Bridging Hardware and Software Formal Verification
(Extended Abstract)

Po-Chun Chien

LMU Munich, Munich, Germany

Abstract. Modern technology relies heavily on the integration of hardware and software systems, from embedded devices in consumer electronics to safety-critical controllers. Despite their interdependence, the tools and methods used for verifying the correctness and reliability of these systems are often segregated, meaning that the advancement in one community cannot benefit another directly. Addressing this challenge, my dissertation aims at bridging the gap between hardware and software formal analysis. This involves translating representations of verification tasks, generating certificates for verification results, integrating state-of-the-art formal analysis tools into a cohesive framework, and adapting and combining model-checking algorithms across domains. By translating word-level hardware circuits into C programs, we found out that software analyzers were able to identify property violations that well-established hardware verifiers failed to detect. Moreover, by adopting interpolation-based hardware-verification algorithms for software analysis, we were able to tackle tasks unsolvable by existing methods. Our research consolidates knowledge from both hardware and software domains, paving a pathway for comprehensive system-level verification.

Keywords: Hardware model checking · Software verification · Representation translation · Craig interpolation · Transferability · Btor2

1 Introduction

Computing systems are widely-adopted in modern society and ensuring their functional correctness is of utmost importance. Formal methods, grounded in theories of logic, automata, and constraint solving, have been applied in real-world applications and provided safety guarantees with mathematical rigor. With the ever-increasing interactions among diverse components, such as software programs, hardware circuits, and cyber-physical devices, formal verification of computing systems has also become more challenging.

While the research communities for formal methods share common theoretical foundations, including satisfiability solving [22], Craig interpolation [31], and abstraction refinement [29], differences in their alignment with distinct computational models create gaps between these communities. Figure 1 depicts such differences between hardware and software verification, highlighting a knowledge gap between the two closely-related fields. We mainly focus on hardware systems

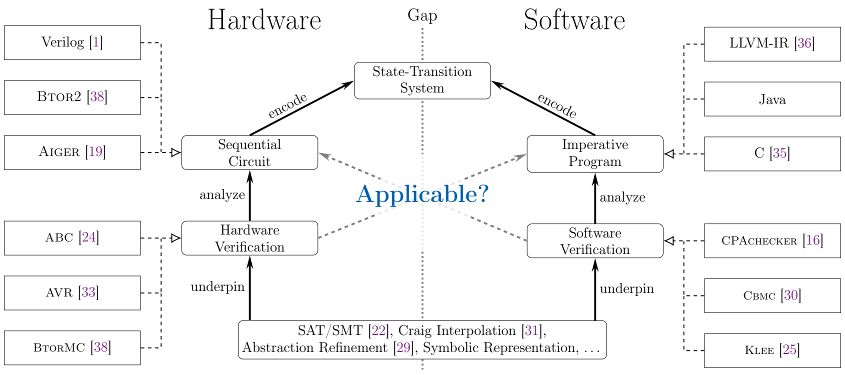

Fig. 1: The gap between hardware and software verification

represented by sequential circuits and software systems written as imperative programs, as they are widely used in practice. Both representations encode *state-transition systems*, and their verification approaches are underpinned by common foundations. Despite these similarities, verification tools (bottom-left and -right corners of Fig. 1) typically consume a specific input format (top-left and -right corners of Fig. 1). Therefore, applying software verifiers to analyze hardware systems or vice versa requires considerable engineering effort, which impedes mutual advancement between communities.

Objectives and Methodology. We plan to bridge the gap between hardware and software verification by considering the following question: ***Is hardware (resp. software) verification algorithms applicable to software (resp. hardware) systems?*** To address this, we employ two different strategies:

- **Task translation:** We translate verification tasks from one representation to a another, and utilize tools for the latter representation to analyze the correctness of the systems (see Sect. 2).
- **Algorithm adoption:** We adopt verification algorithms across domains and make necessary modifications to the algorithms to cater the needs of different system representations. The adopted algorithms can then be applied to verification tasks in the other domain (see Sect. 3).

After developing these strategies, we evaluate their effectiveness through extensive evaluation on thousands of benchmark tasks from both hardware- and software-verification communities. With the collected experimental data, we investigate the performance differences of different verification algorithms and tools across domains. This helps us further consolidate the knowledge and provides us insight on how to combine the strengthens of the two communities (see Sect. 4).

The scientific results of the dissertation have been published in three papers at TACAS [2, 6, 27], one paper at FSE [5], one paper at ASE [7], and one paper at SPIN [11]. The remainder of the manuscript summarizes the related work, high-level concepts, achieved results, and ongoing work of each aforementioned research direction.

2 Cross-Applying Hardware and Software Analysis via Representation Translation

In this section, we described our research on cross-applying hardware and software analyzers by translating the verification tasks between circuits and programs. We mainly focus on hardware circuits in the Btor2 format [38] and software programs in the C language, as they are the standard representations used by the Hardware Model Checking Competitions (HWMCC) [20] and Competitions on Software Verification (SV-COMP) [4], respectively, and come with extensive tool support.

Applying Software Analysis to Hardware Circuits. To apply software verifiers and testers to analyze hardware circuits, we first have to translate the circuits into programs. Our tool Btor2C [6], is the first translator that translates a word-level hardware circuit in Btor2 format [38] to a behaviorally-equivalent C program. We further augment the combination of Btor2C and software analyzers with a *witness translator* into a certifying hardware verification framework Btor2-Cert [2], that can produce certificates (i.e., verification witnesses [15]) for the derived verification results, to increase trustworthiness of the translation and verification processes. Our evaluation on a benchmark set consisting of over 1400 Btor2 tasks shows that: (1) It is feasible to translate Btor2 circuits into C programs. (2) Software analyzers can complement state-of-the-art hardware verifiers by finding more property violations and uniquely solving dozens of tasks.

Applying Hardware Model Checking to Software Programs. On the other direction of translation, we implement CPV [27], a circuit-based program verifier that uses sequential circuits as its intermediate representation. CPV uses Kratos2 [34] to translate C into Btor2 representation, and invokes hardware model checkers like ABC [24] and AVR [33] for verification. It can also extract error paths from Btor2 violation witnesses and export them in software-witness format [15]. As a first time participant in SV-COMP 2024, CPV attained remarkable results in the category *ReachSafety* (ranked 6 out of 26 participants) and surprisingly outperformed several established software verifiers. Furthermore, we plan to extend CPV in the following directions: (1) exploring circuit optimization techniques like combinational rewriting, which have proven effective in reducing circuit size and could potentially improve verification performance, for circuits translated from C programs, (2) producing software correctness witnesses through extracting and translating the fixed points computed by hardware model checkers.

3 Transferring Verification Algorithms Across Domains

Various formal-verification algorithms have been successfully transferred across domains. For instance, bounded model checking (BMC) [21], k-induction [39], and IC3/PDR [14, 23] are originally designed for finite-state systems such as sequential circuits. They have been successfully lifted to software verification [28, 30, 32] thanks to the commonalities between finite-state and infinite-state model checking. In this project, we conduct a systematic investigation into the transferability of two interpolation-based hardware-verification algorithms, *interpolation-sequence-based model checking* (ISMC) [40] and *dual approximated reachability* (DAR) [5], to software verification. Building on previous work on adopting hardware-verification algorithm via large-block encoding [13, 17], we implement ISMC and DAR within the configurable program analysis framework CPAchecker [3, 16], and subsequently evaluate their performance on more than 8000 benchmark tasks, covering a broad spectrum of software-verification problems. The experimental results demonstrate that the characteristics of ISMC and DAR are indeed transferable, and the two algorithms were able to tackle tasks unsolvable by existing techniques. This work consolidates the knowledge about the transferability of ISMC and DAR to software verification and highlight opportunities to enhance software verification by integrating methods from hardware model checking.

4 Joining Forces of Hardware and Software Verification

Each verification algorithm possesses its own unique strengths and weaknesses. An algorithm may perform poorly on a particular class of problems that others can solve efficiently. For example, data-flow analysis is a lightweight scalable technique that can handle large software systems, and has been employed by various static analyzers and compilers. However, it may yield overly-weak invariants insufficient for proving complex properties. There are also more intensive techniques, such as those based on Craig interpolation [31]. Interpolation-based model checking (IMC) [37], originally developed for hardware model checking and recently adopted for software verification [17] (see also Sect. 3), generates stronger invariants from interpolants, but could suffer from scalability issues due to costly interpolation procedures. By combining these two families of techniques, we can leverage the strengths of both to achieve better verification performance. To achieve the synergy, we propose a method to augment IMC [11] with auxiliary invariants generated by an interval-based data-flow analysis [7]. These auxiliary invariants help refine interpolants by pruning unreachable states such that the analysis is focused on reachable program parts. We implemented this approach within the CPAchecker framework [16], and evaluated its performance against established SMT-based methods within CPAchecker and other cutting-edge software verifiers. The findings show that our approach help reduce the number of interpolation queries required to prove safety properties and improve the overall runtime efficiency. As a result, our proposed invariant-injection technique successfully verified challenging tasks that the plain IMC (without auxiliary invariants), the invariant generator itself, or other compared tools could not solve.

5 Conclusion

This dissertation has successfully tackled some of the challenges in bridging hardware and software formal analysis by representation translation and the adoption of advanced verification algorithms across domains. Projects like Btor2-Cert and CPV have demonstrated the feasibility and effectiveness of applying software analysis to hardware systems and vice versa, thereby enlarging the applicability of formal verification methods. Our research not only promotes to the integration of traditionally-segregated research communities, but also sets the stage for future advancements in system-level verification.

Data-Availability Statement. All the software projects we developed are open-source on GitLab (https://gitlab.com/sosy-lab/software/, in repositories btor2c, btor2-cert, btor2-val, cpv, and cpachecker). To enhance the verifiability and transparency of the evaluation results reported in our papers, all used software, verification tasks, as well as raw and processed experimental results are available in supplemental reproduction artifacts archived on Zenodo [8, 9, 10, 12, 18, 26, 41].

References

1. IEEE standard for Verilog hardware description language (2006). https://doi.org/10.1109/IEEESTD.2006.99495
2. Ádám, Z., Beyer, D., Chien, P.C., Lee, N.Z., Sirrenberg, N.: Btor2-Cert: A certifying hardware-verification framework using software analyzers. In: Proc. TACAS (3). pp. 129–149. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_7
3. Baier, D., Beyer, D., Chien, P.C., Jankola, M., Kettl, M., Lee, N.Z., Lemberger, T., Lingsch-Rosenfeld, M., Spiessl, M., Wachowitz, H., Wendler, P.: CPAchecker 2.3 with strategy selection (competition contribution). In: Proc. TACAS (3). pp. 359–364. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_21
4. Beyer, D.: State of the art in software verification and witness validation: SV-COMP 2024. In: Proc. TACAS (3). pp. 299–329. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_15
5. Beyer, D., Chien, P.C., Jankola, M., Lee, N.Z.: A transferability study of interpolation-based hardware model checking for software verification. Proc. ACM Softw. Eng. **1**(FSE) (2024). https://doi.org/10.1145/3660797
6. Beyer, D., Chien, P.C., Lee, N.Z.: Bridging hardware and software analysis with Btor2C: A word-level-circuit-to-C translator. In: Proc. TACAS (2). pp. 152–172. LNCS 13994, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_12
7. Beyer, D., Chien, P.C., Lee, N.Z.: CPA-DF: A tool for configurable interval analysis to boost program verification. In: Proc. ASE. pp. 2050–2053. IEEE (2023). https://doi.org/10.1109/ASE56229.2023.00213
8. Beyer, D., Chien, P.C., Lee, N.Z.: Reproduction package for ASE 2023 article 'CPA-DF: A tool for configurable interval analysis to boost program verification'. Zenodo (2023). https://doi.org/10.5281/zenodo.8245821
9. Beyer, D., Chien, P.C., Lee, N.Z.: Reproduction package for TACAS 2023 article 'Bridging hardware and software analysis with Btor2C: A word-level-circuit-to-C translator'. Zenodo (2023). https://doi.org/10.5281/zenodo.7551707
10. Beyer, D., Chien, P.C., Lee, N.Z.: SoSy-Lab virtual machine (Ubuntu 22.04 LTS). Zenodo (2023). https://doi.org/10.5281/zenodo.1158641
11. Beyer, D., Chien, P.C., Lee, N.Z.: Augmenting interpolation-based model checking with auxiliary invariants. In: Proc. SPIN. Springer (2024)
12. Beyer, D., Chien, P.C., Lee, N.Z.: Reproduction package for SPIN 2024 submission 'Augmenting interpolation-based model checking with auxiliary invariants'. Zenodo (2024). https://doi.org/10.5281/zenodo.10548594
13. Beyer, D., Cimatti, A., Griggio, A., Keremoglu, M.E., Sebastiani, R.: Software model checking via large-block encoding. In: Proc. FMCAD. pp. 25–32. IEEE (2009). https://doi.org/10.1109/FMCAD.2009.5351147
14. Beyer, D., Dangl, M.: Software verification with PDR: An implementation of the state of the art. In: Proc. TACAS (1). pp. 3–21. LNCS 12078, Springer (2020). https://doi.org/10.1007/978-3-030-45190-5_1

15. Beyer, D., Dangl, M., Dietsch, D., Heizmann, M., Lemberger, T., Tautschnig, M.: Verification witnesses. ACM Trans. Softw. Eng. Methodol. **31**(4), 57:1–57:69 (2022). https://doi.org/10.1145/3477579
16. Beyer, D., Keremoglu, M.E.: CPAchecker: A tool for configurable software verification. In: Proc. CAV. pp. 184–190. LNCS 6806, Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_16
17. Beyer, D., Lee, N.Z., Wendler, P.: Interpolation and SAT-based model checking revisited: Adoption to software verification. J. Autom. Reasoning (2024). https://doi.org/10.1007/s10817-024-09702-9
18. Beyer, D., Wendler, P.: CPAchecker release 2.3. Zenodo (2023). https://doi.org/10.5281/zenodo.10203297
19. Biere, A.: The AIGER And-Inverter Graph (AIG) format version 20071012. Tech. Rep. 07/1, Institute for Formal Models and Verification, Johannes Kepler University (2007). https://doi.org/10.35011/fmvtr.2007-1
20. Biere, A., Froleyks, N., Preiner, M.: 11th Hardware Model Checking Competition (HWMCC 2020). http://fmv.jku.at/hwmcc20/, accessed: 2023-01-29
21. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers **58**, 117–148 (2003). https://doi.org/10.1016/S0065-2458(03)58003-2
22. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
23. Bradley, A.R.: SAT-based model checking without unrolling. In: Proc. VMCAI. pp. 70–87. LNCS 6538, Springer (2011). https://doi.org/10.1007/978-3-642-18275-4_7
24. Brayton, R., Mishchenko, A.: ABC: An academic industrial-strength verification tool. In: Proc. CAV. pp. 24–40. LNCS 6174, Springer (2010). https://doi.org/10.1007/978-3-642-14295-6_5
25. Cadar, C., Dunbar, D., Engler, D.R.: Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proc. OSDI. pp. 209–224. USENIX Association (2008), http://www.usenix.org/events/osdi08/tech/full_papers/cadar/cadar.pdf
26. Chien, P.C., Lee, N.Z.: CPV: A circuit-based program verifier. Zenodo (2023). https://doi.org/10.5281/zenodo.10203472, version 0.4
27. Chien, P.C., Lee, N.Z.: CPV: A circuit-based program verifier (competition contribution). In: Proc. TACAS (3). pp. 365–370. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_22
28. Cimatti, A., Griggio, A.: Software model checking via IC3. In: Proc. CAV. pp. 277–293. LNCS 7358, Springer (2012). https://doi.org/10.1007/978-3-642-31424-7_23
29. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Proc. CAV. pp. 154–169. LNCS 1855, Springer (2000). https://doi.org/10.1007/10722167_15
30. Clarke, E.M., Kröning, D., Lerda, F.: A tool for checking ANSI-C programs. In: Proc. TACAS. pp. 168–176. LNCS 2988, Springer (2004). https://doi.org/10.1007/978-3-540-24730-2_15
31. Craig, W.: Linear reasoning. A new form of the Herbrand-Gentzen theorem. J. Symb. Log. **22**(3), 250–268 (1957). https://doi.org/10.2307/2963593
32. Donaldson, A.F., Haller, L., Kröning, D., Rümmer, P.: Software verification using k-induction. In: Proc. SAS. pp. 351–368. LNCS 6887, Springer (2011). https://doi.org/10.1007/978-3-642-23702-7_26

33. Goel, A., Sakallah, K.: AVR: Abstractly verifying reachability. In: Proc. TACAS. pp. 413–422. LNCS 12078, Springer (2020). https://doi.org/10.1007/978-3-030-45190-5_23
34. Griggio, A., Jonáš, M.: Kratos2: An SMT-based model checker for imperative programs. In: Proc. CAV. pp. 423–436. Springer (2023). https://doi.org/10.1007/978-3-031-37709-9_20
35. ISO/IEC JTC 1/SC 22: ISO/IEC 9899-2018: Information technology — Programming Languages — C. International Organization for Standardization (2018), https://www.iso.org/standard/74528.html
36. Lattner, C., Adve, V.S.: LLVM: A compilation framework for lifelong program analysis and transformation. In: Proc. CGO. pp. 75–88. IEEE (2004). https://doi.org/10.1109/CGO.2004.1281665
37. McMillan, K.L.: Interpolation and SAT-based model checking. In: Proc. CAV. pp. 1–13. LNCS 2725, Springer (2003). https://doi.org/10.1007/978-3-540-45069-6_1
38. Niemetz, A., Preiner, M., Wolf, C., Biere, A.: Btor2, BtorMC, and Boolector 3.0. In: Proc. CAV. pp. 587–595. LNCS 10981, Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_32
39. Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a SAT-solver. In: Proc. FMCAD, pp. 127–144. LNCS 1954, Springer (2000). https://doi.org/10.1007/3-540-40922-X_8
40. Vizel, Y., Grumberg, O.: Interpolation-sequence based model checking. In: Proc. FMCAD. pp. 1–8. IEEE (2009). https://doi.org/10.1109/FMCAD.2009.5351148
41. Ádám, Z., Beyer, D., Chien, P.C., Lee, N.Z., Sirrenberg, N.: Reproduction package for TACAS 2024 article 'Btor2-Cert: A certifying hardware-verification framework using software analyzers'. Zenodo (2024). https://doi.org/10.5281/zenodo.10548597