BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2023
http://excel.fit.vutbr.cz

New Techniques for Compact Representation of Boolean Functions

Ján Maťufka

Abstract
Binary decision diagrams (BDDs) represent Boolean functions and are extensively used in formal verification, model checking, circuit synthesis in CAD software, etc. With more variables, the BDD size grows in the worst case exponentially, and that poses a significant concern in terms of space complexity. The aim of this paper is to create an automata-based model for compact representation of BDDs. To achieve this, tree automata are used. By inserting small tree automata with specific properties (boxes) into the BDD structure, one can use their capacity for looping to cover larger repeating patterns in the BDD structure. This model allows for approx. 10-20 % smaller node counts in tested benchmarks (in most cases) compared to the state-of-the-art models (BDDs, TBDDs, ESRBDDs). Using a tree-automata based approach allows for creating custom tree automata suited for reducing specific patterns and thus allowing for possibly even better space complexity reduction properties.

*xmatuf00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

The main aim of this project is to develop an automata-based framework that has potential to reduce state space of binary decision diagrams. This approach works using tree automata [1], so it will be called Automata-based binary decision diagrams (ABDDs). Essentially, by inserting instances of tree automata from some predetermined set (a "catalogue" of tree automata) into the structure of the binary decision diagram, we can utilize the ability of tree automata to loop to effectively remove some repeating patterns. This approach allows for extendability and customizability, by simply introducing custom tree automata into the "catalogue".

Another goal is to evaluate the effectiveness of this model and compare this solution to existing edge-based reduction models. ABDDs will be copmared to binary decision diagrams (BDDs) [2, 3], zero-reduced BDDs (ZBDDs) [4], tagged BDDs (TBDDs) [5], chain-reduced BDDs (CBDDs and CZDDs) [6] and BDDs with edge-specified reductions (ESRBDDs) [7]. All of these models (and the set of reduction rules they use) can also be simulated using ABDDs with some restrictions on the "catalogue" of tree automata used for state-space reductions.

2. ABDD intricacies

A *(finite) tree automaton* is a tuple $\mathcal{A} = \langle Q, \Sigma, \Delta, \mathcal{R} \rangle$ where Q is a finite set of *states*, Σ is a ranked alphabet, $\Delta \subseteq Q \times \Sigma \times Q^*$ is a set of transitions of the form $q \to \{a_{/n}\} \to (q_1, \ldots, q_n)$ (where n denotes arity of symbol a – i.e. the amount of states the edge leads to), and $\mathcal{R} \subseteq Q$ is the set of *root states*. Specifically, transitions of the form $q \to \{a_{/0}\} \to ()$ are called *leaf* transitions. In order to properly utilize finite tree automata in the context of this work, additional information is stored in the symbols from the ranked alphabet.

This model modifies/expands on the definition of a finite tree automaton. It only utilizes three transition symbols of two differing arities (0 for leaf transitions – symbols 0 and 1, 2 for non-leaf transitions – symbol LH). Additionally, each edge can contain information about the variable of the source state. This intuitively means that during a run, state can see a variable and after performing the transition this variable is not encountered again during the run of an ABDD. Apart from a variable, a transition can contain information about reductions in form of tree automata names (or keys to a "catalogue" of used automata). Transitions with an arity of 2 can have box "labels" on both the

short and high parts of the edge (symbol LH is a short to "low-high"). The target states of this transition (or children states) that are then mapped to the output ports of the boxes that label a part of the edge corresponding to the box. And so, the transitions in the ABDD store this information: source state, transition symbol (LH, 0, 1), variable, and in case of non-leaf (LH) transitions, boxes used on each part of the edge and target states (children). Note that if the edge contains boxes with arities higher than 1, the arity of the edge symbol may not correspond with total amount of children.

2.1 Boxes used

This project uses 7 simple tree automata, for which a few restrictions have to apply. These include non-empty language, trimness, one root state, non-vacuity or non-obsoleteness, port-consistency, port-uniqueness, unambiguity with regards to variable assignment to rootstate and output ports. Note that these automata have not yet been instanciated with variables (this happens during folding process). These automata will be called "boxes" in the context of this project (as they "store" or "fold" patterns in the decision diagram). The boxes were designed in such a way so that they would simulate already known reduction rules in other models (see figure 1).

2.2 Obtaining canonical form

There are three operations that are necessary in order to obtain a canonical form of an ABDD.

Unfolding removes boxes from edges and replaces them with the corresponding tree automaton with the correct port-state mapping (see figure 2).

Normalization is a bottom-up determinization with regards to the variables and saturation of the structure with variables where it is possible to compute them (ie. normalization remove equivalent nodes) .

Folding replaces repeating patterns (as much as possible) with the boxes in a given box order. The folding procedure works by creating constructs similar to intersections of tree automata, but they behave differently wrt. the port transitions of the box.

Another important operations apart from these are satisfiability testing and applying boolean operators. These functions currently do not work with the structure that has box reductions already applied and firstly require unfolding. Applying boolean operators over the ABDD works by unfolding the boxes, unwinding the cycles and then recursively calling apply on the operands of the structure. The result then can be normalized and folded back to obtain a canonical result in the form of an ABDD.

3. Results

This framework was tested on a set of combinational circuit benchmarks from LGSynth'91 [8]. They were first tranfsormed into BDDs (with the help of BuDDy [1]) and then unfolded, normalized, saturated with variables and then folded with the corresponding box order. ABDDs were compared to other models: BDDs, ZBDDs, TBDDs, ESRBDDs, CZDDs, CBDDs. The results show that the ABDD model with 7 boxes achieved the best results in terms of compactness. On average, ABDDs had smaller node counts than all other tested models (from 8.6 % compared to CBDDs up to 21 % compared to ZBDDs), as can be seen on the graphs (see figure 3). ABDDs used mostly $L_{\oplus}$ boxes, that were introduced in this thesis, while box H_0 has not been used at all, which was suggested in [7] (see figure 4). The usage is, however, strongly dependant on the box order in which the folding is applied.

4. Conclusions and future work

This project tried to develop a framework that could generalize various binary decision diagram models into one, extensible framework. It achieved this by using the properties of finite tree automata. The results show the potential of ABDDs not only as a tool of unifying all decision diagram models under one framework, but also its potential to achieve better reduction results as currently used models.

4.1 Future work

Of course, this work is only in its early stages, and so there are a couple of things that can and should be improved.

- better memory-management during bottom-up algorithms (reachability, normalization),
- faster folding process,
- applying boolean operations and testing satisfiability without the need for unfolding.

Other than that, there are multiple possibilities to build upon and improve this work. Identifiying more patterns that can be reduced with the corresponding boxes that can reduce them. Extending it to work with multi-terminal BDDs (MTBDDs).

[1] https://github.com/SSoelvsten/buddy

5. Acknowledgements

I would like to thank my supervisor Ing. Ondřej Lengál, Ph.D. his guidance, suggestions, patience and help while working on this project and my consultant Ing. Vojtěch Havlena, Ph.D. for his suggestions and ideas.

References

[1] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. http://www.grappa.univ-lille3.fr/tata, 2007. release October, 12th 2007.

[2] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[3] Randal E. Bryant. Binary decision diagrams. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 191–217, Cham, 2018. Springer International Publishing.

[4] Shin-ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In Alfred E. Dunlop, editor, *Proceedings of the 30th Design Automation Conference. Dallas, Texas, USA, June 14-18, 1993*, pages 272–277. ACM Press, 1993.

[5] T. van Dijk, R. Wille, and R. Meolic. Tagged BDDs: Combining reduction rules from different decision diagram types. In *2017 Formal Methods in Computer Aided Design (FMCAD)*, pages 108–115. IEEE, 2017.

[6] Randal E. Bryant. Chain reduction for binary and zero-suppressed decision diagrams. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I*, volume 10805 of *Lecture Notes in Computer Science*, pages 81–98. Springer, 2018.

[7] Junaid Babar, Chuan Jiang, Gianfranco Ciardo, and Andrew S. Miner. Binary decision diagrams with edge-specified reductions. In Tomáš Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II*, volume 11428 of *Lecture Notes in Computer Science*, pages 303–318. Springer, 2019.

[8] S. Yang. Logic synthesis and optimization benchmarks user guide version 3.0. 1991.