Logic in Computer Science: Modelling and Reasoning about Systems

MICHAEL HUTH
Department of Computing and Information Sciences
Kansas State University, USA.

MARK RYAN
School of Computer Science
University of Birmingham, UK.

Contents

Preface

Our motivation for writing this book

Recent years have brought about the development of powerful tools for verifying specifications of hardware and software systems. By now, the IT industry has realized the impact and importance of such tools in their own design and implementation processes. Major companies, such as Intel, Siemens, BT, AT&T, and IBM, are now actively investigating this technology and its incorporation into their planning and production departments. This necessitates the availability of a basic formal training which allows *undergraduate* students as well as working programmers and beginning graduate students to gain sufficient proficiency in using and reasoning with such frameworks.

The recent shift of information technologies toward internet-based data access and processing means that there is also an increased demand in qualified individuals who can reason about sophisticated autonomous agent-based software which is able to interact with other agents and gather desired information on large networks.

This book addresses these needs by providing a sound basis in logic, followed by an introduction to the logical frameworks which are used in modelling and reasoning about computer systems. It provides simple and clear presentation of material. A carefully chosen core of essential terminology is introduced; further technicalities are introduced only where they are required by the applications.

We believe that our proposed course material makes a vital contribution to preparing undergraduate students for today's fast paced and changeable professional environments. This confidence stems not only from the topicality of our proposed applications, but also from the conviction that a solid background in logical structures and formalisms can very well serve as a buoy in the rough waters of future software and hardware developments.

There is an abundance of books on mathematical logic or logic in computer science on the market. However, we are not aware of any book that suits the contemporary and applications-driven courses that we teach and are beginning to be taught in most computer science curricula. Existing books tend to be written for logicians rather than computer science students and are thus too "heavy", and overloaded with technical terminology. The ties to computer science are merely of a foundational nature, such as the Curry-Howard isomorphism, or cut-elimination in sequent calculi. There is an evident need for a book which introduces the contemporary applications of logic which are beginning to be taken up by industry; the book should be accessible to students who do not want to learn logic for its own sake.

It is important to say what the book does not provide: we completely omitted applications like the design and use of theorem-provers, and the exposure to constructive type theories (such as the Calculus of Constructions and the Logical Framework) as a mathematical foundation for program synthesis; and the design, analysis and implementation of programming languages. This decision is by no means meant to represent a judgment of such topics. Indeed, we hope and anticipate that others will address these important issues in a text that is suitable for undergraduates.

Reasons for adopting this book

Our book zooms in on concepts at the heart of logic and presents them in a contemporary fashion. In that way, and by discussing the implementation of such principles, our material creates stimulating overlaps with other standard courses such as *Formal Language Theory* or an *Introduction to Data Types and Programming*.

It differs from existing books on that subject in the following ways:

- New technical concepts are introduced as they are needed, and never for their own sake. The emphasis is always on applications rather than on mathematical technicalities. Yet, technicalities are always treated with the necessary rigour.
- We introduce, at an accessible level, a framework for program verification (symbolic model checking) which is currently available only in research papers. This is at present a hot topic in industry, and graduates fluent in this material are highly sought.
- Our text is supplemented by a wordwide web site[1] which offers additional

[1] `www.cs.bham.ac.uk/research/lics/`

material useful for classroom presentations; such as postscript files of figures for online, or overhead projector, presentations; and html files of all the SMV code featured in the book.

- All sections of the book have several exercises marked with an $*$ as in

EXERCISES 0.1

* 1. ...
2. ...
* 3. ...
4. ...

---

for which we have provided sample solutions in LaTeX. Bona fide teachers and instructors may obtain the postscript files directly from Cambridge University Press. Exercises end with a short bar, as shown, in order that the reader know where to pick up the text.

Outline of the book

One of the leitmotifs of our book is the observation that most logics used in the design, specification, and verification of computer systems fundamentally deal with a *satisfaction relation*

$$\mathcal{M} \models \phi$$

where $\mathcal{M}$ is some sort of *situation* or *model*, like the snapshot of a system, and ϕ is a specification, a formula of that logic, expressing what should be true in situation $\mathcal{M}$. For example, $\mathcal{M}$ could model a communications protocol and ϕ the property that the protocol be fair. At the heart of this setup is that $\models$ is actually computable in a compositional way. Fixing the situation $\mathcal{M}$, we determine whether $\mathcal{M} \models \phi$ holds by recursively determining this for all subformulas of ϕ. We expand this view for a particular logic (CTL), where this computation and the modelling of a situation may be done purely symbolically using boolean formulas. Tools which support this reasoning make the approach applicable to quite a few realistic systems and designs.

Here is a brief synopsis of what the book covers:

Chapter 1, on propositional logic, should be the common starting point and backbone for any course based on this book; it might also be used as a reference text for courses that presuppose a knowledge of propositional logic. Its sections provide

- a complete presentation of a natural deduction style proof system for propositional logic with a discussion of the intuitionistic fragment;
- a section on propositional logic as a formal language;
- the semantics of propositional logic, where:
  - we constructively prove soundness *and* completeness of the proof system with respect to the usual truth table semantics;
  - we discuss the notions of equivalence, satisfiability, and validity;
  - we cover the principle of mathematical induction, which is needed for soundness, often employed in our book, and is one of the central reasoning tools in computer science; and
  - we feature a section on disjunctive normal forms and Horn formulas; our presentation highlights the development of algorithms computing such normal forms as well as discussing their correctness.

Chapter 2 addresses predicate logic. In this chapter we

- first motivate the need for richer logics via symbolic representations of natural language sentences;
- define and study predicate logic as a formal language with the standard notions of static scoping (free and bound variables) and substitution;
- familiarize students with its semantics,
- introduce a natural deduction style proof system for predicate logic by "enriching" the proof system of Chapter 1 with the introduction and elimination rules for quantifiers; we use this system to prove the standard quantifier equivalences; and
- present Church's proof of the undecidability of satisfaction in predicate logic (via reduction to the Post correspondence problem).

Chapter 3 introduces students to *model checking*, a state-of-the-art technique in verifying concurrent systems. We

- focus on the syntax and semantics of CTL (Computation Tree Logic), and derive the standard algorithm for model checking CTL formulas;
- let students practice the synthesis and interpretation of practically relevant, and frequently occurring, specifications in CTL;
- present two case studies in great detail: a mutual exclusion protocol and an alternating bit protocol; both protocols are developed as labelled transition systems;

- introduce the symbolic model verifier SMV, and provide SMV code for our case studies and discuss the relevant CTL specifications;
- explain how CTL and SMV manage to incorporate fairness constraints;
- discuss the logics LTL and CTL* and compare their expressive power to that of CTL;
- give a fixed-point characterization of those CTL operators which express invariant behaviour; and
- conclude by pointing out that practical specifications often obey common patterns and offer pointers to web-sites, where such patterns are developed, surveyed, and documented.

Chapter 4 covers program verification by discussing deductive reasoning about imperative programs; it presents a Floyd-Hoare style program logic for a sequential imperative core programming language, reminiscent to a fragment of the `C` programming language. The emphasis will be on correctness proofs (partial and total correctness) for fairly simple programs. The main objective is to challenge students to systematically develop small programs meeting required input/output behaviour. In particular, they need to develop the ability to come up with characterizing invariants of while-loops.

Chapter 5 discusses modal logics and agents. Modal logics are motivated through a desire to have possible world semantics.

- We discuss general syntax, semantics, and an extension of the propositional logic deduction calculus for basic modal logic. The theme of the first part of this chapter is that of "logic engineering": e.g. if $\Box\phi$ means that an agent *knows* ϕ, then what axioms and inference rules for $\Box$ should we engineer? We carry out such a task for various meanings of $\Box$.
- The second part of this chapter is devoted to the study of a modal logic modelling general reasoning about knowledge in a Multi-Agent System ($\mathtt{KT45}^n$). It carefully explains how some epistemological puzzles can be solved using this modal logic.

Chapter 6 introduces binary decision diagrams, which are a data structure for boolean functions.

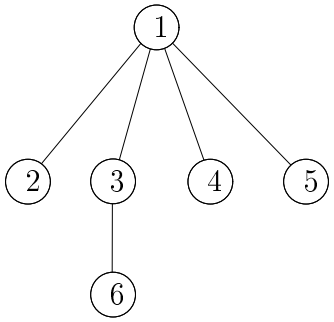
- We describe ordered binary decision diagrams (OBDDs) and their accompanying algorithms;
- We discuss extensions and variations of OBDDs as well as their limitations;
- We explain how CTL models can be coded as boolean formulas;

- We present the syntax and semantics of the relational mu-calculus within which we code CTL models and their specification in the presence of simple fairness constraints.

  That chapter should create stimulating links to courses on algorithms and data structures, courses on circuit design, and can also be used as a foundation for implementation projects which develop tools supporting reasoning with the concepts developed in other chapters.

At the end of each chapter, we provide pointers to the literature and to sites where free software may be downloaded, if applicable. A detailed index should allow for the quick discovery of cross-connections between most of these chapters.

Dependency of chapters and prerequisites. The book requires that students know the basics of elementary arithmetic and naive set theoretic concepts and notation. The core material of Chapter 1 (everything except Sections 1.4.3 to 1.5.3) is essential for all of the chapters that follow. Other than that, only Chapter 6 depends on Chapter 3, and a basic understanding of the static scoping rules covered in Chapter 2 — although one may easily cover Sections 6.1 and 6.2 without having done Chapter 3 at all. The dependency graph of chapters can be seen below:



Suggested course outlines

We suggest at least three different ways of teaching with this text (based on a 12-15 week course).

- A course based on Chapters 1, 2, and 5 would be suitable for students specialising in database and information systems or artificial intelligence; this choice of material should prepare them for more advanced topics in database programming and automated deduction.

- A course based on Chapters 1, 3, and 6 would focus on the complete development of a verification framework for concurrent systems down to the implementation level.
- A course based on Chapters 1, 3, and 4 would provide a broader presentation of the logical foundations of programming.

Suitable courses based on the book. This book can be used as the main text book in a course on the introduction to logic in computer science, and the specification and verification of computer systems and programs. It may be quite useful as an additional text in courses on algorithms and data structures, the introduction to logic in artificial intelligence, sequential circuit and chip design and validation, discrete mathematics for computer scientists, formal language theory, as well as courses in networks and operating systems.

WWW page

This book is supported by a WWW page, which contains a list of errata, the SMV source code for examples in Chapter 3, some further exercises, and details of how to obtain the solutions to exercises in this book which are marked with a $*$. There are also links to other relevant pages. The URL for the book's page is

`www.cs.bham.ac.uk/research/lics/`

Acknowledgments

Many people have, directly or indirectly, assisted us in writing this book. David Schmidt kindly provided serveral exercises for Chapter 4. Krysia Broda has pointed out some typographical errors, and she and the other authors of [BEKV94] have allowed us to use some exercises from that book (notably Exercises 1.6(1(b)), 2.2(5), 2.5(4, 9, 10)). We also borrowed exercises or examples from [Hod77] and [FHMV95]. Zena Matilde Ariola, Josh Hodas, Jan Komorowski, Sergey Kotov, Scott A. Smolka and Steve Vickers have corresponded with us about this text; their comments are appreciated. Matt Dwyer and John Hatcliff made useful comments on drafts of Chapter 3. A number of people read and provided useful comments on several chapters, including: Graham Clark, Christian Haack, Anthony Hook, Achim Jung, Kevin Lucas, Roberto Segala, Alan Sexton, and Allen Stoughton. Numerous students at Kansas State University and the University of Birmingham have given us feedback of various kinds, which have influenced our choice and presentation of the topics. We acknowledge Paul Taylor's LaTeX package for proof boxes. About half a dozen anonymous referees made critical, but constructive comments which helped to improve this text in various ways. In spite of these contributions, there may still be errors in the book, and we alone must take responsibility for those.

Index

Bibliography

[Ake78] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, 1978.

[AO91] K. R. Apt and E.-R. Olderog. *Verification of Sequential and Concurrent Programs*. Springer-Verlag, 1991.

[Bac86] R. C. Backhouse. *Program Construction and Verification*. Prentice Hall, 1986.

[BCM+90] J. R. Burch, J. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1990.

[BEKV94] K. Broda, S. Eisenbach, H. Khoshnevisan, and S. Vickers. *Reasoned Programming*. Prentice Hall, 1994.

[BJ80] G. Boolos and R. Jeffrey. *Computability and Logic*. Cambridge University Press, 2nd edition, 1980.

[Boo54] George Boole. *An Investigation of the Laws of Thought*. Dover, New York, NY, USA, 1854.

[Bra91] J. C. Bradfield. *Verifying Temporal Properties of Systems*. Birkhaeuser, Boston, Mass., 1991.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Compilers*, C-35(8), 1986.

[Bry91] R. E. Bryant. On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Applications to Integer Multiplication. *IEEE Transactions on Computers*, 40(2):205–213, February 1991.

[Bry92] R. E. Bryant. Symbolic Boolean Manipulation with Ordered Binary-decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[CE81] E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In D. Kozen, editor, *Logic of Programs Workshop*, number 131 in LNCS. Springer Verlag, 1981.

[CGL93] E. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In *A Decade of Concurrency*, number 803 in Lecture Notes in Computer Science, pages 124–175. Springer Verlag, 1993.

[CGL94] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and Abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, September 1994.

[Che80] B. F. Chellas. *Modal Logic – an Introduction*. Cambridge University Press, 1980.

[Dam96] D. R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. PhD thesis, Institute for Programming research and Algorithmics. Eindhoven University of Technology, July 1996.

[Dij76] E. W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.

[DP96] R. Davies and F. Pfenning. A Modal Analysis of Staged Computation. In *23rd Annual ACM Symposium on Principles of Programming Languages*. ACM Press, January 1996.

[EN94] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Benjamin/Cummings, 1994.

[FHMV95] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, 1995.

[Fit93] M. Fitting. Basic modal logic. In D. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1. Oxford University Press, 1993.

[Fit96] M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, 2nd edition, 1996.

[Fra92] N. Francez. *Program Verification*. Addison-Wesley, 1992.

[Fre03] G. Frege. *Grundgesetze der Arithmetik, begriffsschriftlich abgeleitet*. 1903. Vol. I and II (Jena).

[Gal87] J. H. Gallier. *Logic for Computer Science*. John Wiley, 1987.

[Gen69] G. Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, chapter 3, pages 68–129. North-Holland Publishing Company, 1969.

[Gol87] R. Goldblatt. *Logics of Time and Computation*. CSLI Lecture Notes, 1987.

[Gri82] D. Gries. A note on a standard strategy for developing loop invariants and loops. *Science of Computer Programming*, 2:207–214, 1982.

[Ham78] A. G. Hamilton. *Logic for Mathematicians*. Cambridge University Press, 1978.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–580, 1969.

[Hod77] W. Hodges. *Logic*. Penguin Books, 1977.

[Hod83] W. Hodges. Elementary predicate logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 1. Dordrecht: D. Reidel, 1983.

[Hol90] G. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1990.

[Koz83] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[Lee59] C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38:985–999, 1959.

[Lon83] D. E. Long. *Model Checking, Abstraction, and Compositional Verification*. PhD thesis, School of Computer Science, Carnegie Mellon University, July 1983.

[McM93] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1991.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, 1995.

[MvdH95] J.-J. Ch. Meyer and W. van der Hoek. *Epistemic Logic for AI and Computer Science*, volume 41 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1995.

[Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[Pau91] L.C. Paulson. *ML for the Working Programmer*. Cambridge University Press, 1991.

[Pnu81] A. Pnueli. A temporal logic of programs. *Theoretical Computer Science*, 13:45–60, 1981.

[Pop94] S. Popkorn. *First Steps in Modal Logic*. Cambridge University Press, 1994.

[Pra65] D. Prawitz. *Natural Deduction: A Proof-Theoretical Study*. Almqvist & Wiksell, 1965.

[QS81] J. P. Quielle and J. Sifakis. Specification and verification of concurrent systems in cesar. In *Proceedings of the fifth International Symposium on Programming*, 1981.

[Ros97] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1997.

[SA91] V. Sperschneider and G. Antoniou. *Logic, A Foundation for Computer Science*. Addison Wesley, 1991.

[Sch92] U. Schoening. *Logik Für Informatiker*. B.I. Wissenschaftsverlag, 1992.

[Sch94] D. A. Schmidt. *The Structure of Typed Programming Languages*. Foundations of Computing. The MIT Press, 1994.

[Sim94] A. K. Simpson. *The Proof Theory and Semantics of Intuitionistic Modal Logic*. PhD thesis, The University of Edinburgh, Department of Computer Science, 1994.

[Tay98] R. G. Taylor. *Models Of Computation and Formal Languages*. Oxford University Press, 1998.

[Ten91] R. D. Tennent. *Semantics of Programming Languages*. Prentice Hall, 1991.

[Tur91] R. Turner. *Constructive Foundations for Functional Languages*. McGraw Hill, 1991.

[vD89] D. van Dalen. *Logic and Structure*. Universitext. Springer-Verlag, 3rd edition, 1989.

[Wei98] M. A. Weiss. *Data Structures and Problem Solving Using Java*. Addison-Wesley, 1998.