AIGROW: A Feedback-Driven Test Generation Framework for Hardware Model Checkers

Wenjing Deng
Shanghai Key Laboratory of Trustworthy Computing
East China Normal University, China
51215902117@stu.ecnu.edu.cn

***Abstract*—This research abstract introduces an effective and efficient approach to automatically generate high-quality hardware model checker benchmarks. The key contribution of this work is to model the input format of hardware model checkers using a tree-based structure named ARTree and build an effective feedback-driven test generation framework based on ARTree named `AIGROW`. The evaluation shows that `AIGROW` generates very small but high-quality benchmarks for coverage-oriented and performance-oriented testing and outperforms the existing generation-based testing tools.**

***Index Terms*—test generation; hardware model checker**

I. Introduction

Hardware model checking [5], [9], [10], [12] plays a vital role for hardware design and verification. Given a model M abstracted from a hardware design and a property P, hardware model checking is to check whether P holds on M. Many hardware model checking algorithms have been implemented as tools, called hardware model checkers (HMCers). HMCers are expected to find problems or prove the safety in hardware designs as fast as possible. Thus, the developers of HMCers need a large number of high-quality tests to evaluate the correctness and performance of HMCers.

However, except for benchmarks provided by the Hardware Model Checking Competition (HWMCC), no more high-quality tests are available for developers. AIGFUZZ [1] and AIGEN [13] are two existing random test generation tools for HMCers; the tests generated by them are generally very large in size. A mutation-based tool for testing HMCers is Hammer [18], but the quality of the generated test is limited by the initial mutation seeds. High-quality but small tests are still required by the developers. This work targets this problem.

Compared with random test generation and mutation, feedback-driven test generation has shown to be more effective in some situations [11], [14]–[17]. However, feedback-driven generation requires the generated test to be extendable. Hence, the existing random generation tools AIGFUZZ and AIGEN do not fit directly into this approach [1], [13].

The challenge of the problem is how to make hardware designs extendable. To solve this problem, this work proposes a tree-based structure ARTree to describe the hardware designs. Then the hardware designs can be extendable using it. Meanwhile, this paper proposes a feedback-driven test generation framework for HMCers, which is implemented as a tool named `AIGROW`.

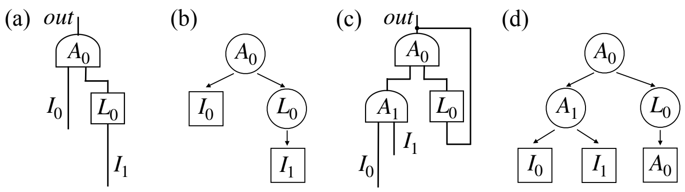


Fig. 1: AIGs and the corresponding ARTrees

This framework uses the feedback from HMCers executions to guide each single-step extension of ARTree. The evaluation shows that `AIGROW` is more efficient than previous tools in generating high-quality but small tests.

II. Approach

The AIGER format [6] is a compact intermediate language and is widely used to describe hardware design, therefore, the approach is demonstrated with this input format. An AIGER file describes a sequential And-Inverter Graph (AIG) built with AND gates and latches. The key idea is that each AIG can be ***transformed*** into an AIG relation tree (ARTree); single-step ***extension*** of an existing ARTree to build a new ARTree; in ***evolution***, following the guidance of feedback, ARTree will be extended in the desired direction.

Transformation is the process of translating an AIG into an ARTree or its reverse. Fig. 1(a) to (b) show an example of the transformation procedure. The output of A_0 is the output of the system; therefore, A_0 is the root node of the ARtree. Since the outputs of L_0 and I_0 are the inputs of A_0, they are the children of A_0 in the ARtree. As I_1 is the input of L_0, I_1 is the child of L_0 in the ARtree. Fig. 1(c) to (d) give a more complex example when there is a cycle in the graph. The output of A_0 is the input of L_0. In this situation, the terminal node labeled as A_0 is the child of L_0, and A_0 cannot be extended further (labeled as a square in the ARtree).

Extension is the process of extending an ARTree into a new one. Extension consists of two stages: the initialization phase and the growth phase. The initialization phase removes the input nodes to make their parent nodes extensible. The growth phase randomly adds new nodes to the extensible nodes. For example, Fig. 1(d) is an extension of Fig. 1(b). The approach first removes I_0 and I_1 and then add node A_1 to node A_0. As A_1 is still an extensible node, A_1 is extended to have two children I_0 and I_1. Extended children can be existing or new

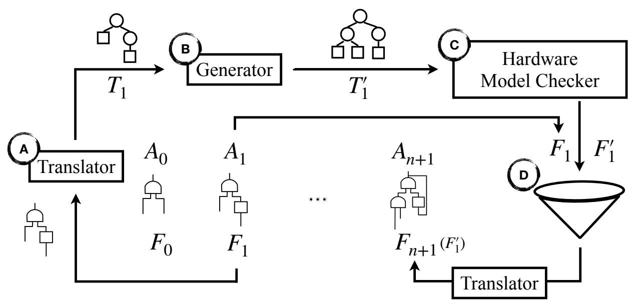


Fig. 2: Workflow of `AIGROW`.

components; for example, node L_0 is extended to have a child node A_0, which is an existing component.

Evolution is to select promising ARTrees and evolve them to generate high-quality tests. Fig. 2 provides an overview of the process, which consists of four steps. Step A selects an AIG from the queue of length n and translates it into an ARTree, assuming that the AIG is A_1 and the ARTree is T_1. Step B extends T_1 to a new ARTree T'_1. Step C transforms T'_1 into an AIG as input to a hardware model checker and obtains the feedback F'_1 (e.g., coverage or solving time). Step D decides whether the extended ARTree should be stored in the queue. If F'_1 is greater than F_1, the new AIG A_{n+1} translated from ARTree T_{n+1} and the corresponding feedback F'_1 are added to the queue. Otherwise, the ARTree is discarded. After sufficient iterations, high-quality tests are in the queue.

III. Evaluation

HMCers PDR [8] in ABC and IC3ref [2] are chosen to evaluate this method because they both implement the state-of-the-art hardware model checking algorithm IC3 [7]. This paper compares `AIGROW` with two existing test generation tools AIGFUZZ [1] and AIGEN [13]. To show the effectiveness of the feedback strategy, a control group is also set to run `AIGROW` without feedback (`AIGROW`-nofeed). The feedback is set to ***coverage*** as the guidance in Fig. 3(a) and ***solving time*** as the guidance in Fig. 3(b), respectively. Each test has a solution time limit of 2 hours.

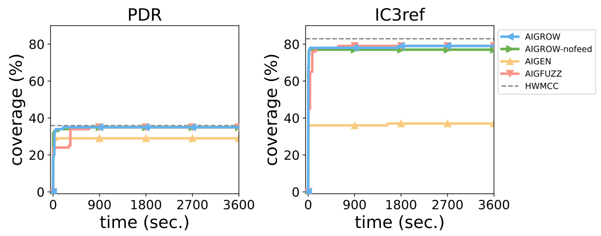
Fig. 3(a) shows the effectiveness of `AIGROW` on coverage-oriented testing and compares the efficiency of generation tools by line coverage. The dashed line is a reference that denotes the line coverage achieved by all the HWMCC benchmarks that can be solved in one hour. For PDR, `AIGROW` peaks faster than other tools with or without coverage guidance. For IC3ref, the growth of `AIGROW` is as fast as AIGFUZZ. Both on PDR and IC3ref, the coverage of `AIGROW` and AIGFUZZ is close to the dashed line within 1 hour; while the efficiency of AIGEN is not as good as desired. The result concludes that ***`AIGROW` can achieve maximum coverage faster than other tools***.

Fig. 3(b) shows the evaluation results of performance-oriented testing. As well as AIGEN, the tests generated by `AIGROW`-nofeed are solved in a short time. From `AIGROW`-nofeed's results, the feedback-driven strategy is necessary when generation is guided by solving time. For PDR, only `AIGROW` can generate hard-to-solve test cases, and a timeout test case is generated in 15,000 s. For IC3ref, `AIGROW` performs better than AIGFUZZ. The results conclude that ***`AIGROW` can generate more difficult test cases than other generation tools with the help of performance guidance.***

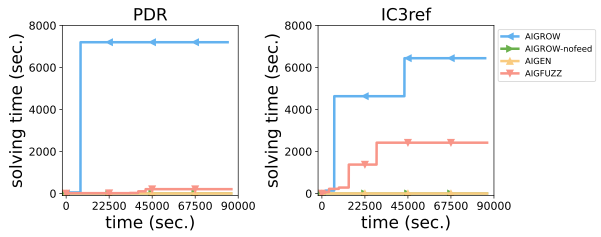
Table. I provides the average size of generated test cases by different generation tools. Size is defined as the sum of the number of latches and the number of AND gates in a test. The results show that the tests generated by `AIGROW` are much smaller than others. The result concludes that ***`AIGROW` generate smaller test cases than other generation-based tools***.

IV. Conclusion

This research abstract introduces ARTree to facilitate extensible hardware design. Based on ARTree, a feedback-driven framework was constructed and implemented as a tool named AIGROW. The evaluation demonstrates that AIGROW is more efficient than other tools in generating high-quality tests. The generated tests are also in small sizes. For future work, it would be interesting to extend the idea of ARTree to generate tests for other hardware design systems, such as Matlab Simulink [3] and Xilinx Vivado [4].



(a) Comparison of coverage. The x-axis denotes the CPU time, and the y-axis means the coverage achieved by all current generated tests.



(b) Efficiency comparison among generation tools. The x-axis denotes the CPU time, and the y-axis means the current max solving time.

Fig. 3: Comparison of different generation tools.

TABLE I: Average size of generated tests.

	AIGROW		AIGFUZZ		AIGEN	
	cov.	*perf.*	*cov.*	*perf.*	*cov.*	*perf.*
PDR	10	109	6,864	6,871	189,190	180,191
IC3ref	15	40	6,785	6,867	180,193	180,194

References

[1] Aiger, http://fmv.jku.at/aiger/

[2] IC3Ref. https://github.com/arbrad/IC3ref

[3] Simulink, https://ww2.mathworks.cn/products/simulink.html

[4] Vivado, https://www.xilinx.com/products/design-tools/vivado.html

[5] Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without bdds. In: TACAS. vol. 1579, pp. 193–207 (1999)

[6] Biere, A., Heljanko, K., Wieringa, S.: Aiger 1.9 and beyond (2011)

[7] Bradley, A.R.: Sat-based model checking without unrolling. In: VMCAI. pp. 70–87. Springer (2011)

[8] Brayton, R., Mishchenko, A.: ABC: An academic industrial-strength verification tool. In: CAV. pp. 24–40 (2010)

[9] Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. TOPLAS **8**(2), 244–263 (1986)

[10] Clarke, E.M., Gupta, A., Jain, H., Veith, H.: Model checking: Back and forth between hardware and software. In: VSTTE. vol. 4171, pp. 251–255 (2005)

[11] Garg, P., Ivančić, F., Balakrishnan, G., Maeda, N., Gupta, A.: Feedback-directed unit test generation for c/c++ using concolic execution. In: ICSE. pp. 132–141 (2013)

[12] Griggio, A., Roveri, M.: Comparing different variants of the ic3 algorithm for hardware model checking. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **35**(6), 1026–1039 (2016)

[13] Jacobs, S., Sakr, M.: Aigen: Random generation of symbolic transition systems. In: CAV (2021)

[14] Jaygarl, H., Lu, K., Chang, C.K.: Genred: A tool for generating and reducing object-oriented test cases. In: COMPSAC. pp. 127–136 (2010)

[15] Pacheco, C., Ernst, M.D.: Eclat: Automatic generation and classification of test inputs. In: ECOOP. vol. 3586, pp. 504–527 (2005)

[16] Pacheco, C., Lahiri, S.K., Ball, T.: Finding errors in. net with feedback-directed random testing. In: ISSTA. pp. 87–96 (2008)

[17] Pacheco, C., Lahiri, S.K., Ernst, M.D., Ball, T.: Feedback-directed random test generation. In: ICSE. pp. 75–84 (2007)

[18] Zhang, C., Sun, M., Li, J., Su, T., Pu, G.: Feedback-guided circuit structure mutation for testing hardware model checkers. In: ICCAD (2021)