Implementation of propositional temporal logics using BDDs

Citation for published version (APA):

DOI:
10.1007/3-540-69778-0_11

Document status and date:
Published: 01/01/1998

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication
Implementation of Propositional Temporal Logics Using BDDs

G.L.J.M. Janssen
Eindhoven University of Technology
Department of Electrical Engineering, Room EH 9.26
P.O. Box 513, 5600 MB Eindhoven
E-mail: geert@ics.ele.tue.nl

1 Topic/Relevance
This tutorial intends to convey the step-by-step process by which computer programs for model checking and satisfiability testing for temporal logics may be derived from the theory. The idea is to demonstrate that it is very well possible to implement such a program in an efficient way without sacrificing a correct-by-construction approach. The tutorial will be fully self-contained, only a general knowledge of programming and propositional logic is assumed.

The proposed tutorial focuses on what might be coined “Implementing the theories”. Unfortunately, too often an interesting approach or novel algorithm once published is not picked up by any user community. This is partly because no effort is spent in creating a state-of-the-art implementation in a readily accessible form. The intended audience are researchers in the field of theorem proving and any users of applications thereof. They will benefit from learning how theoretical results can be implemented in a well-written prototype program. Moreover, being a prototype should not be an excuse for not using advanced data structures and algorithms. Therefore this tutorial takes an engineering approach to dealing with complicated issues such as reasoning in temporal logic, and shows that with a structured approach a powerful tool can be built that is able to handle real-life applications, for instance verification of sequential circuits.

2 Contents Outline
Below an outline is given of the tutorial contents. Much of the material on CTL model checking is based on the pioneering work in this area by Clarke et al.

1. Introduction. Presents the tutorial contents and sets its goals.
2. Dags. Directed acyclic graphs are the datastructure underlying BDDs. Dags will also be used to represent formulas. An efficient implementation based on a hash table and utilizing garbage-collection will be discussed.
3. BDDs. Explains what Binary Decision Diagrams are, how they can be efficiently implemented, and what their applications are.
4. Kripke Structure Model. Introduces the notion of a Kripke structure that is used as a model for the system we like to reason about.
5. **Computation Tree Logic.** Defines the syntax and semantics of a class of branching time logics.

6. **CTL Model Checking.** Shows how a model checker for CTL can be constructed in elegant way using a simple algorithm to calculate a fixed-point of a functional. Also, it is shown how BDDs can be exploited to represent the next-state relation and state-sets of the Kripke structure.

7. **Linear-time Temporal Logic.** Defines the syntax and semantics of the popular Manna/Pnueli propositional linear-time temporal logic.

8. **PTL Satisfiability Checking.** Shows how a satisfiability checker can be constructed for PTL. Again, BDDs will be used to represent the structure of the generated tableau.

9. **Diversion into $\mu$-calculus.** Briefly explains Kozen’s $\mu$-calculus and shows how both CTL and PTL problems can be encoded in it. An implementation of $\mu$-calculus is sketched.

10. **Summary and conclusions.** Summarizes the presented material and demonstrates the derived programs by some example runs and lists results of experiments.

The bibliography lists a number of articles and books on which the tutorial material will be based.

**References**


