Agile Formal Verification with Symbolic Quick Error Detection by Semantically Equivalent Program Execution

Yufeng Li[1], Qiusong Yang[2], Yiwei Ci[2], Enyuan Tian[2,3], Yungang Bao[1], Kan Shi[1]
[1]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2]Institute of Software, Chinese Academy of Sciences, Beijing, China
[3]University of Chinese Academy of Sciences, Beijing, China
crazybinary494@gmail.com, {qiusong, yiwei}@iscas.ac.cn, tianenyuan@nfs.iscas.ac.cn, {baoyg, shikan}@ict.ac.cn

Abstract

As processor complexity continues to grow and development cycles shorten, agile development becomes essential. Formal verification ensures design correctness but is labor-intensive and error-prone due to design-specific properties. Symbolic Quick Error Detection (SQED) avoids manually writing many properties by checking the design-independent, self-consistency universal property, thereby facilitating agile verification. However, since self-consistency is based on assertions that expect the processor to produce consistent results between the original and duplicate instructions, it fails to cover bugs that affect both the original and duplicate instructions, leading to false positives. To address this, we propose Symbolic Quick Error Detection by Semantically Equivalent Program Execution (SEPE-SQED), which utilizes program synthesis to find programs (instruction sequences) with equivalent meanings to original instructions. SEPE-SQED effectively detects the bugs missed by SQED by differentiating their impact on the original instruction and its semantically equivalent program. In the case study of a RISC-V processor, agile formal verification can improve productivity by approximately 60 times compared to conventional Formal Property Verification (FPV).

Introduction

Modern processor designs have become significantly more complex, while supply chain challenges, including global competition and geopolitical risks, have shortened development timelines. This has created major challenges for formal verification, as traditional methods require extensive manual effort to write design-specific properties, which are time-consuming and error-prone, hindering processor development[1].

Recent advancements in agile formal verification, such as *Symbolic Quick Error Detection* (*SQED*)[2, 3], present new opportunities for tackling these challenges. *SQED* utilizes *Bounded Model Checking (BMC)* to prove that any instruction sequence up to a certain bound produces a correct result. It leverages the concept of design self-consistency to establish a single universal property, which declares that the outcomes produced by both original instructions and their duplicates are identical, regardless of the specific microarchitectural design details. Therefore, *SQED* does not require manually writing design-specific properties. Practical examples have demonstrated that *SQED* can efficiently detect many bugs that are otherwise difficult to detect[2]. However, *SQED* lacks coverage for a class of bugs that can affect the execution states of both the original and duplicate instructions uniformly, leading to false positives in the verification results[3].

In this paper, we introduce an improved variant of *SQED*, named *Symbolic Quick Error Detection by Semantically Equivalent Program Execution* (*SEPE-SQED*)[4], by extending the self-consistency universal property. Specifically, *SEPE-SQED* checks that a correctly functioning processor executes the original instruction and its semantically equivalent program (instruction sequence) to produce consistent results. In the case of bugs missed by *SQED*, their effect on the original instruction and its semantically equivalent program can vary, leading to a consistency violation.

Evaluation using an open-source high-performance RISC-V processor demonstrates that *SEPE-SQED* successfully detects all 33 injected bugs, whereas *SQED* fails to identify 13 of them, which results in false positives. Compared to conventional FPV, agile methods such as *SQED* and *SEPE-SQED* can enhance productivity by approximately 60 times.

SQED and *SEPE-SQED*

This type of agile formal verification is based on the combination of *Quick Error Detection* (*QED*) testing technology and BMC[5]. *QED* comprises a series of systematic transformations that convert a wide range of existing original tests into new tests within the *QED* family. One such *QED* transformation is *Error Detection using Duplicated Instructions for Validation (EDDI-V)*. *EDDI-V* divides the processor's locations, including both the register file and memory, into two distinct regions: the original space and the duplicate space, while establishing a unique correspondence between them. At the start of the testing, each corresponding pair of registers (or memory locations) is initialized to the same value. *EDDI-V* then modifies the test program by duplicating instruction sequences. In the transformed program, the original instruction sequence operates on one half of the region, while the duplicate sequence operates on the other half. *In a correctly functioning processor, the architectural state after executing the original program must remain identical to the state after executing the corresponding duplicate program.* The BMC

explores the domain of all possible *QED* tests (within its bound) to find a counterexample to the following universal property (i.e., design-independent):

$$QED\text{-}ready \Rightarrow \bigwedge_{r \in O} \big(r == \Delta(r)\big) \tag{1}$$

where O represents the original space, and, for example, considering N general-purpose registers, $O ::= \{regs[0], \cdots, regs[\ N/2-1]\}$, while D denotes the duplicate space, $D ::= \{regs[N/2], \cdots, regs[N-1]\}$. The function Δ represents a bijective mapping from the original registers to the duplicate registers, i.e., $\Delta : O \mapsto D$. *QED-ready* indicates that both the original program and the corresponding duplicate program successfully write back their execution results.

SQED based on the *EDDI-V* transformation has limitations in terms of bug-type coverage. Specifically, there exists a class of bugs that can equally affect both the execution of original and duplicate instructions. For instance, if the instruction *SUB regs[3] regs[1] regs[2]* is misdecoded as *ADD regs[3] regs[1] regs[2]*, it can lead to false positives (where the processor's state still is consistent), because both the original and duplicate instruction perform the addition operation, allowing this bug to evade detection. To address this issue, we propose the *SEPE-SQED* based on the *Error Detection using Semantically Equivalent Program for Validation (EDSEP-V)* transformation, as illustrated in Figure 1. *EDSEP-V* transforms the original instruction, such as *SUB regs[3] regs[1] regs[2]*, into the semantically equivalent program *{XORI t1 regs[1] 0xfff, ADD t2 t1 regs[2], XORI regs[3] t2 0xfff}*. *In a correctly functioning processor, the architectural state after executing the original program must remain identical to the state after executing the corresponding semantically equivalent program.* Since the datapath of the original and semantically equivalent programs differ, false positives are avoided. Bugs affect only the original instruction (e.g., *SUB regs[3] regs[1] regs[2]* to *ADD regs[3] regs[1] regs[2]*), with the semantically equivalent program unaffected.

The transformation rules from the original instructions to corresponding semantically equivalent programs are determined by *Component-based Counterexample-Guided Inductive Synthesis (CEGIS)*[6]:

$$\exists P : \forall \vec{I}, O : \big(P(\vec{I}) == O\big) \Rightarrow \big(\phi_{orig}(\vec{I}) == O\big) \tag{2}$$

The semantically equivalent program P is derived through *Satisfiability Modulo Theories (SMT)* queries, ensuring that it produces the same output (e.g., destination operand) when given the same input (e.g., source operands) as the original instruction ϕ_{orig}. To accelerate synthesis, we propose *CEGIS based on the Highest Priority First (HPF-CEGIS)*, which dynamically adjusts the priorities of the components and heuristically selects the highest priority components, achieving a speed improvement 50%.

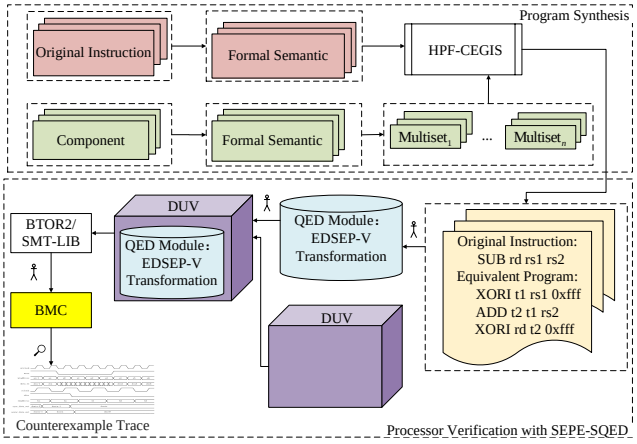

Figure 1: *SEPE-SQED*

Case Study

We choose the open-source RISC-V superscalar out-of-order processor RIDECORE[7] as the case study to evaluate the efficacy of *SEPE-SQED*. 33 different logic bugs from literature[8] and mutations were injected into the RTL code.

Compared to conventional FPV, agile formal verification methods like *SQED* and *SEPE-SQED* can enhance productivity by approximately 60 times. *SEPE-SQED* can detect all 33 bugs within one hour, whereas *SQED* identified only 20. The 13 missed bugs led to false positives in *SQED* checking.

References

[1] Pallab Dasgupta and Pallab DasGupta. *A roadmap for formal property verification*. Springer, 2006.

[2] Eshan Singh et al. "Symbolic qed pre-silicon verification for automotive microcontroller cores: Industrial case study". In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2019, pp. 1000–1005.

[3] Florian Lonsing et al. "Unlocking the Power of Formal Hardware Verification with CoSA and Symbolic QED". In: *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2019, pp. 1–8.

[4] Yufeng Li et al. "SEPE-SQED: Symbolic Quick Error Detection by Semantically Equivalent Program Execution". In: *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 2024, pp. 1–6.

[5] Ted Hong et al. "QED: Quick error detection tests for effective post-silicon validation". In: *2010 IEEE International Test Conference*. IEEE. 2010, pp. 1–10.

[6] Sumit Gulwani et al. "Synthesis of loop-free programs". In: *ACM SIGPLAN Notices* 46.6 (2011), pp. 62–73.

[7] FUJIV Arch Lab. in Tokyo Institute of Technology Olof Kindgren. *RIDECORE*. `https://github.com/ridecore/ridecore`. Accessed: 2024-02-26. 2017.

[8] Eshan Singh et al. "Logic bug detection and localization using symbolic quick error detection". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2018).