Algorithm Selection for Word-Level Hardware Model Checking

Zhengyang Lu[1], Po-Chun Chien[2], Nian-Ze Lee[2], Vijay Ganesh[3]

[1]University of Waterloo, [2]LMU Munich, [3]Georgia Institute of Technology
200 University Ave W, Waterloo, ON N2L 3G1
z52lu@uwaterloo.ca; +1 519-888-4567

Abstract

We build the first machine-learning-based algorithm selection tool for hardware verification described in the BTOR2 format. In addition to hardware verifiers, our tool also selects from a set of software verifiers to solve a given BTOR2 instance, enabled by a BTOR2-to-C translator. We propose two embeddings for a BTOR2 instance, Bag of Keywords and Bit-Width Aggregation. Pairwise classifiers are applied for algorithm selection. Upon evaluation, our tool BTOR2-SELECT solves 30.0% more instances and reduces PAR-2 by 50.2%, compared to the PDR implementation in the HWMCC'20 winner model checker AVR. Measured by the Shapley values, the software verifiers collectively contributed 27.2% to BTOR2-SELECT's performance.

Introduction

It has long been observed that for computationally hard problems, no single algorithm performs well on all instances, and different algorithms perform well on distinct classes of instances. This observation is consistent with the widely-believed conjecture that $P \neq NP$. To leverage such complementary strengths, machine learning (ML)-based algorithm selection techniques are gaining popularity (Xu et al. 2012). Algorithm selection aims to, for each given instance, select the optimal algorithm from a set of candidates. Algorithm selectors are usually ML models, trained using historical performance data, to predict algorithms' performance given some cheaply computable features of each input instance.

Our research focuses on algorithm selection for the hardware model-checking problem in the BTOR2 format (Niemetz et al. 2018). Model checking plays a crucial role in ensuring the correctness and reliability of critical hardware systems, as even minor errors in hardware designs can lead to catastrophic failures (Clarke 1997). Traditionally, hardware model checkers such as ABC (Brayton and Mishchenko 2010) have been developed to verify the correctness of hardware designs against given specifications. What makes this problem more interesting is the recent development of BTOR2C (Beyer, Chien, and Lee 2023), a BTOR2-to-C translator, which allows software verifiers to be applied to hardware verification tasks as well.

We propose, to the best of our knowledge, the first ML-based algorithm selection tool for hardware verification. Our tool, called BTOR2-SELECT, leverages the strength of both hardware and software verifiers. Trained and evaluated on a comprehensive set of 1 441 BTOR2 benchmarks, our tool effectively closed up to 66.0% of the performance gap between the single best solver (SBS) and the virtual best solver (VBS). The software verifiers collectively contributed 27.2% to this performance, measured by the Shapley values (Shapley 1953). Our code and data are available at: https://gitlab.com/sosy-lab/software/btor2-select.

Background

The BTOR2 Language: BTOR2 is a word-level model-checking format for sequential circuits. The modeling-checking problem decides whether a safety property holds on all executions of a circuit.

Cost-sensitive Pairwise Classifier (PWC): A PWC (Xu et al. 2012) is an ML classifier that given a pair of algorithms, predicts which one would perform better on a particular instance. For algorithm selection, a PWC is trained for every pair of candidate algorithms, using their performance data over a training instance set. Each training sample is labeled to indicate which solver in the pair performs better on a specific training instance, with sample weights reflecting the performance difference. At inference time, for a given instance, all PWCs are evaluated, and the algorithm receiving the highest votes is selected to solve this instance.

Portfolio Contribution: Fréchette et al. (2016) proposed to evaluate an algorithm's contribution to a portfolio using *Shapley values*. The Shapley value, originating from cooperative game theory, is widely regarded as a fair measure of individual components' contribution to a coalition's performance.

Algorithm Selection for BTOR2

Instance Representation We propose two types of BTOR2 instance representation: *Bag of Keywords* (BoKW) and *Bit-Width Aggregation* (BWA). For each instance, BoKW counts the occurrence of each *keyword* from a predefined set of 69 BTOR2 keywords, such as `state`, `not`, and `add`, as the instance representation. Most keywords return a variable of a certain bit-width. BWA, instead of counting the

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

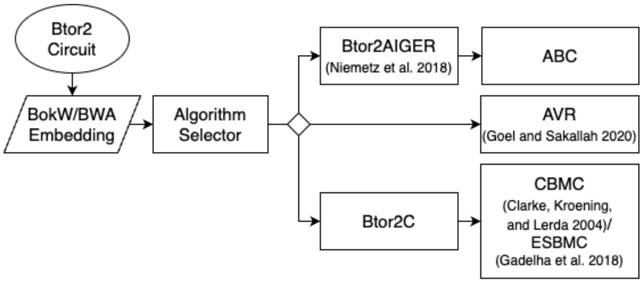

Figure 1: Btor2-Select architecture

occurrence, sums the bit-widths of all returned variables for each relevant keyword.

Algorithm Selection: We use PWC for algorithm selection, with XGBoost models (Chen and Guestrin 2016) as the underlying classifiers due to their inference efficiency and relative robustness to high-dimensional, correlated features. The training samples are weighted by the difference in PAR-2. PAR-2 is a scoring measure that counts the actual used time for successful instances, while penalizing double the timeout for failed instances.

Architecture: Given a Btor2 instance, Btor2-Select selects and applies the expected best verifier from a predefined set. We include four model checkers, each with multiple configurations in the verifier set. Two hardware model checkers `AVR` and `ABC` are the winners of the Hardware Model Checking Competition (HWMCC) 2020; [1] software verifiers `CBMC` and `ESBMC` showed strong performance on Btor2 instances in our previous Btor2C study. The Btor2-Select architecture is shown in Figure 1.

Experimental Results

We have built a comprehensive Btor2 benchmark set.[2] At this stage, we only focus on the benchmarks without arrays. All 1 441 such benchmarks were randomly divided into a training set and a testing set with an 80-20 split, where our tool Btor2-Select was trained and tested accordingly. The PWC models were trained on a MacBook Air with an M2 chip and 8 GB memory. All verifier-instance pairs were executed on Ubuntu 22.04 machines, each with a 3.4 GHz CPU (Intel Xeon E3-1230 v5) with 8 processing units and 33 GB of RAM. Each task was assigned 2 CPU cores, 15 GB RAM, and 15 min of CPU time limit.

Table 1 shows the evaluation results over the 288-instance testing set. Btor2-Select effectively closed around 65% of the VBS-SBS gap. The PWC selector with the BoKW embedding solved 12.9% more instances and reduced PAR-2 by 27.7% compared to `ABC.PDR`, the SBS. We also evaluated each component verifier's contribution by the *Shapley value*, as results shown in Figure 2. The performance was measured by the solved instance numbers. Collectively, the software components contributed 27.2% to the PWC-BoKW performance.

Verifier	#Solved	PAR-2 (sec)
PWC-BoKW	227	120 484.8
PWC-BWA	226	122 008.9
SBS (`ABC-PDR`)	201	166 683.4
VBS	239	96 629.8

Table 1: Evaluation results of Btor2-Select

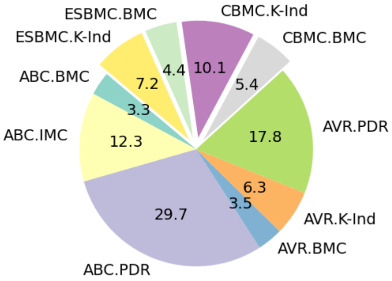

Figure 2: Shapley Value Contribution of Each Component Verifier to PW-BoKW, Expressed as a Percentage (%)

Acknowledgment

This project was funded by the Google Summer of Code program and the Deutsche Forschungsgemeinschaft (DFG) - 536040111 (Bridge). The first author thanks Arie Gurfinkel, Cedric Richter, and Chris Cameron for guidance and help.

References

Beyer, D.; Chien, P.-C.; and Lee, N.-Z. 2023. Bridging hardware and software analysis with Btor2C: A word-level-circuit-to-C translator. In *TACAS 2023*, 152–172.

Brayton, R.; and Mishchenko, A. 2010. ABC: An academic industrial-strength verification tool. In *CAV 2010*, 24–40.

Chen, T.; and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In *ACM SIGKDD 2016*, 785–794.

Clarke, E.; Kroening, D.; and Lerda, F. 2004. A tool for checking ANSI-C programs. In *TACAS 2004*, 168–176.

Clarke, E. M. 1997. Model checking. In *FSTTCS 1997*, 54–56.

Fréchette, A.; Kotthoff, L.; Michalak, T.; Rahwan, T.; Hoos, H.; and Leyton-Brown, K. 2016. Using the Shapley value to analyze algorithm portfolios. In *AAAI 2016*, volume 30.

Gadelha, M. R.; Monteiro, F. R.; Morse, J.; Cordeiro, L. C.; Fischer, B.; and Nicole, D. A. 2018. ESBMC 5.0: an industrial-strength C model checker. In *ACE 2024*, 888–891.

Goel, A.; and Sakallah, K. 2020. AVR: abstractly verifying reachability. In *TACAS 2020*, 413–422.

Niemetz, A.; Preiner, M.; Wolf, C.; and Biere, A. 2018. Btor2, btormc and boolector 3.0. In *CAV 2018*, 587–595.

Shapley, L. S. 1953. A value for n-person games. *Contributions to the Theory of Games*, 2.

Xu, L.; Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2012. Evaluating component solver contributions to portfolio-based algorithm selectors. In *SAT 2012*, 228–241.

[1] https://hwmcc.github.io/2020/

[2] https://gitlab.com/sosy-lab/research/data/word-level-hwmc-benchmarks/