Texts in Theoretical Computer Science. An EATCS Series

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

Springer

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

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

Foreword to the first edition

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 low-level 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 func-

tion" 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 Moura[1], 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

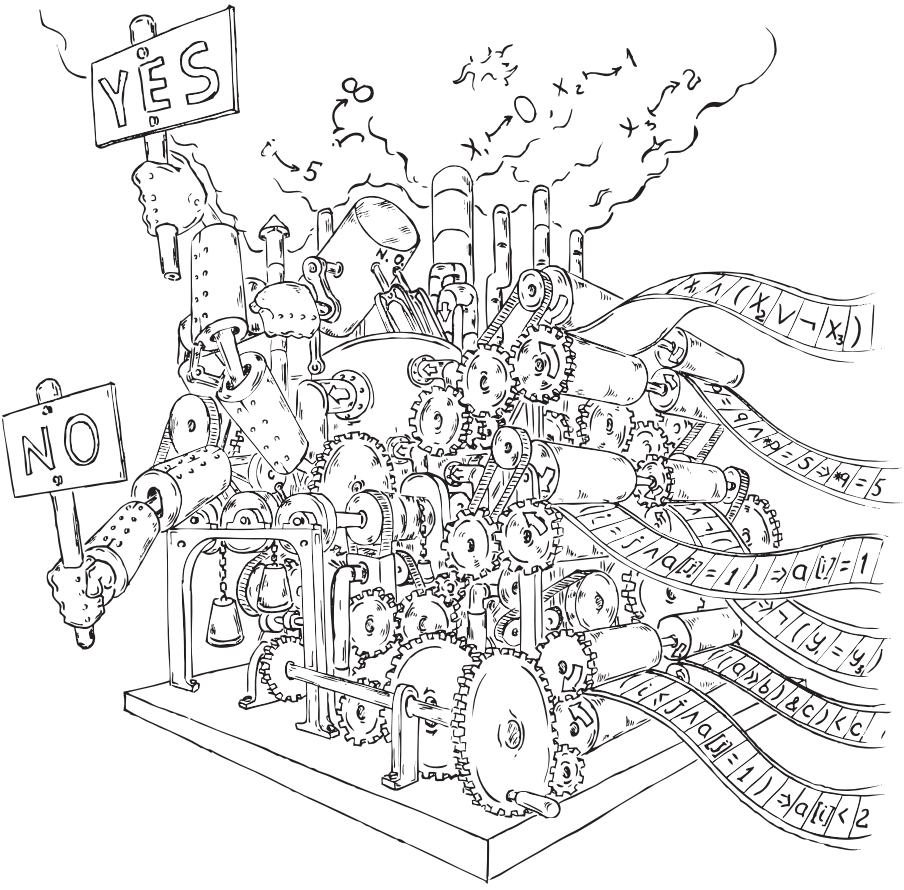

Fig. 1. Decision procedures can be rather complex ... those that we consider in this book take formulas of different theories as input, possibly mix them (using the Nelson–Oppen procedure—see Chap. 10), decide their satisfiability ("YES" or "NO"), and, if yes, provide a satisfying assignment

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