past century of symbolic logic. They combine the problem of Boolean satisfiability with domains such as those studied in convex optimization and term-manipulating symbolic systems. They involve the decision problem, completeness and incompleteness of logical theories, and finally complexity theory.

It is an understatement to say that we use decision procedures in Microsoft on a daily basis. Applications include security testing, static code analysis, constraint solving and software verification, to cite a few. With more than five hundred machine years, security testing at Microsoft is the largest computational usage ever for decision procedures. It has been instrumental in uncovering hundreds of subtle security critical bugs that traditional testing methods have been unable to find.

This book is both an introduction to this fascinating topic and a reference for advanced developers. It dedicates a chapter to many of the useful theories (and their combination), and describes some of their applications in software engineering, including techniques that we use for static code analysis at Microsoft. The new information in this second edition is important for both the researcher and the practitioner, since it includes general quantification (an algorithm such as E-matching), updates on efficient SAT solving and related problems (such as incremental solving), effectively propositional reasoning (EPR), and other topics of great value.

¹ The main developer of Z3, an award-winning SMT solver

Preface

A decision procedure is an algorithm that, given a decision problem, terminates with a correct yes/no answer. In this book, we focus on decision procedures for decidable first-order theories that are useful in the context of automated software and hardware verification, theorem proving, compiler optimization, and, since we are covering propositional logic, any problem that is in the complexity class NP and is not polynomial. The range of modeling languages that we cover in this book—propositional logic, linear arithmetic, bitvectors, quantified formulas etc.—and the modeling examples that we include for each of those, will assist the reader to translate their particular problem and solve it with one of the publically available tools. The common term for describing this field is *Satisfiability Modulo Theories*, or SMT for short, and software that solves SMT formulas is called an *SMT solver*.

Since coping with the above-mentioned tasks on an industrial scale depends critically on effective decision procedures, SMT is a vibrant and prospering research subject for many researchers around the world, both in academia and in industry. Intel, AMD, ARM and IBM are some of the companies that routinely apply decision procedures in circuit verification with ever-growing capacity requirements. Microsoft is developing an SMT solver and applies it routinely in over a dozen code analysis tools. Every user of Microsoft Windows and Microsoft Office therefore indirectly enjoys the benefits of this technology owing to the increased reliability and resilience to hacker attacks of these software packages. There are hundreds of smaller, less famous companies that use SMT solvers for various software engineering tasks, and for solving various planning and optimization problems.

There are now numerous universities that teach courses dedicated to decision procedures; occasionally, the topic is also addressed in courses on algorithms or on logic for computer science. The primary goal of this book is to serve as a textbook for an advanced undergraduate- or graduate-level computer science course. It does not assume specific prior knowledge beyond what is expected from a third-year undergraduate computer science student. The



Fig. 1. Decision procedures can be rather complex ... those that we consider in this book take formulas of different theories as input, possibly mix them (using the Nelson–Oppen procedure—see Chap. 10), decide their satisfiability ("YES" or "NO"), and, if yes, provide a satisfying assignment

book may also help graduate students entering the field, who can save the effort to gather information from what seems to be an endless list of articles.

The decision procedures that we describe in this book draw from diverse fields such as graph theory, logic, operations research, and artificial intelligence. These procedures have to be highly efficient, since the problems they solve are inherently hard. They never seem to be efficient enough, however: what we want to be able to prove is always harder than what we *can* prove. Their asymptotic complexity and their performance in practice must always be pushed further. These characteristics are what makes this topic so compelling for research and teaching.

Which Theories? Which Algorithms?

A first-order theory can be considered "interesting", at least from a practical perspective, if it fulfills at least these two conditions:

- 1. The theory is expressive enough to model a real decision problem. Moreover, it is more expressive or more natural for the purpose of expressing some models in comparison with theories that are easier to decide.
- 2. The theory is either decidable or semidecidable, and more efficiently solvable than theories that are more expressive, at least in practice if not in theory.²

All the theories described in this book fulfill these two conditions. Furthermore, they are all used in practice. We illustrate applications of each theory with examples representative of real problems, whether they may be verification of C programs, verification of hardware circuits, or optimizing compilers. Background in any of these problem domains is not assumed, however.

Other than in one chapter, all the theories considered are quantifier-free. The problem of deciding them is NP-complete. In this respect, they can all be seen as alternative modeling languages that can be solved with a variety of decision procedures. They differ from each other mainly in how naturally they can be used for modeling various decision problems. For example, consider the theory of equality, which we describe in Chap. 4: this theory can express any Boolean combination of Boolean variables and expressions of the form $x_1 = x_2$, where x_1 and x_2 are variables ranging over, for example, the natural numbers. The problem of satisfying an expression in this theory can be reduced to a satisfiability problem of a propositional logic formula (and vice versa). Hence, there is no difference between propositional logic and the theory of equality in terms of their ability to model decision problems. However, many problems are more naturally modeled with the equality operator and non-Boolean variables.

For each theory that is discussed, there are many alternative decision procedures in the literature. Effort was made to select those procedures that are known to be relatively efficient in practice, and at the same time are based on what we believe to be an interesting idea. In this respect, we cannot claim to have escaped the natural bias that one has towards one's own line of research.

Every year, new decision procedures and tools are being published, and it is impossible to write a book that reports on this moving target of "the most efficient" decision procedures (the worst-case complexity of most of the competing procedures is the same). Moreover, many of them have never been thoroughly compared with one another. We refer readers who are interested in the latest developments in this field to the SMT-LIB web page, as well as to the results of the annual tool competition SMT-COMP (see Appendix A). The SMT-COMP competitions are probably the best way to stay up to date as to the relative efficiency of the various procedures and the tools that implement

 $^{^2}$ Terms such as expressive and decidable have precise meanings, and we will define them in the first chapter.

them. One should not forget, however, that it takes much more than a good algorithm to be efficient in practice.

The Structure and Nature of This Book

The first chapter is dedicated to basic concepts that should be familiar to third- or fourth-year computer science students, such as formal proofs, the satisfiability problem, soundness and completeness, and the trade-off between expressiveness and decidability. It also includes the theoretical basis for the rest of the book. From Sect. 1.5 onwards, the chapter is dedicated to more advanced issues that are necessary as a general introduction to the book, and are therefore recommended even for advanced readers. Chapters 2 and 3 describe how propositional formulas are checked for satisfiability, and then how this capability can be extended to more sophisticated theories. These chapters are necessary for understanding the rest of the book. Chapters 4–11 are mostly self-contained, and generally do not rely on references to material other than that in the first three chapters. The last chapter describes the application of these methods for verifying the correctness of software, and for solving various problems in computational biology.

The mathematical symbols and notations are mostly local to each chapter. Each time a new symbol is introduced, it appears in a rounded box in the margin of the page for easy reference. All chapters conclude with problems, bibliographic notes, and a glossary of symbols.

Teaching with This Book

We are aware of 38 courses worldwide that list the first edition of this book as the textbook of the course, in addition to our own courses in the Technion (Haifa, Israel) and Oxford University (UK). Our own courses are combined undergraduate and graduate courses. The slides that were used in these courses, as well as links to other resources and ideas for projects, appear on the book's web page (www.decision-procedures.org). Source code of a C++ library for rapid development of decision procedures can also be downloaded from this page. This library provides the necessary infrastructure for programming many of the algorithms described in this book, as explained in Appendix B. Implementing one of these algorithms was a requirement in the course, and it proved successful. It even led several students to their thesis topic.

Notes for the Second Edition

The sales of the first edition of this book crossed, apparently, the threshold above which the publisher asks the authors to write a second one... Writing this edition was a necessity for more fundamental reasons, however: at the time the first edition was written (2004–2008) the field now called SMT was in its infancy, without the standard terminology and canonic algorithms that it has now. What constituted the majority of Chap. 11 in the first edition (propositional encodings and the DPLL(T) framework) became so dominant in the years that have passed that we expanded it and brought it forward to Chap. 3. In turn, most of the so-called *eager-encoding* algorithms have been moved to Chap. 11. In addition, we updated Chap. 2 with further modern SAT heuristics, added a section about incremental satisfiability, and added a section on the related constraint satisfaction problem (CSP). To the quantifiers chapter (Chap. 9) we added a section about general quantification using *E-matching* and a section about the Bernays–Schönfinkel–Ramsey fragment of first-order logic (also called EPR). Finally, we added a new chapter (Chap. 12) on the application of SMT for software engineering in industry, partially based on writings of Nikolaj Bjørner and Leonardo de Moura from Microsoft Research, and for solving problems in computational biology based on writings of Hillel Kugler, also from Microsoft Research.

Acknowledgments

Many people read drafts of this manuscript and gave us useful advice. We would like to thank, in alphabetical order, those who helped in the first edition: Domagoj Babic, Josh Berdine, Hana Chockler, Leonardo de Moura, Benny Godlin, Alberto Griggio, Alan Hu, Wolfgang Kunz, Shuvendu Lahiri, Albert Oliveras Llunell, Joel Ouaknine, Hendrik Post, Sharon Shoham, Aaron Stump, Cesare Tinelli, Ashish Tiwari, Rachel Tzoref, Helmut Veith, Georg Weissenbacher, and Calogero Zarba, and those who helped with the second edition: Francesco Alberti, Alberto Griggio, Marijn Heule, Zurab Khasidashvili, Daniel Le Berre, Silvio Ranise, Philipp Ruemmer, Natarajan Shankar, and Cesare Tinelli. We thank Ilya Yodovsky Jr. for the drawing in Fig. 1.

Sep. 2016

Daniel Kroening University of Oxford, United Kingdom Ofer Strichman Technion, Haifa, Israel

Contents

1	Int	roduction and Basic Concepts	1
	1.1	Two Approaches to Formal Reasoning	3
		1.1.1 Proof by Deduction $\ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots$	3
		1.1.2 Proof by Enumeration	4
		1.1.3 Deduction and Enumeration	5
	1.2	Basic Definitions	5
	1.3	Normal Forms and Some of Their Properties	8
	1.4	The Theoretical Point of View	14
		1.4.1 The Problem We Solve	17
		1.4.2 Our Presentation of Theories	18
	1.5	Expressiveness vs. Decidability	18
	1.6	Boolean Structure in Decision Problems	20
	1.7	Logic as a Modeling Language	22
	1.8	Problems	23
	1.9	Glossary	24
2	Dec	cision Procedures for Propositional Logic	27
	2.1	Propositional Logic	27
		2.1.1 Motivation	27
	2.2	SAT Solvers	29
		2.2.1 The Progress of SAT Solving	29
		2.2.2 The CDCL Framework	31
		2.2.3 BCP and the Implication Graph	32
		2.2.4 Conflict Clauses and Resolution	38
		2.2.5 Decision Heuristics	42
		2.2.6 The Resolution Graph and the Unsatisfiable Core	15
		2.2.7 Incremental Satisfiability	46
		2.2.8 From SAT to the Constraint Satisfaction Problem 4	48
		2.2.9 SAT Solvers: Summary	19
	2.3	Problems	50
		2.3.1 Warm-up Exercises	50
		2.3.2 Propositional Logic	51
		2.3.3 Modeling	51
		2.3.4 Complexity	52

		2.3.5	CDCL SAT Solving	53
		2.3.6	Related Problems	54
	2.4	Bibli	ographic Notes	54
	2.5	Gloss	sary	58
3	Fro	m Proj	positional to Quantifier-Free Theories	59
	3.1	Intro	duction	59
	3.2	An (Overview of $DPLL(T)$	61
	3.3	Form	nalization	64
	3.4	Theo	bry Propagation and the $DPLL(T)$ Framework	66
		3.4.1	Propagating Theory Implications	66
		3.4.2	Performance, Performance	69
		3.4.3	Returning Implied Assignments Instead of Clauses	70
		3.4.4	Generating Strong Lemmas	71
		3.4.5	Immediate Propagation	72
	3.5	Prob	lems	72
	3.6	Bibli	ographic Notes	73
	3.7	Gloss	sary	75
4	Ea	alities	and Uninterpreted Functions	77
т	4 1	Intro	aduction	77
	1.1	411	Complexity and Expressiveness	77
		412	Boolean Variables	78
		413	Bemoving the Constants: a Simplification	78
	42	Unin	tempered Functions	79
	1.2	421	How Uninterpreted Functions Are Used	80
		4.2.1	An Example: Proving Equivalence of Programs	81
	43	Cone	The Example. Proving Equivalence of Programs	85
	44	Func	tional Consistency Is Not Enough	86
	4.5	Two	Examples of the Use of Uninterpreted Functions	87
	1.0	451	Proving Equivalence of Circuits	89
		4.5.1	Verifying a Compilation Process with Translation Vali-	00
		1.0.2	dation	91
	46	Prob	lems	92
	$\frac{1.0}{4.7}$	Bibli	jographic Notes	92
	4.8	Glos	sary	95
F	T in	oon An	ithmatic	07
9	5 1	ear Ar	duction	97
	0.1	5 1 1	Solvers for Lincon Arithmetic	91
	59	0.1.1 The	Simpley Algorithm	90
	0.2	591	A Normal Form	- 99 - 00
		5.2.1 5.2.2	Register of the Simpley Algorithm	99 100
		5.2.2 5.9.9	Simpley with Upper and Lower Pounds	100
		0.2.3 E 0 4	In property of the second seco	102
		0.2.4	Incremental Problems	100

	5.3	The	Branch and Bound Method					106
		5.3.1	Cutting Planes					108
	5.4	Four	rier–Motzkin Variable Elimination					112
		5.4.1	Equality Constraints					112
		5.4.2	Variable Elimination					112
		5.4.3	Complexity					115
	5.5	The	Omega Test					115
		5.5.1	Problem Description					115
		5.5.2	Equality Constraints					116
		5.5.3	Inequality Constraints					119
	5.6	Prep	processing					124
		5.6.1	Preprocessing of Linear Systems					124
		5.6.2	Preprocessing of Integer Linear Systems					125
	5.7	Diffe	erence Logic					126
		5.7.1	Introduction					126
		5.7.2	A Decision Procedure for Difference Logic					128
	5.8	Prob	blems					129
		5.8.1	Warm-up Exercises					129
		5.8.2	The Simplex Method					129
		5.8.3	Integer Linear Systems					130
		5.8.4	Omega Test					130
		5.8.5	Difference Logic					131
	5.9	Bibl	iographic Notes	•				131
	5.10	Glos	sary	•		•		133
c	D!+	1 74-						195
0		Dit 1	rs Voeton Anithmotia					195
	0.1	DIU-	Suptor	•	• •	·	•••	100
		0.1.1	Notation	•	• •	·	•••	100
		0.1.2	Notation	•	• •	·	•••	107
	69	0.1.3 Doci	ding Dit Vostor Arithmetic with Elettoning	•	•••	·		149
	0.2	Deci	ding Dit-vector Antimietic with Flattening				•••	142
		691	Converting the Sheleton					149
		6.2.1	Converting the Skeleton	•		•		142
	6 9	6.2.1 6.2.2	Converting the Skeleton	•			 	142 144 146
	6.3	6.2.1 6.2.2 Incr	Converting the Skeleton	•	· ·		 	142 144 146
	6.3	6.2.1 6.2.2 Incre 6.3.1 6.2.2	Converting the Skeleton	•	· · ·		 	142 144 146 146
	6.3	6.2.1 6.2.2 Incre 6.3.1 6.3.2	Converting the Skeleton Arithmetic Operators	•	· · ·		 	142 144 146 146 148 140
	6.3 6.4	6.2.1 6.2.2 Incre 6.3.1 6.3.2 Fixe	Converting the Skeleton Arithmetic Operators	- · ·	· · ·		· · · · · ·	142 144 146 146 148 149 140
	6.3 6.4	6.2.1 6.2.2 Incre 6.3.1 6.3.2 Fixe 6.4.1 6.4.2	Converting the Skeleton Arithmetic Operators	· · ·	· · ·	• • • •	· · · · · ·	142 144 146 146 148 149 149
	6.3 6.4	6.2.1 6.2.2 Incre 6.3.1 6.3.2 Fixe 6.4.1 6.4.2 Prob	Converting the Skeleton	• •	· · ·		· · · · · · · · ·	142 144 146 146 148 149 149 150
	6.36.46.5	6.2.1 6.2.2 Incre 6.3.1 6.3.2 Fixe 6.4.1 6.4.2 Prob	Converting the Skeleton Arithmetic Operators emental Bit Flattening Some Operators Are Hard Abstraction with Uninterpreted Functions d-Point Arithmetic Semantics Flattening Semantics	• •	· · · · · · · · · · · · · · · · · · ·		· · · · · · · · ·	142 144 146 146 148 149 149 150 151
	6.36.46.5	6.2.1 6.2.2 Incre 6.3.1 6.3.2 Fixe 6.4.1 6.4.2 Prob 6.5.1 6.5.2	Converting the Skeleton Arithmetic Operators emental Bit Flattening Some Operators Are Hard Abstraction with Uninterpreted Functions d-Point Arithmetic Semantics Flattening Semantics Semantics Bit Level Encodings of Bit Vector Arithmetic	· · ·	· · · · · · · · · · · · · · · · · · ·		· · · · · · · · · · · · · · ·	142 144 146 146 148 149 149 150 151 151 151
	6.36.46.5	6.2.1 6.2.2 Incre 6.3.1 6.3.2 Fixe 6.4.1 6.4.2 Prob 6.5.1 6.5.2 6.5.2	Converting the Skeleton Arithmetic Operators emental Bit Flattening Some Operators Are Hard Abstraction with Uninterpreted Functions ad-Point Arithmetic Semantics Flattening Semantics Semantics Bit-Level Encodings of Bit-Vector Arithmetic Using Solvers for Linear Arithmetic	· · · · · · · · · · · · · · · · · · ·		· · · ·	· · · · · · · · · · · · · · ·	$142 \\ 144 \\ 146 \\ 146 \\ 148 \\ 149 \\ 149 \\ 150 \\ 151 \\ 151 \\ 152 $
	 6.3 6.4 6.5 6.6 	6.2.1 6.2.2 Incre 6.3.1 6.3.2 Fixe 6.4.1 6.4.2 Prob 6.5.1 6.5.2 6.5.3 Bibl	Converting the Skeleton			· · · ·	· ·	$142 \\ 144 \\ 146 \\ 146 \\ 148 \\ 149 \\ 149 \\ 150 \\ 151 \\ 151 \\ 152 \\ 152 \\ 152 \\ 154 $
	 6.3 6.4 6.5 6.6 6.7 	6.2.1 6.2.2 Incre 6.3.1 6.3.2 Fixe 6.4.1 6.4.2 Prob 6.5.1 6.5.2 6.5.3 Bibl	Converting the Skeleton Arithmetic Operators			· · · · · · · · · · · · · · · · · · ·	· · · · · ·	$142 \\ 144 \\ 146 \\ 146 \\ 148 \\ 149 \\ 149 \\ 150 \\ 151 \\ 151 \\ 152 \\ 152 \\ 152 \\ 154 \\ 156 $

v	57	T	гτ
Λ	v	1	ш

7 Arrays		157
7.1 Introduction \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots		157
7.1.1 Syntax		158
7.1.2 Semantics \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots		159
7.2 Eliminating the Array Terms		159
7.3 A Reduction Algorithm for a Fragment of the Array The	eory.	162
7.3.1 Array Properties		162
7.3.2 The Reduction Algorithm		163
7.4 A Lazy Encoding Procedure		165
7.4.1 Incremental Encoding with $DPLL(T)$		165
7.4.2 Lazy Instantiation of the Read-Over-Write Axiom		165
7.4.3 Lazy Instantiation of the Extensionality Rule		167
7.5 Problems		169
7.6 Bibliographic Notes		170
7.7 Glossary		171
8 Pointer Logic		173
8.1 Introduction	• • •	173
8.1.1 Pointers and Their Applications	• • •	173
8.1.2 Dynamic Memory Allocation		174
8.1.3 Analysis of Programs with Pointers	• • •	176
8.2 A Simple Pointer Logic	• • •	177
$8.2.1 \text{Syntax} \dots \dots$		177
8.2.2 Semantics	• • •	179
8.2.3 Axiomatization of the Memory Model		180
8.2.4 Adding Structure Types	• • •	181
8.3 Modeling Heap-Allocated Data Structures	• • •	182
8.3.1 Lists \ldots	• • •	182
8.3.2 Trees	• • •	183
8.4 A Decision Procedure		185
8.4.1 Applying the Semantic Translation		185
8.4.2 Pure variables $\dots \dots \dots$		187
8.4.3 Partitioning the Memory	• • •	100
8.5 Rule-Based Decision Procedures	• • •	189
8.5.1 A Reachability Predicate for Linked Structures	• • •	190
8.5.2 Deciding Reachability Predicate Formulas	• • •	191
8.0 Problems	• • •	194
8.0.1 Formulas	• • •	194
0.0.2 Reachability reducates		190 106
o. Dibilographic Notes		100
0.0 G1055d1 y		190
9 Quantified Formulas		199
9.1 Introduction \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots		100
		199

		9.1.2	Example: Quantified Disjunctive Linear Arithmetic	203
	9.2	Quar	ntifier Elimination	203
		9.2.1	Prenex Normal Form	203
		9.2.2	Quantifier Elimination Algorithms	205
		9.2.3	Quantifier Elimination for Quantified Boolean Formulas	206
		9.2.4	Quantifier Elimination for Quantified Disjunctive Lin-	
			ear Arithmetic	209
	9.3	Searc	ch-Based Algorithms for QBF	210
	9.4	Effec	tively Propositional Logic	212
	9.5	Gene	ral Quantification	215
	9.6	Prob	lems	222
		9.6.1	Warm-up Exercises	222
		9.6.2	OBF	223
		9.6.3	ÊPR	224
		9.6.4	General Quantification	224
	9.7	Bibli	ographic Notes	225
	9.8	Gloss	sarv	227
			0	
10	Dec	iding a	a Combination of Theories	229
	10.1	Intro	duction \ldots	229
	10.2	Preli	minaries 	229
	10.3	The	Nelson–Oppen Combination Procedure	231
		10.3.1	Combining Convex Theories	231
		10.3.2	Combining Nonconvex Theories	234
		10.3.3	Proof of Correctness of the Nelson–Oppen Procedure	237
	10.4	Prob	lems	240
	10.5	Bibli	ographic Notes	240
	10.6	Gloss	sary	244
	-			
11	Pro	positio	nal Encodings	245
	11.1	Lazy	vs. Eager Encodings	245
	11.2	From	Uninterpreted Functions to Equality Logic	245
		11.2.1	Ackermann's Reduction	246
		11.2.2	Bryant's Reduction	249
	11.3	The	Equality Graph	253
	11.4	Simp	Diffications of the Formula	255
	11.5	A Gi	aph-Based Reduction to Propositional Logic	259
	11.6	Equa	dities and Small-Domain Instantiations	262
		11.6.1	Some Simple Bounds	263
		11.6.2	Graph-Based Domain Allocation	265
		11.6.3	The Domain Allocation Algorithm	266
		11.6.4	A Proof of Soundness	269
		11.6.5	Summary	271
	11.7	Acke	rmann's vs. Bryant's Reduction: Where Does It Matter?	272
	11.8	Prob	lems	273

	11.9 11.1(11.8.1 Reductions 2 11.8.2 Domain Allocation 2 Bibliographic Notes 2 0 Glossary 2	274 276 276 276 279
12	App	lications in Software Engineering and Computational	
	Biol	logy 2	281
	12.1	Introduction	281
	12.2	Bounded Program Analysis	283
		12.2.1 Checking Feasibility of a Single Path	283
		12.2.2 Checking Feasibility of All Paths in a Bounded Program 2	287
	12.3	Unbounded Program Analysis	289
		12.3.1 Overapproximation with Nondeterministic Assignments 2	289
		12.3.2 The Overapproximation Can Be Too Coarse 2	291
		12.3.3 Loop Invariants $\ldots \ldots 2$	294
		12.3.4 Refining the Abstraction with Loop Invariants 2	295
	12.4	SMT-Based Methods in Biology	297
		12.4.1 DNA Computing	298
		12.4.2 Uncovering Gene Regulatory Networks	300
	12.5	Problems	302
		12.5.1 Warm-up Exercises	302
		12.5.2 Bounded Symbolic Simulation	302
		12.5.3 Overapproximating Programs	303
	12.6	Bibliographic Notes	304
Δ	SM	T-LIB: a Brief Tutorial	200
11	A.1	The SMT-LIB Initiative	309
	A.2	The SMT-LIB File Interface	310
		A.2.1 Propositional Logic	311
		A.2.2 Arithmetic	312
		A.2.3 Bit-Vector Arithmetic	313
		A.2.4 Arrays	313
		A.2.5 Equalities and Uninterpreted Functions	314
_			
В	AC	++ Library for Developing Decision Procedures 3	315
	B.I	Introduction	315
	B.2	Graphs and Trees	316
	Ъŝ	B.2.1 Adding "Payload"	318
	B.3	Parsing	318
		B.3.1 A Grammar for First-Order Logic	318
		B.3.2 The Problem File Format	520 001
		D.3.5 A Class for Storing Identifiers	521 551
	D 4	D.3.4 The Farse Tree	521 599
	Б.4	UNF and SAL	522 552
		B.4.1 Generating UNF	522

B.4.2Converting the Propositional Skeleton	$\begin{array}{c} 325\\ 325 \end{array}$		
References	329		
Tools index			
Algorithms index	349		
Index	351		