TAL TECH

DOCTORAL THESIS

Enhancing Assertion-Based Verification in Hardware Designs through Data Mining Algorithms

Mohammad Reza Heidari Iman

TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY
TALLINN 2024

TALLINN UNIVERSITY OF TECHNOLOGY
DOCTORAL THESIS
37/2024

Enhancing Assertion-Based Verification in Hardware Designs through Data Mining Algorithms

MOHAMMAD REZA HEIDARI IMAN

TAL TECH PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Computer Systems

The dissertation was accepted for the defence of the degree of Doctor of Philosophy in Information and Communication Technologies on July 17 2024.

Supervisor: Professor Dr. Tara Ghasempouri,
Department of Computer Systems,
Tallinn University of Technology,
Tallinn, Estonia

Co-supervisor: Professor Dr. Gert Jervan,
Department of Computer Systems,
Tallinn University of Technology,
Tallinn, Estonia

Opponents: Professor Dr. Goerschwin Fey,
Hamburg University of Technology,
Hamburg, Germany

Professor Dr. Mottaqiallah Taouil,
Delft University of Technology,
Delft, The Netherlands

Defence of the thesis: August 22 2024, Tallinn

Declaration:
Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology, has not been submitted for any academic degree elsewhere.

Mohammad Reza Heidari Iman _____
signature

European Union
European Regional
Development Fund

Investing
in your future

ISSN 2585-6898 (publication)
ISBN 978-9916-80-176-5 (publication)
ISSN 2585-6901 (PDF)
ISBN 978-9916-80-177-2 (PDF)
DOI https://doi.org/10.23658/taltech.37/2024
Printed by Koopia Niini & Rauam

Heidari Iman, M.R. (2024). *Enhancing Assertion-Based Verification in Hardware Designs through Data Mining Algorithms* [TalTech Press]. https://doi.org/10.23658/taltech.37/2024

TALLINNA TEHNIKAÜLIKOOL
DOKTORITÖÖ
37/2024

Andmekaeve algoritmide kasutamine riistvarasüsteemide väidete-põhise verifitseerimise parendamiseks

MOHAMMAD REZA HEIDARI IMAN

TAL TECH
KIRJASTUS

Contents

List of Publications

The present PhD thesis is based on the following publications that are referred to in the text by Roman numbers.

I MRH. Iman, G. Jervan, and T. Ghasempouri, "ARTmine: Automatic Association Rule Mining with Temporal Behavior for Hardware Verification," *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Valencia, Spain, 2024, pp. 1-6.

II MRH. Iman, J. Raik, M. Jenihhin, G. Jervan, and T. Ghasempouri, "An Automated Method for Mining High-Quality Assertion Sets," *Microprocessors and Microsystems Journal*, Vol. 97, pp. 104773, (2023), DOI: https://doi.org/10.1016/j.micpro.2023.104773.

III MRH. Iman, J. Raik, G. Jervan, and T. Ghasempouri, "IMMizer: An Innovative Cost-Effective Method for Minimizing Assertion Sets," *2022 25th Euromicro Conference on Digital System Design (DSD)*, Maspalomas, Spain, 2022, pp. 671-678, DOI: https://doi.org/10.1109/DSD57027.2022.00095.

IV A. Roberts and MRH. Iman, M. Bellone, T. Ghasempouri, J. Raik, O. Maennel, M. Hamad, and S. Steinhorst, "ADAssure: Debugging Methodology for Autonomous Driving Control Algorithms," *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Valencia, Spain, 2024, pp. 1-6.

V MRH. Iman, S. Ahmadi-Pour, R. Drechsler, and T. Ghasempouri, "Processor Vulnerability Detection with the Aid of Assertions: RISC-V Case Study," *TechRxiv, to be submitted*, 2024, pp. 1-8, DOI: https://doi.org/10.36227/techrxiv.172101134.45466090/v1.

Other related publications

VI MRH. Iman, J. Raik, M. Jenihhin, G. Jervan, and T. Ghasempouri, "A Methodology for Automated Mining of Compact and Accurate Assertion Sets," *2021 IEEE Nordic Circuits and Systems Conference (NorCAS)*, Oslo, Norway, 2021, pp. 1-7, DOI: https://doi.org/10.1109/NorCAS53631.2021.9599865.

VII H. Rostami, M. Hosseini, A. Azarpeyvand, MRH. Iman, and T. Ghasempouri, "Automatic High Functional Coverage Stimuli Generation for Assertion-based Verification," *The 30th IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS'24)*, Rennes, Brittany, France, 2024.

VIII MRH. Iman, P. Chikul, G. Jervan, H. Bahsi, and T. Ghasempouri, "Anomalous File System Activity Detection Through Temporal Association Rule Mining," *In Proceedings of the 9th International Conference on Information Systems Security and Privacy (ICISSP)*, Lisbon, Portugal, 2023, pp. 733-740, DOI: https://doi.org/10.5220/0011805100003405.

IX M. Shahin, M. Burtl, MRH. Iman, T. Ghasempouri, R. Sharma, SA. Shah, and D. Draheim, "Significant Factors Extraction: A Combined Logistic Regression and Apriori Association Rule Mining Approach," *13th Computer Science On-line Conference 2024 (CSOC'24)*, 2024.

X M. Shahin, MRH. Iman, M. Kaushik, R. Sharma, T. Ghasempouri, and D. Draheim, "Exploring Factors in a Crossroad Dataset Using Cluster-Based Association Rule Mining," *2022 the 13th International Conference on Ambient Systems, Networks and Technologies (ANT)*, Porto, Portugal, 2022, pp. 231-238, DOI: https://doi.org/10.1016/j.procs.2022.03.032.

Author's Contributions to the Publications

I In Publications I to III, as well as Publications V, VI, and VIII, I served as the main author, conceptualizing the paper's idea, formulating the initial hypotheses, and proposing all methodologies and algorithms. Additionally, I undertook the responsibility for implementing the algorithms and methods, conducting experiments, and collecting the results. As the lead author, I authored the entire of the initial manuscript.

II In Publication IV, A. Roberts and I are both listed as primary authors, sharing equal first authorship. The initial idea and formulation of all algorithms and methodologies stem from my contributions. Additionally, I executed the implementation of all algorithms presented in the paper. While I was responsible for collecting the experimental results, specifically related to the association rule mining algorithms, the analysis of these findings was conducted collaboratively with the co-first author. The writing of the paper was a joint effort between the authors.

III In Publication VII, my contributions involved generating assertion sets for the experimental section, assisting in shaping the paper's methodology, and contributing to the writing of some sections.

IV In Publications IX and X, I contributed to the writing of certain sections of the papers.

Abbreviations

ABV	Assertion-Based Verification
AD	Autonomous Driving
AR	Association Rule
AV	Autonomous Vehicle
BIST	Built-in Self-Test
CC	Correlation Coefficient
DDG	Dynamic Dependency Graphs
DoS	Denial-of-Service
DSI3	Distributed System Interface
DT	Different Terms
DUV	Design Under Verification
FSM	Finite State Machine
GPS	Global Positioning System
HDL	Hardware Description Language
IMU	Inertial Measurement Unit
LBDR	Logic-Based Distributed Routing
LTL	Linear Temporal Logic
MBIST	Memory Built-in Self-Test
NDT	Normal Distributions Transform
NoC	Network on Chip
PSL	Property Specification Language
RISC-V	Reduced Instruction Set Computer-Five
RTL	Register-Transfer Level
SoC	System-on-Chip
SOP	Sum-Of-Products
SQL	Structured Query Language
SVA	SystemVerilog Assertions
TLM	Transaction-Level Model
VCD	Value Change Dump

1 Introduction

In our modern lives, electronic devices are ubiquitous. From basic household items like radios and televisions to sophisticated machinery such as airplanes, automobiles, and medical equipment, these devices all rely on digital and embedded systems [1–4]. With each iteration, these systems grow increasingly intricate. Given their application in various safety-critical contexts, ensuring their correctness and trustworthiness is paramount. Any malfunction or failure within these systems can lead to dire consequences, ranging from financial or environmental damage to loss of life or serious injury [5, 6].

To ensure the correctness of the functionality of these systems various functional verification techniques are employed [7–11]. Functional verification techniques aim to verify that a hardware design behaves in accordance with a specified set of requirements outlined in its specification [12]. Throughout the verification process, a verification engineer ensures that the system is developed according to its specifications and requirements [13]. As hardware designs continue to increase in size and complexity, verification becomes a critical process of modern design flows [14]. Based on the 2020 Wilson Research Group's study on functional verification, approximately 51% of the total cost and time in the design process is allocated to verification [7, 15].

The costs associated with undetected bugs in designs escalate over time. Early detection of bugs during the design process reduces the expenses incurred in identifying and resolving them later when the design is being used in the real operational environment. Detecting a bug in a system post-fabrication can result in significant financial losses, often amounting to thousands of dollars, as it necessitates re-fabrication and prolongs time-to-market. Moreover, undetected bugs during the design process may be identified by customers, leading to warranty replacements and potentially damaging the reputation of the company or brand. These bugs and errors can affect the functionality of the system or lead to even security issues and vulnerabilities in the system [7, 16–19]. Given the significant costs associated with verification, it is imperative to improve and enhance current verification techniques.

To enhance existing functional verification techniques, it is vital to understand various methods and approaches in verification. These include static verification techniques, also referred to as formal verification [8, 20–25], dynamic verification approaches known as simulation-based verification techniques [26–28], and semi-formal methods commonly referred to as Assertion-Based Verification (ABV) techniques [7, 29–33]. Formal verification techniques aim to establish the functional correctness of a design through proof rather than simulation [34]. These techniques encompass various methods, including model checking, theorem proving, and equivalence checking [34]. Conversely, simulation-based verification techniques assess design functionality through simulation, checking its correctness within the simulated environment [26, 35, 36]. Despite the advantages of each verification method, they encounter drawbacks such as scalability issues in formal verification techniques and limitations in exhaustiveness for simulation-based approaches [34, 37, 38].

Although formal and simulation-based verification techniques hold promise, assertion-based verification has emerged as a popular approach for verifying complex digital systems. ABV combines the strengths of both formal and simulation-based methods to verify design functionality [39–46]. It can be implemented across various levels of verification abstraction, from high-level assertions within transaction-level testbenches to implementation-level assertions synthesized into hardware [7, 40, 47–51]. Assertion-based verification techniques rely primarily on the efficacy of assertions. These assertions are regarded as valid and accurate rules that delineate the behavior and functionality of

designs, which the designs must adhere to [42, 52].

On the other hand, given the various security vulnerabilities and attacks prevalent in contemporary designs, it is imperative to include security verification alongside functional verification in the design process. Functional verification techniques, such as formal verification and ABV, have demonstrated significant potential in verifying designs. The efficacy and benefits of these verification techniques have prompted researchers to explore their application in security verification as well [53]. Consequently, current research efforts are exploring the integration of functional verification techniques into the realm of security verification [53]. Some researchers have started utilizing formal verification for security verification to establish the trustworthiness of the designed systems against security vulnerabilities [53]. However, studies indicate that due to the effectiveness of assertion-based verification in verifying designs, ABV has attracted considerable attention for the security verification of hardware designs [54]. Hence, considering the advantages of ABV for functional verification and its efficacy in detecting security vulnerabilities, its incorporation into security verification processes shows significant promise.

This thesis introduces several innovative data mining-based solutions and methodologies designed to enhance assertion-based verification techniques. These proposed solutions, applicable in both verification and security verification domains, strive to generate assertions of superior quality and accuracy, ultimately reducing overall verification costs. Moreover, these methodologies cater to both academia and industry, providing valuable insights for researchers and engineers.

1.1 Motivations and Objectives

As illustrated in Fig. 1, this thesis endeavors to improve both functional verification and security verification by introducing innovative solutions for the automatic generation of assertions. These assertions, applicable for the verification and security verification of hardware designs, embedded systems, and cyber-physical systems like autonomous vehicles (AVs), can subsequently undergo automatic evaluation and minimization. This process yields a set of evaluated, minimized, and high-quality assertions, consequently reducing the overall costs associated with functional verification and security verification processes.

Regarding functional verification, while formal verification techniques hold promise and can accurately verify designs, they also suffer from some limitations. Applying these techniques to large-scale designs can be time-consuming and may result in memory explosion [34, 55]. Conversely, while memory explosion is a significant issue in formal verification, simulation-based verification approaches face a critical challenge due to the lack of exhaustiveness inherent in simulation-based verification processes [37, 56]. Therefore, simulation-based verification is adept at identifying bugs but cannot guarantee their absence [37].

Assertion-based verification leverages both formal verification and simulation-based verification to offer a more robust and streamlined approach to verifying complex digital systems [7]. This technique utilizes assertions, which serve as the golden rules to which the implementation is compared and any deviation from these rules can cause design errors [57]. In the domain of assertion-based verification, two primary approaches exist for defining assertions: manual and automatic approaches of assertion generation [42, 58].

The manual creation of assertions requires human expertise and a profound understanding of the design's functionality [57, 59–61]. Furthermore, it is costly and

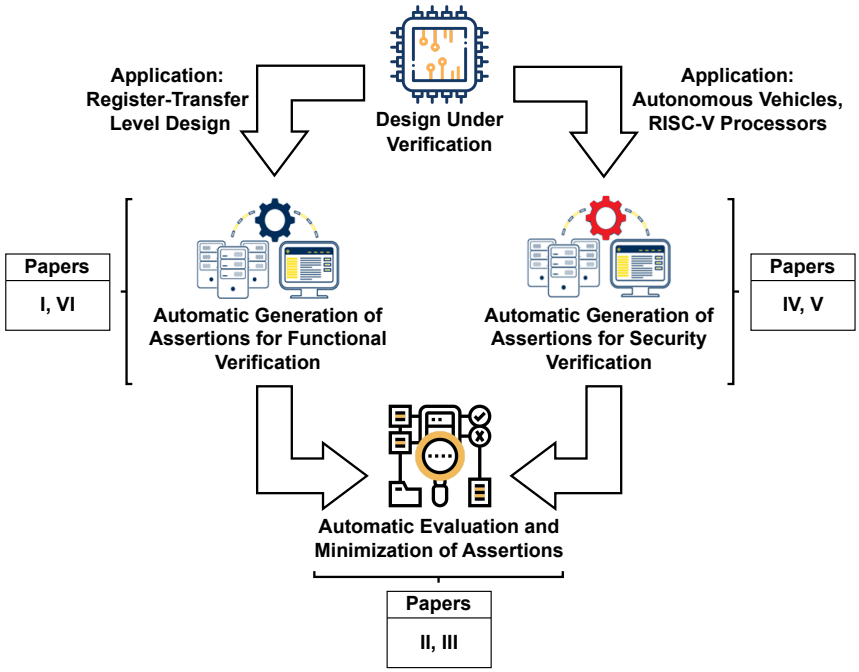


Figure 1: Contributions of the thesis

error-prone [59, 62]. Consequently, there have been numerous efforts to automate assertion mining [42, 63–75]. While these automated techniques hold promise, they still suffer from certain limitations and drawbacks. One significant drawback of current assertion miners is their inability to generate accurate assertion sets with comprehensive design behavior coverage. This can result in assertions that fail to encompass all design scenarios and corner cases, ultimately leading to incomplete assertion sets in terms of design behavior coverage. Other key limitations of state-of-the-art assertion miners include the excessive number of generated assertions, redundancy (*i.e.,* assertions that describe the same design behavior) and inconsistency (*i.e.,* assertions that contradict each other) in the mined assertions, prolonged execution times for assertion mining, and lack of readability in the generated assertions [76, 77]. Any of these deficiencies can render the verification process more error-prone, time-consuming, and costly.

Furthermore, in the current state-of-the-art, several studies have focused on evaluating the quality, interestingness, and ranking of assertions [63, 78–82]. However, although these methods are capable of evaluating and ranking assertions, each may yield a different set of ranked assertions, with rankings varying significantly between different methods. This discrepancy arises from the diverse criteria and metrics employed by each method to evaluate and rank assertions. Consequently, there is a notable absence in the literature of a unified method that integrates all existing interestingness and evaluation techniques, providing a cohesive and comprehensive set of evaluated and ranked assertions. Each of these shortcomings can present challenges for verification engineers in verifying hardware designs, ultimately resulting in various costs associated with the verification process.

On the other hand, in the realm of security and security verification, there has

been limited effort to leverage assertion and assertion-based verification techniques to enhance system security [33, 83–86]. For instance, in [83], security assertions for System-on-Chip (SoC) vulnerabilities are manually defined, yet they tend to be overly general, lacking coverage of design corner cases. Conversely, in [87], security assertion sets are automatically generated, albeit solely applicable to firmware. Additionally, there are endeavors concentrating on processor security using assertions, such as [54, 88–91]. However, these approaches are hindered by the manual assertion generation process or are restricted to specific processor types.

In the context of the safety and security of autonomous vehicles, there are a few works using assertions [92–95]. Nevertheless, these methodologies are limited to only addressing the software-level security of autonomous vehicles. Furthermore, they rely on manual processes tailored to individual AV models, rendering them inapplicable to other autonomous vehicles.

In this thesis, innovative solutions are proposed to enhance assertion-based verification techniques and improve the quality of existing assertion miners and their generated assertion sets, consequently overcoming the current limitations in the state of the art. Furthermore, several novel methods and solutions are proposed to enhance the security verification of processors and autonomous vehicles, addressing the aforementioned shortcomings in this domain.

1.2 Novelty, Contributions, and Outline of the Thesis

The outcomes of this thesis are structured into three primary branches, each explored in distinct chapters. Table 1 briefly illustrates the interconnection among the chapters, papers, and contributions. One branch of the thesis, as also depicted in 1 is associated with the automatic generation of assertions for functional verification. This branch involves the introduction of ARTmine, an automated assertion miner proposed for use by researchers and verification engineers in academia and industry. Chapter 4 delves into ARTmine, complemented by the details provided in paper I. In contrast to conventional methods where assertion miners rely on Finite State Machine (FSM) creation, ARTmine employs a suite of association rule mining algorithms specifically tailored for the context of ABV and assertion mining. These algorithms form the foundation of ARTmine, facilitating the generation of more accurate assertions within shorter timeframes. The proposed assertion miner offers superior design behavior coverage with a reduced number of assertions compared to existing assertion miners. ARTmine produces assertions that effectively address corner cases in designs and offer extensive design behavior coverage.

As presented in Fig. 1, the other branch of this thesis is associated with the automatic generation of assertions for security verification. In this branch of the thesis, the application of assertion-based verification is extended to the domain of security to address the limitations of existing approaches in security verification. Here, ADAssure is proposed as a data mining-driven technique designed for debugging and localizing bugs within the autonomous driving control algorithms of autonomous vehicles. This method employs a novel data mining algorithm to analyze the behavior of AVs and pinpoint bugs, particularly those manifesting as adversarial cyber-attacks, with the support of assertions. Additionally, an automatic security-based assertion miner tailored for generating security assertions specific to RISC-V processors, with potential applicability to security verification across other processor types, is introduced. The proposed security-based assertion miner is capable of detecting security vulnerabilities, including hardware Trojans. To the best of my knowledge, this marks the first instance of an assertion miner capable of automatically generating security assertions for processors.

Table 1: Thesis contributions and outline

Chapter	Paper	Contribution
4	I	Automatic Generation of Assertions for Functional Verification (ARTmine)
5	II, III	Automatic Evaluation and Minimization of Assertions (Dominance and IMMizer)
6	IV, V	Automatic Generation of Assertions for Security Verification (ADAssure and Security-Based Assertion Miner)

Comprehensive details of these methodologies are expounded upon in Chapter 6. The corresponding paper for ADAssure is paper IV, while the paper corresponding to the security-based assertion miner for RISC-V processor is paper V.

The final branch of this thesis, as illustrated in Fig. 1, focuses on the automatic evaluation and minimization of assertions. While in this thesis the methodologies and solutions proposed in this branch have been employed for functional verification, they are also applicable to security verification. This branch of the thesis focuses on addressing the deficiencies present in the current state-of-the-art assertion-based verification, particularly concerning assertion evaluation, interestingness, redundancy, and inconsistency. To tackle these challenges, Dominance, a data mining-based approach for analyzing, evaluating, and qualifying assertion sets, is introduced. This method is capable of assessing various assertion rankings, consolidating them, and presenting a unified and integrated collection of evaluated, high-quality assertions, while considering diverse rankings from other methodologies. Furthermore, to address issues of redundancy and inconsistency within assertions, IMMizer, a novel method that identifies and minimizes redundant and inconsistent assertions is presented. IMMizer has been specifically designed to detect and minimize these types of assertions. IMMizer generates a reduced number of assertions with minimal redundancy and inconsistency compared to the original assertions while maintaining the same level of design behavior coverage. The intricacies of these methods are elaborated in Chapter 5, with detailed discussions available in papers II and III. Incorporating ARTmine, Dominance, and IMMizer into the hardware design verification process can furnish a comprehensive toolkit, significantly reducing overall verification costs.

The organization of the thesis and a brief description of each chapter are as follows:

- **Chapter 2** This chapter offers the relevant background and concepts necessary for comprehending the remainder of the thesis. It provides a concise overview of verification, outlining its various approaches and types. Additionally, the chapter introduces assertion-based verification and defines assertions. Finally, it delves into the mutant analysis technique utilized in assertion-based verification.

- **Chapter 3** In this chapter, state-of-the-art methods and approaches documented in the literature pertaining to automatic assertion mining, automatic assertion evaluation and minimization, and the application of assertion-based verification in the realm of security are elucidated. The doctoral contributions in this thesis lie within these domains, and the subsequent chapters elaborate on each contribution in greater depth.

- **Chapter 4** The primary objective of this chapter is to introduce the automatic assertion miner, ARTmine. To develop ARTmine, several association rule mining

algorithms specifically tailored for extracting crucial temporal patterns within the framework of assertion-based verification are introduced. This chapter provides an overview of these algorithms, followed by a detailed exposition of ARTmine.

- **Chapter 5** This chapter introduces Dominance, a data mining-based algorithm designed to analyze the quality of assertions. Unlike conventional assertion evaluation methods that merely rank assertions, Dominance can assess assertion quality and eliminate low-quality ones. The intricacies of Dominance are elaborated upon in this chapter. Additionally, this chapter provides details on IMMizer. IMMizer operates by categorizing assertions into various classes and subsequently identifying redundant and inconsistent assertion sets. Upon detection, IMMizer minimizes these types of assertions collectively.

- **Chapter 6** This chapter incorporates assertion-based verification into the realm of security and elucidates the intricacies and outcomes of the contributions made by this PhD thesis in this domain. A method is introduced for bug localization and debugging within autonomous driving control algorithms in autonomous vehicles, utilizing assertions, which is expounded upon in this chapter. The proposed method was implemented and assessed using the TalTech autonomous vehicle, IseAuto, as a case study. Moreover, this chapter introduces an automatic security-based assertion miner tailored for the security verification of RISC-V processors.

- **Chapter 7** This chapter serves as the conclusion to my thesis and outlines potential future research and development directions.

2 Background

Traditionally, the verification of hardware designs and embedded systems has involved two distinct techniques: static verification and dynamic verification [8, 37, 96, 97]. Static verification employs formal methods to ensure design correctness, while dynamic verification relies on simulation [23, 26, 98]. While both techniques offer unique advantages, they also have limitations in hardware verification. Recently, there has been significant interest in assertion-based verification, which combines the mathematical rigor of formal methods with the intuitive and rapid approach of simulation and its coverage metrics [99, 100]. In the following, an overview of static verification, dynamic verification, and assertion-based verification is elaborated in sections 2.1, 2.2, and 2.3, respectively. Furthermore, to explore the application of these functional verification techniques for security verification purposes, an overview of security verification is presented in section 2.4.

2.1 Static Verification

Static verification, also known as formal verification, utilizes mathematical proofs to establish the correctness of the Design Under Verification (DUV) [12]. Unlike dynamic verification, static verification is static in nature, as it verifies the presence of errors without explicitly simulating the behavior of the model [101].

A formal verification framework comprises three fundamental elements: a mathematical model of the system under verification, a formal language for framing the correctness problem, and a methodology for proving the correctness statement [101]. Based on these components, there are two primary formal verification mechanisms that can be applied for different purposes: model checking and equivalence checking [23]. In model checking, the correctness problem involves demonstrating whether the model satisfies the specification represented as logical formulas [23, 102]. Its primary application is to detect design errors at the early stage of the design process [103]. Conversely, equivalence checking addresses the task of verifying if two models implement the same functionalities [23, 103]. Its main application is to identify discrepancies between descriptions of the same system at different abstraction levels [23, 103]. Static approaches in functional verification offer significant advantages. However, they also present limitations, such as memory explosion and scalability issues, particularly in large-scale designs [34, 55]. Consequently, it is beneficial to consider alternative verification techniques.

2.2 Dynamic Verification

An extensively utilized alternative to formal verification is dynamic verification, also known as simulation-based verification, which assesses the correctness of a design through simulation-based techniques [37]. In dynamic verification, the model's functionality is verified by generating a substantial number of input stimuli (test cases), which are then simulated to observe the behavior of the DUV at its primary outputs [27, 104, 105]. To execute dynamic verification, several components are required: a description of the design, a testbench for applying stimuli to the primary input of the design, obtaining simulation traces, and a method for establishing the correctness of the design based on the simulation results [37, 105]. While dynamic verification shows promise in bug detection, it cannot guarantee their absence due to the lack of exhaustiveness inherent in the simulation-based verification process [104, 105].

2.3 Assertion-Based Verification

Assertion-based verification leverages the strengths of both formal verification and simulation-based verification techniques to offer a more robust approach to verifying complex digital systems [7, 106]. ABV primarily relies on the efficacy of assertions [75, 107]. In the domain of hardware verification, a significant challenge pertains to enhancing controllability and observability to uncover internal errors and bugs [7]. Controllability refers to the capability to control internal signals, while observability denotes the ability to monitor the state of designs [7]. Embedded assertions serve to capture any unexpected behavior, thereby increasing observability of internal activities within the design [7]. For instance, an assertion may verify that the output of an adder corresponds to the sum of its inputs, even when the implementation occurs within the execution stage of a CPU. Any deviation from the predefined properties within assertions can readily reveal and detect design bugs [7, 106, 108].

Enhancing the observability of internal states facilitates faster error localization, resulting in a considerable reduction in overall verification time [7, 106, 108]. As depicted in Fig. 2, ABV improves both controllability and observability, accelerating the debugging process and reducing functional errors within designs [7]. In the event of assertion failure during simulation, sufficient information is provided to designers and verification engineers, enabling them to promptly address the issue. Conversely, without assertions, diagnosing the cause of failure may take hours or even days. Consequently, the effective utilization of assertions can significantly reduce both verification and debugging times [7, 106, 108].

Additionally, ABV can be applied at various levels of abstraction, including Transaction-Level Model (TLM), Register-Transfer Level (RTL), and gate-level [7, 42]. Compared to RTL designs, TLM is more abstract and offers faster simulation [7, 42]. Consequently, TLM designs are better suited for verifying large designs and hardware/-software co-designs [7, 26, 42]. Furthermore, as illustrated in Fig. 2, assertion-based verification utilizes assertions that can be employed for pre-silicon verification, post-silicon verification, and in-field operations [7].

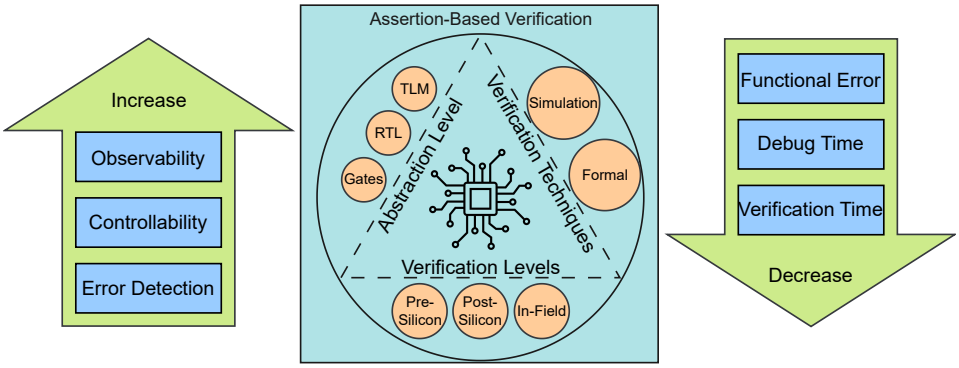


Figure 2: Overview of Assertion-Based Verification in Hardware Designs

2.3.1 Definition of Assertions

In ABV, an assertion represents a Boolean expression that delineates the behavior of hardware designs [42, 72]. Assertions are classified as immediate and concurrent depending on their integration into the design [7]. For instance, *assert(req2==ack2)*

constitutes an immediate assertion, triggered promptly after the execution of the statement 'req2=ack2' if req2 is unequal to ack2. Conversely, *assert property(ack1 |-> ∼ ack2)* is a concurrent assertion, operating concurrently with other modules and activating when both ack1 and ack2 become true. Subsequently, the following section formally defines the assertion by outlining several key terms [42]:

Definition 1 *An **atomic proposition** $\mathcal{P}$ is a logical formula that does not contain logical connectives.*

Example: Examples of *atomic propositions* are such as $\mathcal{P}1 =$ True, and $\mathcal{P}2 =$ False.

Definition 2 *A **proposition** is a composition of atomic propositions $\mathcal{P}$ through logical connectives such as && and ||.*

Example: An example for *proposition* is a1 = True && b1 = False && c1 = True.

Definition 3 *An **assertion** $\mathcal{A}$ is a composition of propositions through temporal operators that must hold or must become true during the execution of the design. Typically, an assertion is divided into two parts: the left side, named antecedent, and the right side, called consequent.*

The general structure of an assertion in Property Specification Language (PSL) is like $always(antecedent \rightarrow consequent)$, which implies that the consequent will hold whenever the antecedent occurs [109].

2.4 Security Verification

In the literature, numerous studies aim to enhance the security verification of DUVs by leveraging functional verification techniques. The fundamental distinction between functional and security verification lies in their objectives: the former addresses functional problems in alignment with functional specifications, while the latter identifies security issues by considering security requirements and threat models [110]. The majority of security verification techniques rely on various formal verification methods such as model checking and theorem proving, primarily applicable to the RTL level of abstraction, while some works also utilize information flow tracking techniques [53]. However, employing these techniques introduces various overheads in the design process, primarily due to the inherent limitations of static and formal verification methods (section 2.1). Conversely, assertion-based verification techniques show promise in covering diverse security aspects and detecting various classes of security vulnerabilities and attacks [54, 111, 112]. This underscores the potential of ABV for utilization and enhancement in the context of current complex DUVs.

3 State-of-the-Art

In this chapter, the state-of-the-art in automatic generation of assertions for functional verification, automatic evaluation and minimization of assertions, and automatic generation of assertions for security verification are presented.

3.1 Automatic Generation of Assertions for Functional Verification

In the domain of assertion-based verification, numerous automatic assertion miners have been developed with the objective of automatically extracting assertions. The study presented in [65] introduced an assertion miner that utilizes dynamic dependency graphs to extract signal relationships within the design and generate assertions. Notably, this assertion miner requires only a few simulation runs, significantly reducing the necessary number of use cases compared to alternative methods. Furthermore, the approach proposed in this study does not rely on expression templates to establish signal relationships within the design.

Another approach, as outlined in [66], involves a syntax-guided enumeration assertion miner that produces a collection of short and comprehensible assertions for sequential logic networks. In this study, the generated temporal properties delineate the relationship between the primary inputs and outputs, as well as the latches within the network. The methodologies proposed in [42] and [80] involve techniques that extract assertions utilizing various templates structured as Finite State Machines (FSMs). A-Team, as described in [42], integrates coverage analysis and data mining to generate a compact and expressive set of Linear Temporal Logic (LTL) assertions. This miner, guided by user-defined templates, extracts assertions from the simulation trace of the design, irrespective of the abstraction level (TLM, RTL, gate-level) of the design. HARM [72] represents a hint-based assertion miner, generating LTL assertions from a set of user-defined hints and the simulation trace of the DUV. An advantage of this approach is its ability to extract assertions without access to the source code of the design under verification, rendering it a notably efficient assertion miner. However, it tends to produce an excessive number of assertions, potentially prolonging the verification process. Furthermore, the work presented in [73] constitutes an extended version of HARM, leveraging a clustering algorithm to refine the assertion sets. Compared to the original HARM [72], this extended version yields a more effective assertion set in terms of mutant detection coverage.

The GoldMine assertion miner, introduced in [64], can generate assertions for a given RTL design through the utilization of formal verification and static code analysis. As one of the pioneering assertion miners in the field of automatic assertion mining, GoldMine aims to minimize human effort, time, and resources required for the verification process. However, its applicability is restricted to the source code of the DUV in Verilog alone, lacking support for other Hardware Description Language (HDL) DUVs. Additionally, the method outlined in [71] extracts assertions by converting sequential designs into pseudo-combinational designs. This assertion miner is proficient in generating compact and comprehensible assertions, facilitating verification engineers in the verification of designs with a limited number of assertions. Another assertion miner outlined in the literature is discussed in [58]. This assertion miner adopts a dynamic approach, incrementally analyzing control signals within the simulation traces of the DUV. The assertions generated by this approach are notably more expressive, effectively capturing the I/O communication protocol. Additionally, the study outlined in [52] combines dynamic dependency graphs and FSMs to harness the advantages of both techniques

for assertion mining.

While current assertion miners exhibit promise, they are not without their drawbacks, indicating a need for improvement. For instance, assertion miners discussed in [42,65,66] suffer from redundancy and inconsistency, necessitating additional tools like IMMizer [76] for resolution. Some miners, such as those outlined in [66], [72], and [73], produce an excessive number of assertions, requiring tools like Dominance [77] and Shayan [78] to sift through and select the most appropriate assertions. Readability issues may arise, particularly with complex antecedents, as seen in the work described in [65]. Furthermore, extended execution times observed in works such as [113] result in increased costs during the verification process. These limitations underscore the importance and necessity of proposing new techniques aimed at mining more effective assertion sets.

3.2 Automatic Evaluation and Minimization of Assertions

The assertions generated by existing assertion miners in the state-of-the-art exhibit certain shortcomings that require resolution. Specifically, there remains a significant need for efforts in ranking and evaluating assertions to ensure high-quality assertions capable of providing comprehensive coverage of design behavior. In this regard, studies have been conducted to estimate the quality of assertions using various proposed metrics. For instance, in [81], assertions are evaluated and ranked based on two primary metrics: *'Importance'* and *'Complexity'*. The degree of *'Importance'* is higher for assertions describing the output of the design, while *'Complexity'* pertains to the number of logical operators used in an assertion. In [63], the quality estimation is predicated on the number of propositions included in the antecedent of the assertion. Another study, as presented in [114], introduced a ranking function that assesses the quality of mined assertions based on a cause-effect relationship between the antecedent and consequent of an assertion. In works [78–80, 82], mined assertions are primarily ranked using data mining-based metrics such as *'Support'* (*i.e.*, their frequency of occurrences during simulation), *'Correlation Coefficient'* (*i.e.*, the correlation of the occurrence of an assertion to other assertions during simulation), and *'IS'* measure (*i.e.*, assertions with a low frequency of occurrence but high correlation to other assertions), etc.

However, these assertion evaluators primarily compute incompatible degrees of quality for a specific assertion. This disparity arises from the inclusion of multiple metrics in existing approaches, with each metric addressing a specific aspect of the assertion. For example, the methodology outlined in [79] assesses the quality of each assertion based on three metrics: *'Support'*, *'Correlation Coefficient'*, and *'IS'* measure. Nevertheless, the quality assessment for each assertion based on these metrics may differ. For instance, assertion $\mathcal{A}$ may be classified as high-quality based on *'Support'*, but as medium-quality and low-quality based on *'Correlation Coefficient'* and *'IS'* measure, respectively.

Hence, the crucial question at this juncture is: which metric can dominate the other metrics for estimation of the assertion quality? Is assertion $\mathcal{A}$ deemed high-quality, medium-quality, or low-quality? The studies mentioned above fail to establish assertion quality using a unified metric, resulting in a lack of consensus on the evaluation of assertions. This limitation underscores the necessity of introducing an innovative method for ranking and evaluating mined assertions, as well as unifying the outcomes of diverse assertion evaluation techniques in the current state-of-the-art.

On the other hand, apart from assertion evaluation, the generated assertions from assertion miners exhibit additional shortcomings. These assertions are often not human-readable, containing numerous propositions that diminish their readability. Moreover, each proposition typically requires memory allocation, thereby imposing overheads (*e.g.*

memory overhead) on the system. Furthermore, within the set of generated assertions, redundancy and inconsistency are prevalent, resulting in prolonged verification time, increased costs, and a susceptibility to errors during the verification process.

To address the aforementioned drawbacks, several studies have been conducted to minimize the extracted assertion sets. Typically, the main concept in the literature is to select only the best and most interesting assertions for the verification process. These approaches are generally categorized into two groups. The first approach involves data-mining-based techniques, such as those discussed in previous paragraphs ([63, 78–82, 114]). The second approach is the mutant-analysis-based approach ([115–117]). In the second approach, mutants are injected into the DUV, and the assertions that detect more mutants are typically selected for the verification process. While these approaches are primarily utilized for assertion evaluation, they can also be considered for assertion minimization.

However, despite the aim of the aforementioned studies to generally reduce the number of assertions, they can only select a few of the best assertions among the others. In fact, they are unable to minimize the initially generated assertion set to eliminate their redundancy and inconsistency and consequently decrease the overheads imposed on the system. In practice, in these approaches, the number of assertions remains unchanged, and the verification engineer selects a subset from the assertion set. However, this kind of assertion selection typically leads to issues such as inaccuracies in the verification process and lower design behavior coverage. Therefore, a gap exists in assertion minimization in the literature, and novel solutions are needed to address it.

3.3 Automatic Generation of Assertions for Security Verification

While there is an extensive body of literature on utilizing assertions for functional verification, there has been limited initial effort in addressing security vulnerabilities using assertions. In the realm of security, assertions can be employed to ensure that an unspecified transition in an FSM does not grant a user access to a higher privilege level. Similarly, assertions can be utilized to identify a wide range of other security vulnerabilities. The study in [54] delineated several classes of security vulnerabilities detectable using assertions. These classes are such as permissions and privileges, unauthorized resource accesses, illegal states and transitions in FSMs, numeric exceptions, buffer errors, malicious implants, and spectre attacks. Furthermore, there have been preliminary endeavors to employ assertions for detecting vulnerabilities in processors, as evidenced by studies such as [88–91].

In [88], security assertions for processor vulnerabilities are manually crafted to validate security requirements against the processor design. However, this manual approach demands significant time and expertise, leading to high costs. The method presented in [89] translates security assertion sets from one design to another but relies on manual assertion definition and has only been evaluated for OR1200 processors. The work in [90] introduces a tool named Isadora for generating security properties based on information flow tracking, which is suitable for security verification. However, it produces overly general security properties that merely delineate signal paths between block diagrams. The work in [91] formally verifies information flow security for ARM processor kernel and user modes. Nevertheless, its formal nature introduces scalability issues, making it less applicable to large designs. Moreover, the HARM assertion miner presented in [72] has been explicitly designed for the functional verification of hardware designs. However, it also claims to have the capability to identify Trojans within these designs.

On the other hand, as this thesis leverages assertions for ensuring the safety and

security of autonomous vehicles, this section delves into the pertinent literature in this domain. The study outlined in [92] presents a methodology for constructing and implementing assertion checks to validate the behavior of an autonomous vehicle. They have devised procedures for translating driving codes of practice into formal logical expressions that can be automatically monitored by computers, either through direct translation or physical modeling. However, the assertions generated in this study are mainly in the form of Structured Query Language (SQL), rendering them unsuitable for lower levels of abstraction, such as the hardware level. In [95], formal verification and simulation-based verification techniques have been integrated to demonstrate a proof-of-concept application for a basic autonomous vehicle. The correctness of the proposed approach has been formalized and verified through interactive theorem proving with the Prototype Verification System. However, this study solely establishes the correctness of AV operation without providing any assertions, either automatically or manually. The approach introduced in [118] introduces a unified scenario coverage framework capable of offering a formal assessment of safety verification for Level 3 and Level 4 autonomy in AVs. Level 3 autonomy refers to vehicles capable of performing all driving tasks that might require human intervention under specific situations and conditions [119, 120]. Level 4 autonomy signifies vehicles capable of executing all driving tasks and functions under particular conditions or environments without human intervention, even in challenging driving scenarios [121].

The study described in [122] utilizes an assertion-based monitoring framework to assess the correctness of a Distributed System Interface (DSI3) mixed-signal protocol implementation within a modern airbag SoC application for autonomous vehicles. Meanwhile, the paper referenced in [123] tackles the challenge of generating efficient tests for simulation-based verification of autonomous vehicles using software testing agents. Specifically, this research concentrates on crafting tests capable of triggering the precondition of an assertion. Additionally, several other studies in the current literature, such as the works presented in [124–128] have focused on the verification of autonomous vehicles. Nevertheless, these investigations predominantly rely on formal or simulation-based verification techniques and do not offer an automated approach for AV verification using assertions.

Moreover, in the state-of-the-art, there exists a gap in leveraging Assertion-Based Verification for the purposes of security and attack detection in Autonomous Vehicles. Although the studies mentioned above have utilized verification techniques such as formal methods, simulation-based approaches, or ABV, they have not applied these methodologies for detecting security attacks. Detecting various types of attacks, such as adversarial cyber-attacks targeting autonomous vehicles, poses a significant challenge in ensuring the safety and security of these systems. Given the efficacy of assertions in ABV for identifying errors and bugs within the realm of verification, it would be advantageous to utilize them to address security and attack detection concerns in autonomous vehicles.

4 Automatic Generation of Assertions for Functional Verification

This chapter introduces an innovative data mining-based approach for the automatic generation of assertions. Unlike traditional methods in the literature that rely on FSM-based approaches for assertion mining, this chapter proposes an association rule mining algorithm specifically tailored for ABV and functional verification contexts. The proposed algorithm forms the foundation of ARTmine, an automatic assertion miner that generates assertion sets from simulation traces of designs under verification [129]. ARTmine is capable of mining assertions in the forms of *next[N]*, *Until*, and *Eventually*. Section 4.1 provides the necessary background and concepts for understanding the methodology. Section 4.2 outlines the algorithms and intricacies of ARTmine, while section 4.3 presents the experimental results of it. Finally, section 4.4 offers concluding remarks for the chapter.

4.1 Preliminaries

In this section, the definitions and concepts utilized in this chapter are briefly explained. Alongside the definitions presented herein, definitions 1 to 3 of Chapter 2 have also been referenced in this chapter.

Definition 4 *A **simulation trace** consists of the values of multiple variables of hardware designs ($\{v^1,...,v^m\}$) that have been stored as records of data for a finite sequence of time instants (clock cycles) ($\{t_1,t_2,...,t_n\}$) during the execution of the designs [72].*

Definition 5 ***Temporal pattern next[N]**: next[N] temporal pattern in PSL is in the form of: $always(antecedent \rightarrow next[N]\ consequent)$. This pattern indicates that when antecedent occurs, after N time instant (clock cycle), consequent will occur [109]. N is an integer value and $N > 0$.*

Definition 6 ***Temporal pattern until**: until temporal pattern in PSL is in the form of: $always(antecedent\ until\ consequent)$. This pattern indicates that the antecedent is true and holds up until the time that the consequent happens [109].*

Definition 7 ***Temporal pattern eventually**: Eventually temporal pattern in PSL is in the form of: $always(antecedent \rightarrow eventually!\ consequent)$. This pattern indicates that there exists a future time instant (clock cycle) where the consequent of assertion finally holds [109].*

Definition 8 ***Frequent itemsets** refer to a set of variables in simulation trace that occur with a frequency (minimum support), indicating significant relations and associations between the variables. In this definition, each single variable v is called an item.*

Definition 9 *An **Association Rule (AR)** is defined as an implication of the form $X \rightarrow Y$ where X, $Y \subseteq I$, with $X \cap Y = \varnothing$, and I is an itemset [130–132]. In this definition, X and Y are frequent itemsets.*

Definition 10 ***Support** is a metric in association rule mining that indicates how frequently an itemset appears in the dataset (simulation trace) [132]. The value of support is in the range of 0 and 1. For the rule $X \rightarrow Y$, this value is calculated with the following formula [133]:*

$$Supp(X \rightarrow Y) = P(X \cup Y) \tag{1}$$

In (1), $P(X \cup Y)$ is the probability where $X \cup Y$ indicates that a record contains both X and Y, that is the union of itemsets X and Y.

Definition 11 *The **min_supp** value is the threshold and a minimum value for support to decide whether an itemset is frequent (i.e., occurs frequently in the simulation trace) or not. If the frequency of the itemset is more than this threshold, the itemset is considered a frequent itemset. A higher min_supp value results in the generation of commonly occurring (general) association rules, whereas a lower min_supp value leads to the generation of rarely occurring ARs (corner cases) [132].*

Definition 12 ***Confidence** is an indication of how often the rule has been found to be true [134]. For the rule X $\rightarrow$ Y, confidence value is calculated with the following formula [132, 134]:*

$$Conf(X \rightarrow Y) = P(Y \mid X) \tag{2}$$

The confidence evaluates the certainty degree of the detected association rule. This is calculated as the conditional probability $P(Y \mid X)$, representing the probability that a record containing X also contains Y. This value ranges between 0 and 1.

Definition 13 *The **min_conf** is the minimum value for confidence. The higher value of min_conf leads to fewer but more accurate and valid association rules [132].*

To illustrate these definitions, consider the example provided in Table 2. This table displays a list of records in a simulation trace, each identified by an ID (*e.g.*, T1 in the *'Record ID'* column) and featuring a set of variables (items) per record (*e.g.*, A, B, C for T1 in the *'Variables'* column).

Table 2: List of records

Record ID	Variables
T1	A, B, C
T2	A, C, D
T3	A, C, E
T4	B, E, F
T5	B, C, D, E, F

In this example, record T1 shows an itemset that contains items A, B, and C. The support value for item A is calculated as $3/5 = 0.6$ since it appears in 3 out of 5 records. Furthermore, the support value for the association rule $A \rightarrow B$ is computed as $1/5 = 0.2$ as these two items co-occur in only record T1. Additionally, the confidence value for the association rule $A \rightarrow B$ in this example is calculated as $1/3 = 0.2$. This is because A and B have occurred together in only 1 record (T1) and there are 3 records in which A has occurred.

4.2 Proposed Methodology

Fig. 3 illustrates the general flow of ARTmine. ARTmine begins by taking a simulation trace (Definition 4) of the design under verification as input. The resulting output includes a set of temporal assertions, namely *next[N]* (Definition 5), *until* (Definition 6), and *eventually* (Definition 7). These temporal assertions are then incorporated into the DUV for use in the verification process. The first phase of ARTmine, illustrated

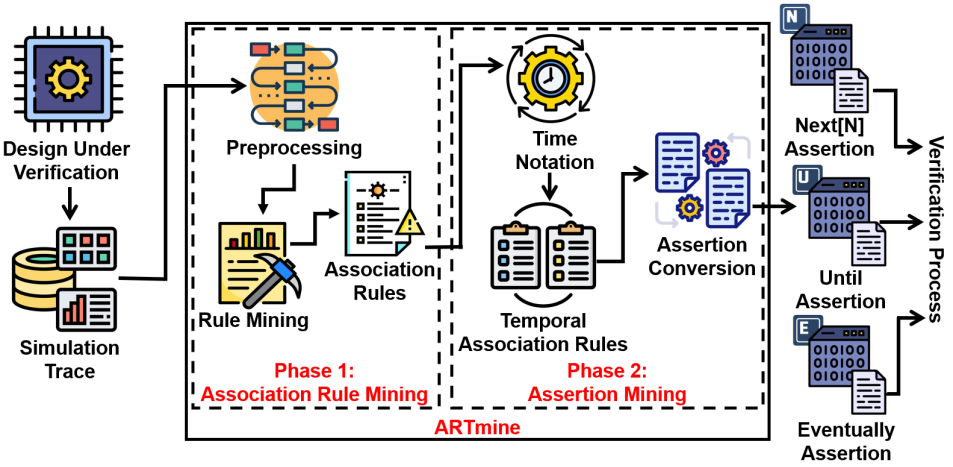


Figure 3: General flow of ARTmine

in Fig. 3, is the *Association rule mining*, which encompasses two steps: *preprocessing* and *rule mining*. Subsequently, phase 2 is *Assertion mining* which consists of two steps: *Time notation* and *Assertion conversion*.

During the *Association rule mining* phase, the simulation trace of the design under verification is initially preprocessed to prepare the data. Afterward, a procedure is applied to the preprocessed data to mine all association rules (Definition 9) derived from the simulation trace.

In the second phase, known as *Assertion mining*, the association rules derived from the initial phase are passed to the *Time notation* step. Here, these extracted association rules are integrated with the concept of time to formulate suitable time-integrated rules (temporal association rules). These rules serve as the input for the *Assertion conversion* step. Subsequently, the *Assertion conversion* step transforms the rules from the preceding step into assertions. The generated assertions are now ready for use in the verification process. In the following subsections, each phase of the method is discussed in detail.

4.2.1 Association Rule Mining

The main objective of *Association rule mining* phase is to first preprocess the simulation trace and second, mine the association rules from the preprocessed data. Traditional association rule mining algorithms such as Apriori and FP-growth [132] often lack the ability to integrate temporal aspects necessary for extracting crucial temporal patterns in assertion-based verification. To overcome this limitation and introduce a time-aware method for association rule mining in ABV, Algorithm 1 is proposed. This algorithm delineates the entire procedure in the *Association rule mining* phase.

Lines 1 to 10 of Algorithm 1 pertain to its initialization. 'N' denotes the time instant used for mining the *next[N]* pattern, while $\mathcal{ST}$ represents the simulation trace from which association rules are to be mined. The minimum support threshold is referred to as min_supp (Definition 11), and the minimum confidence threshold is denoted as min_conf (Definition 13). In this algorithm, $\mathcal{FI}$ represents a set of frequent itemsets (Definition 8). Moreover, α and β signify the values of a variable at consecutive time instants in the simulation trace (*e.g.*, time instants t1 and t2). The output of the

Algorithm 1 ARTmine – Association Rule Mining

1: Inputs: ▷ initialization
2: N = time instant;
3: simulation trace $\mathcal{ST}$;
4: minimum support threshold min_supp;
5: minimum confidence threshold min_conf;
6: Set of frequent itemsets $\mathcal{FI}$;
7: α = value of variable a in time t1;
8: β = value of variable a in time t2;
9: Output:
10: Set of association rules $\mathcal{R}$;
11: **case** (next[N]): ▷ preprocessing of next[N]
12: **for** all records of $\mathcal{ST}$:
13: **move** each output of $\mathcal{ST}$, N records to up;
14: **store**(moved $\mathcal{ST}$);
15: **go to** line 32;
16: **case** (until): ▷ preprocessing of until
17: $\mathcal{F} = (\beta - \alpha)/\alpha$;
18: **if** $\mathcal{F} = 0$ or $\mathcal{F} = undefined$: ▷ $\mathcal{F} = 0 \div 0 = undefined$
19: $\beta = 0$;
20: **if** $\mathcal{F} = \infty$:
21: $\beta = 0.5$;
22: **if** $\mathcal{F} = -1$:
23: $\beta = -1$;
24: **store**(modified $\mathcal{ST}$);
25: **go to** line 32;
26: **case** (eventually): ▷ preprocessing of eventually
27: **for** every input of the $\mathcal{ST}$ in time t:
28: **for** every output from time(t+1) to time(t+N):
29: **move** the output to the front of the input in time t;
30: **store**(moved $\mathcal{ST}$);
31: **go to** line 32;
32: **initialize** $\mathcal{FI}$ to be an empty set; ▷ association rule mining
33: **generate** frequent itemsets of size 1 and add them to $\mathcal{FI}$;
34: **while** $\mathcal{FI}$ is not empty do:
35: **generate** candidate itemsets Ck+1 of size k+1 by joining frequent itemsets of size k;
36: **for** each record in $\mathcal{ST}$:
37: **count** the support for each candidate itemset in Ck+1;
38: **prune** the candidate itemsets in Ck+1 that is not equal to the minimum support threshold min_supp;
39: **add** the remaining candidate itemsets to $\mathcal{FI}$;
40: **Increment** k;
41: **generate** association rules from the frequent itemsets in $\mathcal{FI}$:
42: **for** each frequent itemset X in $\mathcal{FI}$:
43: **generate** all non-empty subsets Y of X;
44: **for** each subset Y:
45: **generate** the rule Y => X-Y;
46: **calculate** the support and confidence of each rule;
47: **prune** the rules that do not meet the minimum confidence threshold min_conf;
48: **add** the remaining rules to $\mathcal{R}$;
49: **return** $\mathcal{R}$;

algorithm is a set of association rules ($\mathcal{R}$) (Definition 9). Following that, lines 11 to 14 comprise the preprocessing steps for mining association rules for the *next[N]* pattern. Lines 16 to 24 in Algorithm 1 handle the preprocessing for the *until* pattern and lines 26 to 30 perform the same for the *eventually* pattern. After preprocessing the $\mathcal{ST}$ for each pattern, lines 32 to 49 identify frequent itemsets in the simulation trace and subsequently mine association rules.

Preprocessing of Simulation Trace – next[N]

In Algorithm 1, lines 11 to 13 represent the preprocessing of the simulation trace for the *next[N]* pattern. To elucidate the algorithm's hypothesis, consider a rule in the form of $antecedent \rightarrow next[N]consequent$. To prepare the simulation trace for mining this temporal pattern, all the output of the simulation trace is moved N records above its original position, while the inputs remain unchanged. The simulation trace is modified in this way since the corresponding output of input variables in a sequential hardware design may occur in the simulation trace N time instants later. This modification ensures the correct alignment of outputs with their corresponding inputs, facilitating accurate temporal analysis and also the extraction of patterns for various N time instants (clock cycles). In line 14, the preprocessed simulation trace is stored for mining the *next[N]* temporal pattern using lines 32 to 49 of the Algorithm 1.

In Fig. 4, an example of preprocessing for the *next[2]* pattern is illustrated. The simulation trace in Fig. 4.1 undergoes preprocessing by moving the output parts 2 time instants above their original positions, resulting in the modified simulation trace shown in Fig. 4.2. The figure represents true values with 'T' and false values with 'F'. The simulation trace comprises 5 records, categorized into input and output variables. Each variable is assigned its corresponding value at each time instant. For example, the first row indicates that v2 equals 01 at time t0 and 11 at time t1. The last two records in Fig. 4.2 are marked as 'NA' (not available) due to the absence of data after time instant t4 to be moved in front of these two records.

Notably, ARTmine primarily emphasizes mining essential temporal patterns such as *next[N]*, which are of significant importance in ABV. However, the method is readily extensible to other temporal patterns, such as *before[N]*, owing to the symmetry shared between these two patterns [135].

Time	Inputs			Outputs	
	v1	v2	v3	v4	v5
t0	001	01	F	11	T
t1	001	11	F	10	T
t2	101	00	T	01	T
t3	110	10	F	10	F
t4	000	01	T	11	F

(1)

moving 2 time instants

Preprocessed

Time	Inputs			Outputs	
	v1	v2	v3	v4	v5
t0	001	01	F	01	T
t1	001	11	F	10	F
t2	101	00	T	11	F
t3	110	10	F	NA	NA
t4	000	01	T	NA	NA

(2)

Figure 4: (1) Simulation trace (2) Preprocessed simulation trace

Preprocessing of Simulation Trace – until

Lines 16 to 24 in Algorithm 1 outline the details of preprocessing for the *until* pattern (Definition 6). To elucidate the algorithm's hypothesis, consider a temporal pattern $antecedent \rightarrow until\ \ consequent$, where $antecedent$ encompasses input variables of

the simulation trace, and $consequent$ represents an output variable of the simulation trace. The preprocessing step explores the points in the simulation trace where the value of an input variable undergoes a change and subsequently identifies the corresponding output. This information is derived through the execution of lines 17 to 23 in Algorithm 1. In this algorithm, α and β denote the values of a variable at two consecutive time instants (*e.g.*, t1 and t2).

The result of the equation in line 17 of the algorithm is stored in $\mathcal{F}$. Using this value, a mapping is performed on the simulation trace as described below to investigate the changes in variable values within the simulation trace according to the definition of the *until* pattern:

- the first row of the simulation trace remains unchanged.
- if $\mathcal{F} = 0$ or $\mathcal{F} = undefined$, β is mapped to 0, indicating no change in the values of variables (α and β).
- if $\mathcal{F} = \infty$, β is mapped to 0.5, indicating a change from 0 (α) to 1 (β).
- if $\mathcal{F} = -1$, β is mapped to -1, indicating a change from 1 (α) to 0 (β).

The mapped values (*i.e.*, 0, 0.5, and -1) serve solely as indicators for ARTmine to enhance the detection of changes in the simulation trace and facilitate the categorization of these distinct changes.

Fig. 5 illustrates an example of mapping performed on a simulation trace, as described in lines 17 to 23 of Algorithm 1. Additionally, for further clarification, α and β are depicted in this figure. For instance, at time t3, the value of v2 is mapped to 0.5 since $(\beta - \alpha)/\alpha$ = (1-0)/0 = ∞.

While ARTmine primarily targets mining crucial temporal patterns in ABV, it can be easily expanded to include other patterns such as *release* [135] with minor adjustments to its algorithm. This expandability arises from the similarities shared between *release* and *until* patterns. Specifically, while the *until* pattern dictates that the consequent must hold until the antecedent becomes true, the *release* pattern mandates that the consequent must persist continuously until the antecedent becomes true [135].

Time	Inputs			Outputs	
	v1	v2	v3	v4	v5
t0	0	0	0	0	1
t1	0	0	1	0	1
t2	0 α	0	1	1	1
t3	0 β	1	1	1	0

Mapping

Time	Inputs			Outputs	
	v1	v2	v3	v4	v5
t0	0	0	0	0	1
t1	0	0	0.5	0	0
t2	0	0	0	0.5	0
t3	0	0.5	0	0	-1

Figure 5: An example of mapping for until pattern

Preprocessing of Simulation Trace – eventually

Algorithm 1 undertakes preprocessing of the simulation trace for the *eventually* pattern in lines 26 to 31. To elucidate the algorithm's hypothesis, consider a rule in the form of $antecedent \rightarrow eventually!\ consequent$, where $antecedent$ encompasses input variables of the simulation trace, and $consequent$ represents an output variable of the simulation trace. To prepare the concept of time for the *eventually* pattern according to Definition 7, for each row of inputs at time t in the simulation trace, all outputs from time t+1 to t+n are moved to the front of the input at time t.

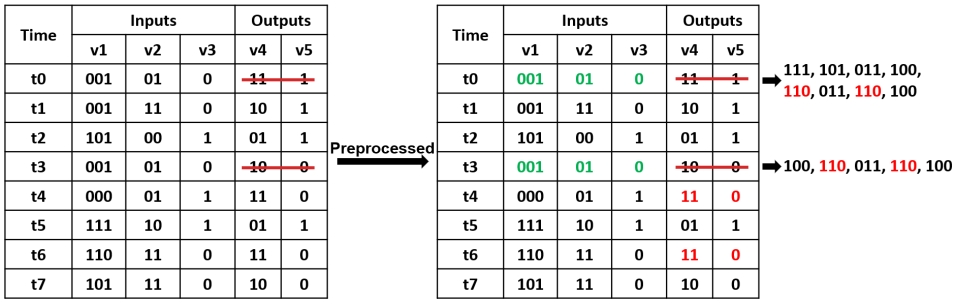


Figure 6: An example for preprocessing of eventually pattern

Fig. 6 illustrates an example of preprocessing the simulation trace for the *eventually* pattern, corresponding to the rule $001010 \rightarrow 110$. In this rule, 001010 appears in the simulation trace at time instants t0 and t3. Consequently, for time instant t0, after preprocessing, in addition to the output at this time instant, the outputs from time t1 to t7 are moved to the front of the time t0. This has happened similarly for the outputs at time instant t3. According to the definition of *eventually*, when 001010 occurs, there exists a future time instant (*i.e.*, t4 and t6) where the consequent (110) finally holds.

Rule Mining – next[N], until, and eventually

Following the preprocessing of the simulation trace in the previous steps to handle temporal patterns *next[N], until*, and *eventually*, the resulting preprocessed simulation trace is then fed into lines 32 to 49 of Algorithm 1 to extract association rules for these three patterns. This segment of the algorithm is executed uniformly across all pattern types.

The lines 32 to 40 of Algorithm 1 iteratively generate frequent itemsets ($\mathcal{FI}$) (Definition 8) of various sizes (1-itemsets, 2-itemsets, etc.) until the $\mathcal{FI}$ list is empty. The algorithm specifically mines frequent itemsets whose support values (Definition 10) exceed the min_supp value (Definition 11), while pruning the others. In this algorithm, 1-itemsets comprise individual variables of the simulation trace, 2-itemsets represent pairs of variables, etc.

After extracting the frequent itemsets and adding them to the $\mathcal{FI}$ list, lines 41 to 49 of the algorithm mine association rules from the $\mathcal{FI}$ list. To elucidate these lines of the algorithm, let's consider an example where $\mathcal{FI}$ comprises 4-itemsets of {A, B, C, D}. To generate association rules from this frequent itemset, all non-empty subsets of it are considered. These subsets include 1-itemsets of {A}, {B}, {C}, {D}, 2-itemsets of {A, B}, {A, C}, {A, D}, {B, C}, {B, D}, {C, D}, as well as the 3-itemsets of {A, B, C}, {A, B, D}, {A, C, D}, {B, C, D}, and the 4-itemset of {A, B, C, D}. Subsequently, for each non-empty subset Y, an association rule of the form Y $\Rightarrow$ X-Y is generated. This approach allows the algorithm to explore all possible combinations of items within the frequent itemset to identify significant associations between different sets of items. For instance, if X = {A, B, C, D} and Y = {A, B, C}, then the rule {A, B, C} $\Rightarrow$ {D} is generated. Here, the antecedent of the rule (Y) is the subset {A, B, C}, while the consequent of the rule (X-Y) is the set difference between the frequent itemset {A, B, C, D} and the antecedent {A, B, C}, which is {D}. This iterative process is applied to each non-empty subset of $\mathcal{FI}$, yielding a set of association rules. Finally, each association rule is evaluated for its support and confidence, with those below the min_conf threshold (Definition 13) being pruned.

In this part of the algorithm, ARTmine optimizes the generation of assertions by thoroughly inspecting non-empty subsets of itemsets, exploring various combinations, and selectively pruning those that are subsets of each other and also fail to meet the min_supp and min_conf thresholds. This process minimizes the redundancy of assertions while ensuring the creation of valid and accurate assertion sets with minimal overhead.

Increasing the min_supp value yields fewer assertions that describe more general design behavior, while decreasing it results in assertions covering rare design behavior (corner cases). Similarly, raising the min_conf value generates fewer but more valid assertions. Valid assertions are those that will not be violated during the simulation with various scenarios. Utilizing these values in ARTmine facilitates an effective verification process. In this chapter, min_supp and min_conf are set to 0.01 and 1, respectively, aiming to uncover corner cases while achieving high design behavior coverage (details are presented in section 4.3).

At this point, following the completion of association rule mining for all three patterns (*i.e.*, *next[N], until*, and *eventually*), these rules constitute the foundational components of the *Assertion mining* phase.

4.2.2 Assertion Mining

This phase comprises two steps: *Time notation* and *Assertion conversion*, which are presented in the following:

Time Notation

In this step, ARTmine incorporates the concept of time into the *association rules* generated in the first phase, resulting in a set of *temporal association rules*. Algorithm 2 illustrates the time notation process for all three temporal patterns, as elaborated in the subsequent subsections. The first seven lines of the algorithm pertain to its initialization, wherein $\mathcal{U}$ denotes a set of mined association rules, and $\mathcal{R}$, $\mathcal{R}'$, and $\mathcal{R}''$ represent three distinct association rules.

Time Notation – next[N]

After mining association rules in the initial phase (section 4.2.1), the method furnishes a set of rules in the general form of $antecedent \rightarrow consequent$. Afterward, in the *Time notation* step, ARTmine determines to which temporal pattern each extracted rule belongs. Consequently, it assigns the corresponding time label to the rule using Algorithm 2.

In line 9 of the algorithm, ARTmine detects which rules are associated with the *next[N]* pattern. Subsequently, line 10 assigns the corresponding N to *next[N]* in the rule $antecedent \rightarrow next[N]\ consequent$. If the antecedent value matches an input in the simulation trace, and the consequent value has already been moved to another record of it ('move_flag $==$ true'), the rule is labeled as a *next* temporal association rule. Otherwise, other mined rules are discarded.

Time Notation – until

In lines 13 to 15 of Algorithm 2, the process of identifying the *until* pattern is described. If the value of $\mathcal{F}$ (computed in phase 1) is 0.5 or -1, and for this value, both $antecedent \rightarrow consequent$ and $consequent \rightarrow antecedent$ rules were obtained in the first phase, the rule is labeled as an *until* temporal association rule. Subsequently, the rule $consequent \rightarrow antecedent$ is eliminated from the set of mined rules. Indeed, the required criterion for identifying a rule as an *until* pattern is the presence of both the

Algorithm 2 ARTmine – Time Notation

1: $\mathcal{U}$ = set of mined association rules; ▷ initialization
2: $\mathcal{R} = antecedent \rightarrow consequent$ in time instant t_i;
3: $\mathcal{R}' = antecedent \rightarrow consequent$ in time instant t_j;
4: $\mathcal{R}'' = consequent \rightarrow antecedent$;
5: $\{\mathcal{R}, \mathcal{R}'\} \in \mathcal{U}$;
6: TID = time instant difference;
7: move_count = number of moved records in the preprocessing phase, it corresponds with N in the next[N] pattern;
8: **for** a rule $\mathcal{R}$ in $\mathcal{U}$: ▷ next[N] time notation
9: **if** ($antecedent$ == an input of preprocessed sim. trace) and (($consequent$ == an output of preprocessed sim. trace) and (move_flag == true)):
10: **label** $\mathcal{R}$ as **next**[move_count] temporal association rule;
11: **else**:
12: **discard** $\mathcal{R}$;
13: **if** (($\mathcal{F}$ = 0.5) or ($\mathcal{F}$ = -1)) and ($\mathcal{R}'' \in \mathcal{U}$): ▷ until time notation
14: **label** $\mathcal{R}$ as **until** temporal association rule;
15: **discard** $\mathcal{R}''$;
16: **for** rules in form of $\mathcal{R}$ and $\mathcal{R}'$ in $\mathcal{U}$: ▷ eventually time notation
17: **if** ($count(antecedent) == count(consequent)$) and (move_flag == true):
18: **for** each corresponding $antecedent$ and $consequent$ in $\mathcal{R}$:
19: TID[$\mathcal{R}$] = time_instant[$antecedent$] - time_instant[$consequent$];
20: **for** each corresponding $antecedent$ and $consequent$ in $\mathcal{R}'$:
21: TID[$\mathcal{R}'$] = time_instant[$antecedent$] - time_instant[$consequent$];
22: **if** (TID[$\mathcal{R}$] < 0) and (TID[$\mathcal{R}'$] < 0) and (TID[$\mathcal{R}$] != TID[$\mathcal{R}'$]):
23: **label** $\mathcal{R}$ as **eventually** temporal association rule;

$antecedent \rightarrow consequent$ and $consequent \rightarrow antecedent$ rules in the set of mined rules of phase 1.

Time Notation – eventually

Lines 16 to 23 in Algorithm 2 pertain to detecting the *eventually* pattern. To accomplish this, ARTmine first computes the occurrences of $antecedent$ and $consequent$ in the preprocessed simulation trace, represented as $count(antecedent)$ and $count(consequent)$, respectively. If these two values are equal, the time instants of their occurrences are captured. Subsequently, ARTmine calculates the difference in time instants (TID in Algorithm 2) between $antecedent$ and $consequent$. If the differences are negative and unequal (line 22), the extracted association rule represents an *eventually* pattern and is labeled as an *eventually* temporal association rule (lines 23). Indeed, the required condition for identifying a rule as an *eventually* pattern is that the TIDs must be negative and unequal.

Assertion Conversion

In the *Assertion conversion* step, the temporal association rules mined by ARTmine are transformed into temporal assertions using the labels assigned in the *Time notation* step (section 4.2.2). While ARTmine provides assertions in the SystemVerilog Assertions (SVA) [136] language, this section elucidates the general structure and format of temporal mined rules in PSL [109] for enhanced comprehension.

The output of the *Time notation* step for temporal association rules labeled as *next[N]* is transformed into the PSL format of "*always(antecedent → next[N] consequent)*". Temporal association rules labeled as *until* are converted to the assertion

"*always(antecedent until consequent)*". Finally, temporal rules with the *eventually* label are changed to the assertion "*always(antecedent → eventually! consequent)*". The generated assertion sets in this phase are now ready for use in the verification process.

4.3 Experimental Results

In this section, a brief overview of mutant analysis is provided, and afterward, the experimental results of ARTmine are presented.

4.3.1 Mutant Analysis

In this thesis, to evaluate the assertions and experiments mutant analysis has been used. Mutation analysis operates based on the concept of generating multiple models of the DUV, each mutated by incorporating a syntactically correct functional alteration (mutant) [137–139]. These mutations are intended to disrupt the behavior of the model and ascertain whether the test suite can identify discrepancies between the original model and the mutated versions [139–142].

Table 3: Example of mutation operators

Source Code	Mutant 1	Mutant 2	Mutant 3

if (a && b)
C = 1;
else
C = 1;
	if (a || b)
	C = 1;
	else
	C = 1;
		if (a && b)
		C = 0;
		else
		C = 1;
			if (a && b)
			C = 1;
			else
			C = 0;

As an illustration, Table 3 demonstrates the creation of three mutants derived from a segment of Verilog code. The initial mutant substitutes the 'and' operator (&&) of the original program with the 'or' operator (||). The remaining two mutants alter two assignment statements. A mutation operator refers to a transformation rule that generates a mutant from the original program [143]. Table 3 showcases only three instances of mutation operators, while several others could be considered. Typical mutation operators are designed to modify variables, expressions, and assignments through replacement, insertion, or deletion operators. Subsequently, once a mutant is generated, a test set is administered to the system, and the outputs of the model without the mutation are compared to those of the model with the mutation.

If differences are detected in these outputs, the mutant is deemed 'killed'; otherwise, it is considered 'survived'. Even after executing all test sets, some mutants may still survive. Consequently, verification engineers may introduce additional test inputs to kill these surviving mutants, thereby improving the quality of the test set and mutant analysis. A mutant that cannot be killed by any input or sequence of inputs is deemed 'equivalent' [142]. Models containing equivalent mutants are syntactically different but functionally equivalent to models without mutants. Automatically detecting equivalent mutants is infeasible, as determining model equivalence is undecidable. Mutation testing yields an adequacy score, referred to as a mutation score, which reflects the quality of the input test set and mutant analysis [142, 143]. The mutation score represents the ratio of killed mutants to the total number of non-equivalent mutants. The objective of mutation analysis is to achieve a mutation score of 100%, indicating that the test set is comprehensive enough to identify all design errors represented by the mutants.

In this thesis, mutant analysis has been conducted to assess the effectiveness of the proposed methods and assertions, using Table 4. To evaluate the assertion sets,

Table 4: Description of injected mutants

Mutation operator types	List of operators
arithmetic operators	$+$, $-$, $\times$, $/$, %
relational operators	$==$, !=, $>$, $<$, $>=$, $<=$
logical operators	&&, $\|\|$
assignment operators	$+=$, $-=$, $\times=$, $/=$, $\%=$, $=$
unary operators	$+$, $-$, $\sim$, !
bitwise operators	$\ll$, $\gg$, &, $\|$, $\wedge$
bitwise assignment operators	$\ll=$, $\gg=$, $\&=$, $\|=$, $\wedge=$

an automatic mutant generator and injector were implemented, according to the details presented in [71, 76, 77, 144]. A comprehensive set of mutants was employed, encompassing the conversion of all operators and bits, which were subsequently injected into the RTL designs. Table 4 provides detailed information about the injected mutants, including the types of mutated operators in the 'Mutation operator types' column and the alterations made to the operators listed in each row of the 'List of operators' column. Additionally, all 0s and 1s were interchanged.

4.3.2 Experimental Results – ARTmine

ARTmine has been implemented in Python and evaluated using several benchmarks developed in Verilog and SystemVerilog Assertions languages. The benchmarks comprise some of the ISCAS'89 designs from [145] and Arb2, Id_stage, Decoder, Controller, and Multdiv from the GoldMine repository in [146]. One benchmark is the Bridge that is used to connect memory and IO. The other benchmarks, Arbiter, and LBDR, are crucial components of an open-source project named NoC router Bonfire [147].

In all the experiments for ARTmine, the minimum support (Definition 11) and minimum confidence (Definition 13) values have been set to 0.01 and 1, respectively. These settings enable the verification engineer to guide ARTmine in mining fewer yet more valid assertions, effectively addressing the corner cases of designs. Additionally, N has been set to 2 for the *next[N]* pattern, but it can be adjusted to other values.

Table 5: Experimental results of ARTmine

Benchmarks	T_Len	Lines	I/O	#A-*N*	#A-*U*	#A-*E*	ET*N*	ET*U*	ET*E*
Arb2	100	28	6	8	0	0	0.001s	0.11s	0.05s
Id_stage	1K	813	82	1793	107	2	13s	6m11s	50s
Decoder	1K	426	4	369	20	1	0.24s	2m	38s
Controller	10K	788	57	748	72	1	12s	5m14s	37s
Multdiv	1K	559	15	348	79	4	49s	6m9s	3m12s
Arbiter	30K	245	22	105	55	3	10s	6m	3m
LBDR	80K	95	13	195	27	3	57s	7m3s	4m
Bridge	100K	196	16	79	36	1	7s	13m	7m
S27	1K	36	5	1	0	0	0.05s	20s	15s
S15850	1K	11247	227	9542	227	3	1m55s	16m42s	5m13s
S35932	1K	39481	355	1084	48	1	54s	9m11s	4m5s
S38417	1K	26190	134	1509	308	12	18s	15m	3m
S38584	1K	22734	342	5093	1460	8	1m10s	22m21s	8m14s

Table 5 presents the experimental results of ARTmine, including the number of mined assertions for *next[N], until*, and *eventually* patterns (columns '#A-*N*', '#A-*U*', and '#A-*E*'), along with their corresponding execution times ('ET*N*', 'ET*U*', and 'ET*E*'). In this table, the 'T_Len' column indicates the length of the simulation traces, while the 'Lines' column displays the lines of code for each benchmark. Furthermore, 'I/O' represents the number of Inputs/Outputs in each benchmark. The experimental results demonstrate that ARTmine is capable of generating a reasonable number of assertions in a suitable amount of time, even for large-scale designs such as the ISCAS'89 benchmarks.

Table 6: Comparison of ARTmine with other assertion miners

Benchmarks	#Mutants	#Assertions			Mutant Detection (%)			Execution time		
		ARTmine	HARM	GoldMine	ARTmine	HARM	GoldMine	ARTmine	HARM	GoldMine
Arb2	10	8	12	9	100	100	100	0.161s	0.49s	2.4s
Id_stage	660	1902	105716	2790	90	80	45	7m14s	13m41s	18m11s
Decoder	400	390	780	418	69	53	40	2m38.24s	4m33s	4m11s
Controller	644	821	13672	1068	91	76	37	6m3s	9m17s	15m51s
Multdiv	1801	431	21620	700	96	88	54	10m10s	17m30s	13m23s
Arbiter	860	163	2017	113	100	66	68	9m10s	20m14s	26m12s
LBDR	632	225	480	100	100	39	18	12m	15m38s	16m10s
Bridge	560	116	1596	806	78	50	31	20m7s	1h15m	36m4s
S27	12	1	1	NS *	100	100	NS *	35.05s	0.56s	NS *
S15850	2642	9772	68174	NS *	80	60	NS *	23m50s	33m09s	NS *
S35932	3700	1133	159315	NS *	52	34	NS *	14m10s	38m18s	NS *
S38417	3022	1829	15889	NS *	68	40	NS *	18m18s	25m12s	NS *
S38584	2905	6561	145925	NS *	84	62	NS *	31m45s	41m37s	NS *

* GoldMine could not provide any solution (assertion) for this benchmark.

Table 6 shows the efficiency of ARTmine compared to the leading assertion miners in the literature, namely HARM [72] and GoldMine [64]. The column '#Mutants' denotes the number of injected mutants for each benchmark. The column '#Assertions' compares the number of assertions generated by ARTmine, HARM, and GoldMine, while the column 'Mutant Detection (%)' represents the percentage of the detected mutants. To mine the assertions for HARM, and GoldMine, the tools available on their repositories in [72] and [146] have been executed. In the experiments, the same simulation traces and designs have been used for all assertion miners, and mutant injection has been performed similarly for all the designs.

As can be seen, ARTmine has generated significantly fewer assertions compared to the other tools in nearly all cases, while demonstrating notably higher effectiveness in mutant detection. The results show that HARM has mined an excessive number of assertions for most benchmarks, which could potentially prolong the verification process. Conversely, GoldMine shows limitations in handling ISCAS'89 benchmarks, as indicated by 'NS' (No Solution) in Table 6. Unlike other tools, GoldMine exclusively operates with Verilog designs and cannot handle simulation traces or Value Change Dump (VCD) files [64]. Furthermore, it is limited to mining assertions solely from designs in the RTL format [64], rendering GoldMine incapable of generating any assertions for ISCAS'89 benchmarks which are implemented in the gate-level format. However, ISCAS'89 benchmarks have been employed to evaluate ARTmine with large-scale designs and compare it to the latest miner in the literature, HARM [72]. Moreover, ARTmine demonstrates shorter execution times in contrast to HARM and GoldMine.

4.4 Conclusions

Considering the shortcomings of the assertion miners in the state-of-the-art, in this chapter, a set of association rule mining algorithms were introduced that form the foundation of the automatic assertion miner, ARTmine. This assertion miner efficiently mines accurate assertion sets, including *next[N]*, *until*, and *eventually* temporal patterns. Experimental results demonstrate that ARTmine surpasses other leading assertion miners such as HARM [72] and GoldMine [64] by detecting more injected mutants with fewer assertions. According to the experimental results, on average, ARTmine detects 20% more mutants than HARM and 55% more than GoldMine.

5 Automatic Evaluation and Minimization of Assertions

The assertions generated by assertion miners often exhibit drawbacks such as redundancy and inconsistency. Moreover, these assertions may feature excessively long antecedents, resulting in low readability and complicating the verification process for engineers. Conversely, there remains a need for a method to accurately evaluate and rank assertion sets based on their quality. To address these shortcomings, this chapter first introduces an innovative data mining-based algorithm called Dominance to evaluate the quality of assertions and rank them accordingly. Additionally, an assertion minimizer named IMMizer is proposed to significantly minimize the assertion sets. Utilizing Dominance and IMMizer by verification engineers has the potential to substantially lower overall verification costs. Section 5.1 provides detailed insights into Dominance, while section 5.2 elucidates IMMizer.

5.1 Automatic Evaluation of Assertions – Dominance

Dominance was initially defined within the domain of data mining to evaluate the quality of association rules (Definition 9) [148, 149]. In this section, Dominance is adapted and, for the first time, utilized within the context of ABV and assertion evaluation. Dominance takes the assertion sets generated by assertion miners as its input and produces an evaluated set of assertions as its output. In the following, the details of the Dominance algorithm are elucidated.

5.1.1 Dominance Algorithm

To delineate the Dominance algorithm, each assertion within an assertion set is considered as an association rule (Definition 9), and $\mathcal{R}$ represents the set of all mined rules, denoted by $\mathcal{R} = \{r_1,\ r_2,\ \ldots,\ r_n\}$. These rules undergo evaluation based on a set $\mathcal{M}$ of various measures such as *Support, Correlation Coefficient (CC)*, and *IS*, represented by $\mathcal{M} = \{m_1,\ m_2,\ \ldots,\ m_n\}$. Consequently, the relation $\Omega = (\mathcal{R}, \mathcal{M})$ creates a table (similar to Table 7) comprising the rules in $\mathcal{R}$ and the measures in $\mathcal{M}$. Furthermore, the value of the measure m for the rule r is denoted as $r[m]$ such that $r \in \mathcal{R}$ and $m \in \mathcal{M}$.

Table 7: Relation between the rules and measures (Ω)

Rules	Support	CC	IS
r_1: a→d	0.20	0.67	0.02
r_2: b→c	0.10	0.50	0.00
r_3: c→a	0.10	0.50	0.02
r_4: a→b	0.20	0.40	0.10
r_5: b→d	0.20	0.33	0.02
r_6: d→c	0.20	0.33	0.10
r_7: c→b	0.10	0.20	0.01
r_8: b→a	0.10	0.17	0.02

The evaluation of rule quality based on each measure differs from the others, potentially resulting in varying rankings and quality assessments for the rule sets. For instance, Table 7 presents eight different rules along with the values of their measures for *Support*, *CC*, and *IS*. In this table, according to the *Support* measure, r_1, r_4, r_5, and r_6 are deemed the best rules, while according to the *CC* measure, only r_1 is considered

superior. In terms of the *IS* measure, r_4, and r_6 emerge as the top rules. To address this issue, Dominance is defined on two levels: *value dominance* and *rule dominance*.

Definition 14 ***Value dominance****: Considering two values of a measure m from the set $\mathcal{M}$ corresponding to two rules r and r' from the set $\mathcal{R}$, we say that $r[m]$ dominates $r'[m]$, denoted by $r[m] \succeq r'[m]$, iff $r[m]$ is preferred to $r'[m]$. If $r[m] \succeq r'[m]$ and $r[m] \neq r'[m]$ then, $r[m]$ strictly dominates $r'[m]$, denoted by $r[m] \succ r'[m]$. Consequently, the dominant values will be retained, and the others will be removed.*

Definition 15 ***Rule dominance****: Considering two rules r and $r' \in \mathcal{R}$, we define the dominance relationship with respect to the set $\mathcal{M}$ of measures as follows:*

- *r dominates r', denoted by $r \succeq r'$, iff $r[m] \succeq r'[m],\ \forall\ m \in \mathcal{M}$.*
- *If $r \succeq r'$ and $r' \succeq r$, i.e., $r[m] = r'[m], \forall\ m \in \mathcal{M}$ then r and r' are equivalent, denoted by $r \equiv r'$.*
- *If $r \succeq r'$ and $\exists\ m \in \mathcal{M}$ so that $r'[m] \succ r[m]$, then r' is strictly dominated by r, denoted by $r \succ r'$.*

Furthermore, the *strict dominance* relationship fulfills the following properties:

- **Irreflexive**: $r \not\succ r$, *i.e.*, $r \succ r$ is false for each $m \in \mathcal{M}$,
- **Transitive**: $\forall\ r,\ r'$ and $r'' \in \mathcal{R}$, if $r \succeq r'$ and $r' \succeq r''$ then $r \succeq r''$.

As a consequence of *rule dominance*, the dominant rules will be retained while the others will be eliminated. In other words, according to the Dominance algorithm, if a rule r dominates another rule r', it signifies that r is equal to or superior to r' across all measures in the set $\mathcal{M}$.

As an illustrative example, in Table 7, rule r_3 strictly dominates r_2, because r_3[*Support*] $\succeq r_2$[*Support*], r_3[*CC*] $\succeq r_2$[*CC*], and r_3[*IS*] $\succ r_2$[*IS*]. Therefore, the Dominance algorithm preserves rule r_3 and eliminates rule r_2. By applying the Dominance algorithm to all rules in Table 7, the resulting table will be equivalent to Table 8. This indicates that no rule in $\mathcal{R}$ dominates either r_1 or r_4. Consequently, rules r_1 and r_4 are regarded as high-quality rules to retain.

Table 8: Dominance results

Rules	Support	CC	IS
r_1: a→d	0.20	0.67	0.02
r_4: a→b	0.20	0.40	0.10

According to the Definition 3 and Definition 9, the structures of an assertion and an association rule exhibit similarities. Moreover, similar to the evaluation of the quality of association rules, various measures are considered in evaluating the quality of an assertion. Therefore, the Dominance algorithm is adaptable to the context of assertion evaluation. It assists verification engineers in comparing assertions based on the value of each important metric and measure and evaluate them in a unified manner. Section 5.3.1 illustrates the efficiency of the Dominance algorithm in evaluating the quality of assertions.

5.2 Automatic Minimization of Assertions – IMMizer

This section introduces IMMizer, an innovative method designed to identify redundant and inconsistent assertions and subsequently minimize them. IMMizer takes assertion sets generated by various assertion miners as input and produces a set of minimized assertions with reduced redundancy and inconsistency as output. The IMMizer process comprises three primary steps: Assertion Classification, Deduction based on Contradictory Terms, and Assertion Composition. These steps are discussed in detail in the following subsections.

5.2.1 Preliminaries

The following subsection briefly explains the definitions and concepts necessary for a comprehensive understanding of IMMizer. Initially, two examples of assertions in SVA language are showcased in Listing 1 to elucidate the ensuing definitions more effectively. In this listing, property p1 signifies that whenever a && !b occurs, t1 occurs at the same time instant (represented by |->). Property p2 demonstrates that whenever a && b && !c occurs, t2 occurs after 2 clock cycles (denoted by ##2).

Listing 1: Example of assertions

```
property p1;
   @(posedge clk) a && !b |-> t1;
endproperty

property p2;
   @(posedge clk) a && b && !c |-> ##2 t2;
endproperty
```

Definition 16 ***Different Terms*** *is defined as a set denoted by DT, representing propositions that differ in the antecedent of an assertion compared to another assertion.*

In Listing 1, *(!b -)* and *(b !c)* in properties p1 and p2, respectively, are considered as elements of *DT*. In property p1, since there is neither proposition c nor !c, it is considered as don't care (denoted by -). Thereby, the set *DT* is as follows:

DT = {(!b -), (b !c)} = {(!b c), (!b !c), (b !c)}, where two different values, c and !c are assigned to '-'. Note that *a* is not considered as an element of *DT* since it exists in both properties p1 and p2.

Definition 17 ***Sum-Of-Products (SOP)*** *is a type of Boolean algebra expression where different product inputs are summed together [150].*

For example, considering the set *DT* computed in *Definition 16* for the properties of Listing 1, the *SOP* expression is as follows:

SOP(DT) = (!b c) + (!b !c) + (b !c) = !b(c + !c) + (b !c) = !b + (b !c)

Definition 18 ***Contradictory Terms*** *are defined as the complement of the SOP of the set DT, represented by* $SOP(DT)^C$.

The complement for the calculated *SOP* in *Definition 17* is as follows:

$SOP(DT)^C = (!b + (b\ \ !c))^C = b(!b + c) = (b\ \ c)$

To provide further clarity on why calculating the *SOP* and its complement is necessary in IMMizer, let's consider two terms, β and γ, and the Universal set *U* comprising all distinct combinations of these two terms:

$U = \{(\beta\ \ \gamma), (\beta\ \ !\gamma), (!\beta\ \ \gamma), (!\beta\ \ !\gamma)\}$

Let's consider the set of distinct combinations produced by each DT as follows:

$DT = \{(\beta\ \ \gamma), (\beta\ \ !\gamma), (!\beta\ \ !\gamma)\}$

The *SOP* for the set DT is calculated as:

$SOP(DT) = (\beta\ \ \gamma) + (\beta\ \ !\gamma) + (!\beta\ \ !\gamma) = \beta + (!\beta\ \ !\gamma)$

To find the *Contradictory Terms*, the complement of *SOP(DT)*, denoted as $(SOP(DT)^C)$, is calculated:

$SOP(DT)^C = (\beta + (!\beta\ \ !\gamma))^C = \ !\beta(\beta + \gamma) = (!\beta\ \ \gamma)$

$SOP(DT)^C$ consists of terms that do not exist in the set DT and can complete the Universal set U such that:

$SOP(DT)^C \notin DT$ and $SOP(DT)^C \cup DT = U$

In other words, $SOP(DT)^C$ represents the *Contradictory Term* that is sought by IMMizer, which are terms not present in DT but necessary to complete U:

$SOP(DT)^C \cup DT = U$

After obtaining the $SOP(DT)^C$ (*Contradictory Term*), IMMizer finally calculates $\sim (SOP(DT)^C)$ (more details in Assertion Composition section):

$\sim (SOP(DT)^C) = \ \sim (!\beta\ \ \gamma) = (\beta\ +\ !\gamma)$ (*i.e.* β or $!\gamma$)

The reason for this calculation is that comparing the set DT with $(\beta + !\gamma)$ reveals that all three terms in DT $((\beta\ \ \gamma), (\beta\ \ !\gamma), (!\beta\ \ !\gamma))$ are included in the new term $(\beta + !\gamma)$. Thus, the new term $(\beta + !\gamma)$ is equivalent to all the previous terms in DT and can be replaced with them by IMMizer.

Structure of generated assertions studied in this work

In this study, IMMizer is employed to analyze various assertions extracted by different assertion miners [42, 65, 66]. Typically, these assertions are generated in diverse formats and languages, including SVA, PSL, and LTL [151]. As one of IMMizer's advantages is its applicability to assertions written in different languages, we provide examples studied in this research to illustrate its effectiveness.

Listing 2 demonstrates an example of assertions generated by the assertion miner introduced in [66]. Listing 3 illustrates an extracted assertion from the work in [65]. Finally, Listing 1 represents the format of assertions produced by the assertion miner in [42]. Each of these assertions describes a specific behavior of the DUV. For instance, property p1 in Listing 2 states that eventually, from time 0 to an infinite clock cycle later, the condition *a==1* will occur. This implies that eventually, from time 0 to an infinite clock cycle later, the condition *!(t1==1)* will occur as a consequence.

Listing 2: Example of generated assertions in [66]

```
property p1;
   (((s_eventually[0:$]( a==1 )))) implies (((s_eventually[0:$](!(t1==1)))));
endproperty
```

Listing 3: Example of generated assertions in [65]

```
property p1;
   @(posedge clk)
   ((reset)
     ##1 (!reset && !(c==’b1) && !(d==’b1))[*2]
     ##1 (!reset && !(a==’b1) && !(c==’b1) && !(d==’b1))
     ##1 ((a==’b1) && !reset && !(b==’b1))
     ##1 ((d==’b1) && !reset && !(a==’b1) && !(b==’b1))
     ##2 ((c==’b1) && !reset && !(b==’b1) && !(e==’b1))
     ##1 (!reset && !(b==’b1) && !(e==’b1))[*8]
     ##5 ((b==’b1) && !reset)
     ##1 ((a==’b1) && (b==’b1) && !reset))
     |-> ##1 t1==’b100;
endproperty
```

5.2.2 Proposed Methodology

In this section, the proposed method for minimizing assertion sets is discussed. As illustrated in Fig. 7, IMMizer comprises the following main steps:

- Assertion Classification
- Deduction based on Contradictory Terms
- Assertion Composition

The initial step (Assertion Classification) involves applying three distinct classifications to the assertions generated by assertion miners. These classifications are conducted according to the similarity of consequents, temporal patterns, and antecedents of assertions (*Definition 3*), respectively. Such classifications aid IMMizer in grouping assertions more uniformly, thereby facilitating easier and faster assertion minimization.

The second step (Deduction based on Contradictory Terms) comprises two substeps: 1) Proposition Extraction, and 2) Proposition Subtraction. This step forms the core of IMMizer, where the main tasks of minimization are executed. Initially, the method conducts 'Proposition Extraction' on the classified assertion sets to analyze propositions (*Definition 2*) within all the antecedents. If there exist *Different Terms (DT)* (*Definition 16*) among the antecedents, 'Proposition Subtraction' is then applied to identify the *Contradictory Terms* (*Definition 18*). This is performed by computing the *Sum-Of-Products* (*SOP*) (*Definition 17*).

In the third step (Assertion Composition), a new set of minimized assertions is generated based on the ingredients provided by the previous steps.

This flow is further elucidated in Algorithm 3 to provide a clearer overview of the proposed method, and is discussed in greater detail in the subsequent subsections.

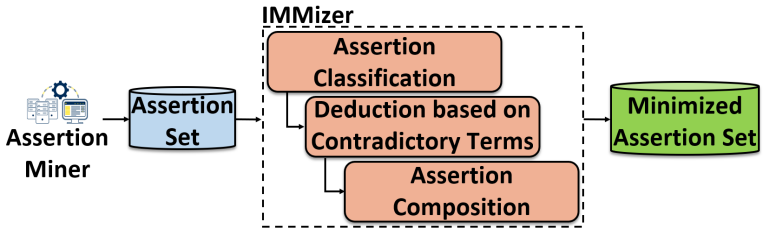


Figure 7: General flow of IMMizer

Algorithm 3 IMMizer

Assertion Classification:
$classified\_assertions \leftarrow$ **classification**$(assertion(consequent))$;
$classified\_assertions \leftarrow$ **classification**$(assertion(temporal\_patterns))$;
$classified\_assertions \leftarrow$ **classification**$(assertion(antecedent))$;
Deduction based on Contradictory Terms:
$similar\_terms \leftarrow$ **explore_similar_terms**$(classified\_assertions(antecedent))$;
$DT \leftarrow$ **explore_different_terms**$(classified\_assertions(antecedent))$;
$DT\_comb \leftarrow$ **different_combinations**(DT);
$Contradictory\_Term \leftarrow$ **SOP**$(DT\_comb)^C$;
Assertion Composition:
if $(Contradictory\_Term == 0)$ **then**
  remove(DT_comb);
else
  $negated\_Contradictory\_Term \leftarrow \sim(Contradictory\_Term)$;
  $minimized\_assertion \leftarrow$ **replace**$(DT, negated\_Contradictory\_Term)$;

Assertion Classification

This step focuses on classifying the initial assertion set to establish the necessary conditions for subsequent steps. Classifying assertions involves grouping similar assertions together, thereby facilitating the identification of redundant and inconsistent assertions. Fig. 8 illustrates the three classifications employed in IMMizer: classification on consequents, temporal patterns, and antecedents.

The initial classification is based on the similarities among the consequents of the assertion sets. This implies that assertions can only be minimized together if they pertain to the same consequent. Otherwise, it is logically untenable to minimize and combine them together, as they describe disparate behaviors of the DUV.

As an illustrative example, in Listing 4, a selection of assertions generated by A-Team [42], an assertion miner used for one of our benchmarks (Arbiter), have been examined. Following the first classification, these assertions are classified into three distinct classes (Class 1, Class 2, and Class 3) as depicted in Listing 5. This classification is determined by the similarity of consequents associated with each class. In these listings, *A*, *!A*, *B*, *C*, *!C*, *D*, and *!D* are included in the antecedent part of the assertions, while *F*, *!F*, and *G* are in their consequents. Additionally, *##2* and *##3* specify the number of clock cycles where the consequent occurs after the antecedent.

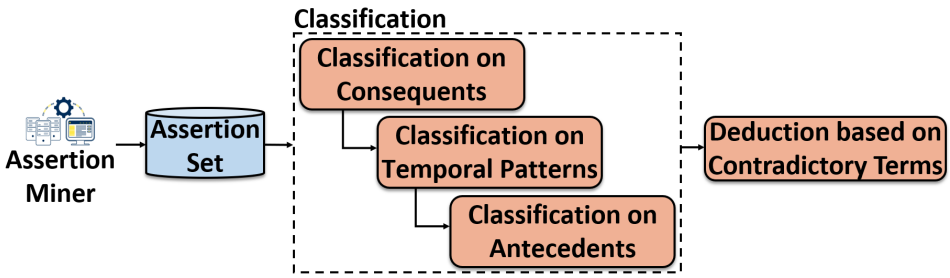


Figure 8: Three different classifications of IMMizer

Listing 4: A group of assertions provided by A-Team [42]

```
1) !A && B && !C |-> F
2) A && D |-> G
3) B && C && !D |-> ##2 G
4) !A && !C |-> F
5) !A && C |-> F
6) B && C |-> ##3 F
7) A && B && C ##1 !A && !C |-> ##2 !F
8) A && !C ##1 !A && D && C |-> ##2 !F
9) A && C ##1 !A && !C |-> ##2 !F
```

Listing 5: First classification on consequents

```
Class 1:
1) !A && B && !C |-> F
4) !A && !C |-> F
5) !A && C |-> F
6) B && C |-> ##3 F
Class 2:
7) A && B && C ##1 !A && !C |-> ##2 !F
8) A && !C ##1 !A && D && C |-> ##2 !F
9) A && C ##1 !A && !C |-> ##2 !F
Class 3:
2) A && D |-> G
3) B && C && !D |-> ##2 G
```

The assertions analyzed in this research exhibit diverse temporal patterns, necessitating classification based on similarities in their temporal patterns to facilitate minimization. Therefore, following the classification based on consequents of assertions, the second classification categorizes the assertions according to their temporal patterns. The results of this classification are presented in Listing 6.

Listing 6: Second classification on temporal patterns (clock cycles)

```
Class 1.1:
1) !A && B && !C |-> F
4) !A && !C |-> F
5) !A && C |-> F
Class 1.2:
6) B && C |-> ##3 F
Class 2.1:
7) A && B && C ##1 !A && !C |-> ##2 !F
8) A && !C ##1 !A && D && C |-> ##2 !F
9) A && C ##1 !A && !C |-> ##2 !F
Class 3.1:
2) A && D |-> G
Class 3.2:
3) B && C && !D |-> ##2 G
```

In the final classification, IMMizer heuristically identifies assertions with the most similarities in their antecedents. Essentially, it detects and classifies assertions with the most similar propositions in their antecedents. Referring back to the illustrative example in Listing 6, Classes 1.1 and 2.1 contain assertions with the most similar propositions in their antecedents among all other assertions. Consequently, assertions in Classes 1.2, 3.1, and 3.2 are unique and devoid of any redundancy or inconsistency. Therefore, the outcome of the final classification is represented by the two gray sections shown in Listing 6.

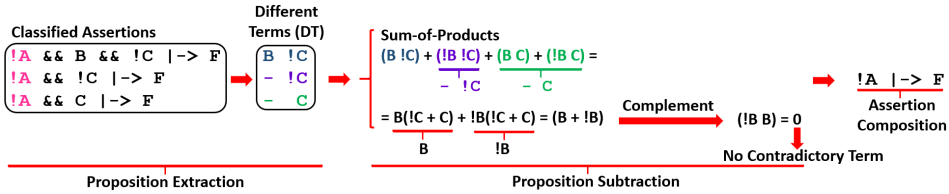


Figure 9: Example 1 for minimizing the assertions

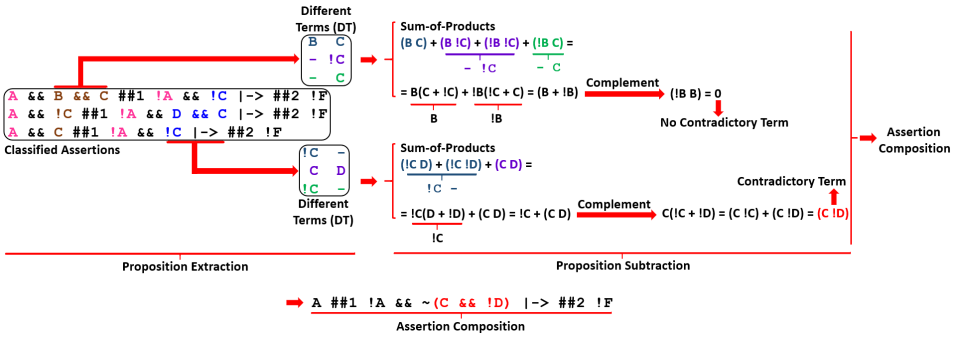


Figure 10: Example 2 for minimizing the assertions

Following the completion of the three classifications and the categorization of assertions for minimization, the subsequent step, namely Deduction based on Contradictory Terms, is implemented on the classified assertions to facilitate their minimization. The ensuing sections delve into these processes in greater detail.

Deduction based on Contradictory Terms

In this section, the process of deduction based on Contradictory Terms is discussed. The objective is to identify behaviors of the DUV that are not covered by the assertion set. To achieve this, assertions grouped within the same class are analyzed (as detailed in the previous section) to identify which DUV behaviors are absent and are not covered by those assertions. These unaddressed behaviors are referred to as *Contradictory Terms*. These *Contradictory Terms* are then utilized to formulate a new assertion set.

This section primarily comprises two steps: 1) Proposition Extraction and 2) Proposition Subtraction. The first step seeks similar terms, as well as *Different Terms i.e.*, dissimilar propositions among the antecedents of assertions. Conversely, the second step focuses on identifying *Contradictory Terms* by computing the *SOP* for dissimilar propositions in the antecedents of assertions. Further details are described in the following.

Proposition Extraction

This step identifies *Different Terms (DTs)* (*Definition 16*), within a classified assertion set. For this purpose, at first, the propositions that are the same in the antecedents of the classified assertion sets are identified. Second, dissimilar propositions among the assertions are determined. In other words, *DT*s are extracted here. In essence, the calculation of *DTs* delineates conditions that result in a 'don't care' situation, denoted by '-'.

As an illustrative example, Fig. 9 depicts the Proposition Extraction, Proposition Subtraction, and Assertion Composition steps for Class 1.1 in Listing 6. Observing

Fig. 9, in the antecedent part of the three assertions within the box labeled *'Classified Assertions'*, *!A* appears in all of them. However, *B* and *C* appear with different values. Thereby, in the provided example, *!A* is retained as is. Subsequently, *B* and *C* are identified as *Different Terms (DT)*.

At this step, all possible combinations generated by the *(DTs)* of any of the assertions are computed. Combinations denote the scenario where, if a proposition does not appear in the classified assertion set, it is marked as a 'don't care' situation, represented by '-'. For example, in the first assertion within the *'Classified Assertions'* box, all propositions are present, hence no 'don't care' is indicated, resulting in *B !C* (displayed in the *Different Terms (DT)* box). Conversely, in the second assertion, neither *B* nor *!B* is present, thus they are considered 'don't care' (-). Consequently, for this assertion, *- !C* is displayed in the *Different Terms (DT)* box, as illustrated in Fig. 9. This procedure is repeated for the third assertion, resulting in the computation of *- C*.

If the classified assertions contain a temporal operator (*e.g.*, *##1*) in the antecedent, the workflow of IMMizer undergoes a slight change. Specifically, IMMizer needs to deconstruct the antecedents based on the number of temporal operators used in each antecedent of the classified assertion. Consequently, *DT*s are identified.

Fig. 10, illustrates an example of a classified assertion set containing a temporal operator in the antecedent. Here, two flows are delineated: one for the antecedent segment preceding the temporal operator *##1*, and another for the segment following *##1*. In the *'Classified Assertions'* box, propositions for the first flow are depicted in brown, while those for the second flow are depicted in blue. Correspondingly, *DT* boxes are computed as described earlier.

Proposition Subtraction

The objective of this step is to identify behaviors of the DUV that are absent in the classified assertion set, *i.e.*, to find the *Contradictory Terms*. This involves, firstly, extracting all possible combinations from the *Different Terms* box (from the previous step). Secondly, extracting *Contradictory Terms* based on the *DT*. To achieve this, the *Sum-Of-Products (SOP)* for the *DT* set is computed, and its corresponding complement is determined. This outcome is subsequently utilized to formulate a new and minimized assertion set.

Returning to the illustrative example depicted in Fig. 9, it is observed that, *DT = {B !C, - !C, - C}*. In this step, all possible combinations resulting from replacing '-' are computed. Therefore, *DT* becomes *{B !C, !B !C, B C, !B C}*. This arises from the generation of *DT* elements corresponding to each assertion: *B !C* for the first assertion, *B !C* and *!B !C* for the second assertion, and *B C* and *!B C* for the third assertion. Consequently, the *SOP* of this set is expressed as detailed in Fig. 9, namely (B !C) + (!B !C) + (B C) + (!B C). Evidently, the final result of the *SOP* simplifies to *(B + !B)*. At this juncture, the *Contradictory Term*, which is the complement of *(B + !B)*, is extracted, *i.e.*, *(!B B) = 0*. In this example, this value equates to 0, indicating that the *DTs* (*B, !B, C*, and *!C*) should be excluded and not incorporated into the minimized assertion.

As mentioned, when dealing with classified assertions containing a temporal operator in their antecedents, IMMizer follows a slightly different workflow. In such instances, each temporal operator's preceding and succeeding components are treated as separate flows. Consequently, *SOP* and *Contradictory Terms* are computed for each flow. Referring back to the illustrative example depicted in Fig. 10, where *##1* is present in the assertions' antecedents, two *DT* boxes are compiled from the assertion set. For the left side of *##1*, *DT = {(B C), (B !C), (!B !C), (!B C)}*. The computed *SOP* for

this segment equals *(B + !B)*, whose complement is *(!B B) = 0*. Consequently, for the left side of *##1*, *B*, *!B*, *C*, and *!C* will be excluded from the final minimized assertion. Conversely, for the right side of *##1*, the *DT* box comprises *{(!C -), (C D), (!C -)} = {(!C D), (!C !D), (C D)}*. The *SOP* and its corresponding complement are *(!C + (C D))* and *(C !D)*, respectively. *(C !D)* signifies that *(C && !D)* is the new term to be replaced with the *DT*s and will be added to the right side of *##1* in the minimized assertion.

Assertion Composition

In this step, the final minimized assertion set is generated. To achieve this, the result of $(SOP(DT)^C)$ is taken into consideration. If the result equals 0, it signifies that the *DT*s should be removed and not included in the final assertion set. Otherwise, if the result of $(SOP(DT)^C)$ equals a set of propositions, a negation (*i.e.*, $\sim$) is applied to $(SOP(DT)^C)$, *i.e.*, $\sim (SOP(DT)^C)$. Subsequently, $\sim (SOP(DT)^C)$ is appended to the final result. In other words, while similar propositions from the classified assertions remain unchanged, all the generated *DT*s are replaced with $\sim (SOP(DT)^C)$.

Returning to the illustrative example depicted in Fig. 9, the result of $(SOP(DT)^C)$ equals *B !B = 0*. Consequently, all *DTs*, *i.e.*, *B !C*, *- !C*, *- C* are eliminated from the assertions within the 'Classified assertion' box. As a result, the final assertion set is minimized to only one assertion (*!A |-> F*). It is evident that the final assertion set contains a significantly reduced number of propositions compared to the classified assertions.

Referring to Fig. 10, two *Contradictory Terms* are identified. The first yields '0', signifying that the *DTs* should be excluded from the final assertion set. However, for the result of the second flow (*C !D*), a negation is applied to *(C !D)*, resulting in *$\sim$(C !D)*. Consequently, the resultant minimized assertion is as follows: *A ##1 !A && $\sim$(C && !D) |-> ##2 !F*.

5.2.3 Discussion on computation time of IMMizer

This section discusses how the proposed method can execute the algorithm within a reasonable timeframe and without imposing additional overhead on the system. As previously mentioned, during the Deduction step, IMMizer must explore various combinations generated by each *DT*, calculate the *SOP*, and its complement to identify *Contradictory Terms*. This process can lead to longer execution times, particularly when there are numerous propositions with varying values in the antecedent of assertions. To address this concern and prevent IMMizer from additional computational comparisons, equations (3) and (4) are utilized.

$$u = \sum_{i=1}^{n} 2^{\#don't\ \ cares} + \sum \#combinations\ \ without\ \ don't\ \ care \tag{3}$$

In equation (3), u represents the total number of all possible combinations that classified assertions can generate. The first segment of this equation, denoted as $\sum_{i=1}^{n} 2^{\#don't\ \ cares}$, pertains to those combinations of *Different Terms* (DT) containing at least one don't care (-). Here, n signifies the number of combinations with don't care in DT with $n \geqslant 1$. If no combinations contain don't care, this portion of the equation yields 0. $\#don't\ \ cares$ denotes the count of don't cares in each combination of DT. Additionally, $\#combinations\ \ without\ \ don't\ \ care$ refers to the number of combinations in DT that do not have any don't care.

$$\mathcal{F} = r - (u - s) \tag{4}$$

In equation (4), s represents the number of repetitive created combinations generated by different terms (*e.g.*, *B !C* in Fig. 9), while r denotes the number of all possible combinations that these terms (*e.g.*, *B* and *C* in Fig. 9) can generate. The value of $\mathcal{F}$ determines whether IMMizer can classify assertions together or not.

If $\mathcal{F} <$ *(#grouped assertions)*, IMMizer can classify them. Otherwise, the assertions cannot be classified with each other for minimization. The reason is that in this case, the number of generated *Contradictory Terms* is more than or equal to the count of assertions that have been grouped together, thereby yielding more or an equal number of assertions compared to the initially grouped ones. Here, *#grouped assertions* represents the number of assertions IMMizer has grouped together based on the highest similarities in the propositions of their antecedent part, aiming at their minimization. In the illustrative example depicted in Fig. 9, u equals 5 ($2^1 + 2^1 + 1$), while s and r equal 1 and 4, respectively. Thus, the value of $\mathcal{F}$ amounts to 0, which is lower than 3 (3 is the number of assertions that have been grouped for minimization (*#grouped assertions*)).

In the proposed minimizer, a pivotal aspect is the initial computation of $\mathcal{F}$ before determining whether to minimize the grouped assertion set. Calculating $\mathcal{F}$ requires only identifying the *Different Terms* and the count of don't cares in each *DT*. Thus, there is no immediate need to identify various combinations generated by each *DT*, perform *SOP* computations, or undertake other processes in the Deduction step. The parameters in $\mathcal{F}$, *i.e.*, r, u, and s can be calculated solely by identifying the *Different Terms (DT)* and the number of don't cares in each *DT*. This approach helps alleviate potential system overheads, particularly in cases with numerous propositions of differing values in the antecedent part of assertions, thereby contributing to reduced execution time.

5.3 Experimental Results – Dominance and IMMizer

In this section, the experimental results concerning the automatic evaluation and minimization of assertions are presented. Subsection 5.3.1 outlines the findings regarding the Dominance algorithm, while subsection 5.3.2 delves into the results for IMMizer.

5.3.1 Experimental Results – Dominance

The efficiency of the Dominance algorithm has been evaluated on the crucial components of an open-source NoC router Bonfire, namely the Arbiter, LBDR, and crossbar switch [147]. The Dominance algorithm has been implemented in Python and the benchmarks have been developed in Verilog and SystemVerilog Assertions languages.

The Dominance algorithm has been applied to the assertions generated by the method presented in [66] for the three benchmarks of Arbiter, LBDR, and crossbar switch. Moreover, the measures utilized by Dominance have been adopted from the study in [79]. Table 9 showcases these results.

Table 9: Result of assertion evaluation after applying dominance algorithm

Benchmarks	#Measures	#Initial assertions	#Selected assertions
Arbiter	3	4175	544
LBDR	3	1546	238
Crossbar Switch	3	500	124

In this table, the column '#Measures' denotes the number of measures reported for each assertion. In this study, these metrics and measures consist of *Support,*

Correlation Coefficient, and *IS*. Interested readers seeking detailed information on how these measures compute the quality of assertions are referred to [79]. The column '#Initial assertions' reports the quantity of initial assertions generated by the assertion miner, while the column '#Selected assertions' indicates the number of assertions chosen by Dominance. This figure represents the count of high-quality assertions after the assertion evaluation. In other words, these assertions are identified as superior for the verification process.

As evident from the data presented in Table 9, the number of initial assertions for Arbiter and LBDR stands at 4175 and 1546, respectively. For the crossbar switch, this figure is equal to 500 assertions. The count of selected assertions deemed as high-quality, attained after applying the Dominance algorithm, equals 544 and 238 for Arbiter and LBDR, respectively. Similarly, for the crossbar switch, this number equals 124 assertions. This highlights that leveraging the Dominance algorithm allows us to achieve nearly the same level of assertion quality while utilizing fewer assertions.

To ensure the effectiveness of Dominance, the results underwent evaluation through mutant analysis. For this purpose, an automatic mutant generator and injector have been implemented, following the details presented in section 4.3.1 of Chapter 4. Moreover, a complete set of mutants, as described in Table 4, has been used.

The Dominance algorithm proposed in this study has been compared with an assertion quality evaluator tool named Shayan [78]. The results of this comparison have been depicted in Fig. 11, Fig. 12, and Fig. 13 for Arbiter, LBDR, and the crossbar switch, respectively. In these figures, the blue bar represents the results of mutant analysis, indicating the number of mutants detected by an assertion set without the application of any assertion evaluator. The orange bar illustrates the number of mutants detected by the high-quality assertions as evaluated by Shayan. Finally, the gray bar depicts the number of mutants detected by the high-quality assertions as evaluated by the proposed Dominance algorithm.

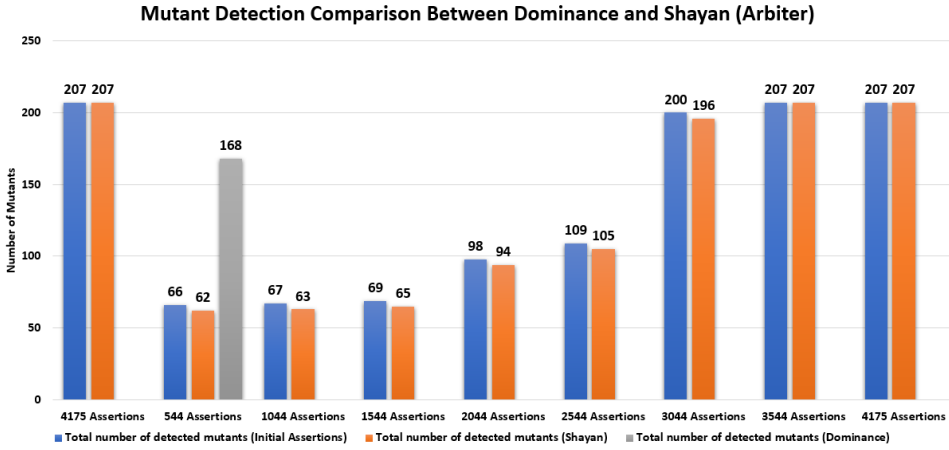


Figure 11: Comparison between Dominance and Shayan (Arbiter)

For the Arbiter benchmark (refer to Fig. 11), the initial set comprises 4175 assertions capable of detecting 207 injected mutants (first set of bars). Upon the application of Dominance to the initial assertions, 544 assertions out of the initial 4175 remain, detecting 168 injected mutants. Conversely, the top 544 assertions identified by Shayan can detect 62 injected mutants (second set of bars). The results indicate that Shayan

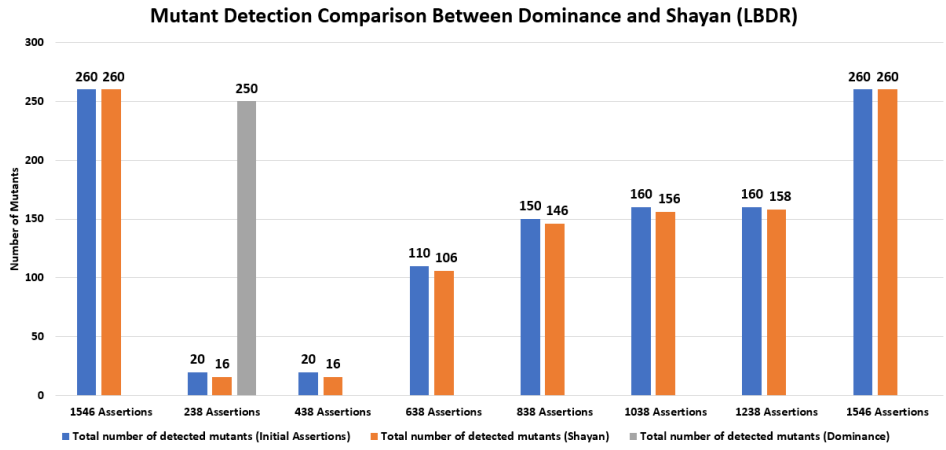


Figure 12: Comparison between Dominance and Shayan (LBDR)

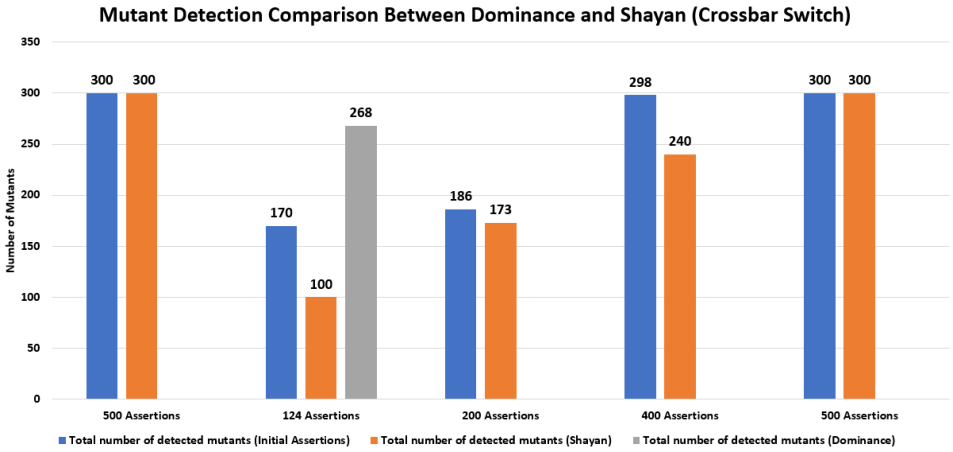


Figure 13: Comparison between Dominance and Shayan (Crossbar Switch)

requires between 2544 and 3044 assertions to detect the same number of mutants as those detected by Dominance (sixth and seventh sets of bars). The remaining bars provide further clarification on the accuracy of the proposed algorithm.

In the case of LBDR (refer to Fig. 12), the initial assertion set comprises 1546 assertions, capable of detecting up to 260 injected mutants (first set of bars). Upon applying Dominance to these initial assertions, 238 assertions that detect 250 mutants are obtained. Conversely, the top 238 assertions identified by Shayan can detect only 16 mutants, indicating that Shayan requires approximately 1546 assertions to detect nearly the same number of mutants as those detected by Dominance (second and last sets of bars).

Fig. 13 illustrates the comparison results between Shayan and Dominance for the crossbar switch benchmark. The initial assertion count for this benchmark is 500, which can detect 300 mutants. Dominance yields 124 high-quality assertions out of the 500 initial assertions, which can detect 268 injected mutants. Meanwhile, Shayan detects 100 mutants, and the initial assertions detect 170 mutants with 124 assertions.

Table 10: Reduction in the number of assertions after applying IMMizer, the report on what percentage of the detected mutants by the initial set are correspondingly detected by the minimized set, and the execution time of IMMizer for each benchmark

Assertion Miners	#Initial Assertions	#Minimized Assertions	Reduction in #Assertions	Mutant Detection	Execution Time
A-Team [42] (Arbiter)	318	198	38%	100%	15s
DDG [65] (Arbiter)	207	199	4%	100%	21s
Assertion Miner [66] (Arbiter)	4175	304	93%	100%	60s
Assertion Miner [66] (LBDR)	1546	258	83%	100%	48s

It is important to highlight that Dominance retains assertions capable of detecting more mutants while pruning the rest. Conversely, Shayan does not have the capability to prune assertions. Essentially, Shayan functions solely as a ranker, and the number of assertions remains unchanged. This explains why only one bar (in gray) is displayed for Dominance results, while different bars are shown for Shayan.

5.3.2 Experimental Results – IMMizer

To assess the effectiveness of the assertion minimization method, IMMizer has been applied to the generated assertions of different assertion miners. These assertion miners correspond to the studies outlined in [42], [65], and [66].

The assertion miners are applied to an open-source project named NoC router Bonfire [147], along with two of its crucial components: Arbiter and LBDR. Arbiter facilitates the connection of input links to output links based on the routing algorithm, while LBDR serves as the control component of the router, responsible for determining the candidate output ports for packet forwarding. Given their role in constructing the control aspect of the router, the verification of these two components is paramount.

Table 10 presents the number of initial assertion sets (*#Initial Assertions*) provided by the assertion miners, the number of minimized assertion sets (*#Minimized Assertions*), and the percentage of reduction in the number of assertions (*Reduction in #Assertions*). Additionally, the percentage of detected mutants by the initial set which is correspondingly detected by the minimized set (*Mutant Detection*), along with the execution time of IMMizer (*Execution Time*) are presented in this table. One notable advantage of IMMizer is its capacity to minimize assertions generated through various approaches. As detailed in section 6.1.2, the general structure of the assertions provided by the assertion miners resembles those in Listings 1, 2, and 3. According to Table 10, IMMizer reduces the assertions of A-Team [42] by 38%. It achieves a significant reduction of 93% for the assertions generated by [66] for Arbiter and 83% for LBDR. The reduction percentage for assertions from [65] is 4%.

It is important to note that the reduction percentage indicates the level of redundancy and inconsistency within the assertions generated by each assertion miner. In essence, IMMizer is capable of identifying and addressing redundant and inconsistent assertions. The 4% reduction observed in the assertions from [65] can be attributed to the uniqueness of assertions generated by this miner. Consequently, the level of redundancy and inconsistency in assertions produced by this assertion miner tends to be lower

Table 11: Reduction in the Allocated Memory and Propositions

Assertion Miners	Initial Assertions	Minimized Assertions	Memory Reduction	Proposition Reduction
A-Team [42] (Arbiter)	45.28 KB	30.32 KB	33.03%	10%
DDG [65] (Arbiter)	31.76 KB	27.36 KB	13.85%	without change
Assertion Miner [66] (Arbiter)	33.24 MB	4.1 MB	87.66%	24%
Assertion Miner [66] (LBDR)	31.36 MB	4 MB	87.24%	73%

compared to other assertion miners.

Furthermore, a complete set of mutants according to the details presented in section 4.3.1 of Chapter 4 has been employed and injected into the benchmarks. Based on the mutant analysis results outlined in Table 10, there is no reduction in the percentage of detected mutants. This is evident in the column '*Mutant Detection*', where mutant detection remains at 100% across all benchmarks. This signifies that all mutants detected by the initial assertion set are also accounted for by the minimized assertion set. This underscores a key advantage of IMMizer, which manages to detect mutants with identical coverage as the initial assertion sets, but with fewer assertions and reduced overhead.

Furthermore, concerning the execution time required for minimizing assertions, as per the data in Table 10, IMMizer accomplishes the minimization process for A-Team's assertions [42] within 15 seconds. Similarly, for DDG's assertions [65], the process takes 21 seconds, while for assertions from [66], it takes 60 seconds for Arbiter and 48 seconds for LBDR.

In addition to IMMizer's capability to reduce the number of generated assertions, in some cases, it can enhance their readability by eliminating extra propositions from the antecedent of assertions. This is exemplified in the illustrative example depicted in Fig. 9, where *B, !B, C*, and *!C* have been removed in the minimized assertion. According to the experiments presented in the '*Proposition Reduction*' column of Table 11, for A-Team [42], in 10% of assertions, the number of propositions has been decreased. This percentage remained constant for DDG [65]. However, in the assertion miner of [66], a reduction of 73% and 24% has occurred for LBDR and Arbiter, respectively.

Moreover, a reduction in the number of assertions and propositions results in a reduction in the memory allocated for these assertions. This outcome is presented in Table 11. The columns labeled '*Initial Assertions*' and '*Minimized Assertions*' display the memory allocated to the initial and minimized assertions, respectively. The column '*Memory Reduction*' indicates the reduction in the allocated memory for the minimized assertion sets. For the assertions associated with A-Team [42], approximately 33% memory reduction has occurred after minimization. Similarly, for DDG [65], this reduction is 13.85%. For the assertion miner in [66] 87.24% memory reduction for LBDR and 87.66% for Arbiter have been reported.

To investigate the efficiency of the proposed method compared to other approaches, a comparison has been conducted between IMMizer and a tool called Shayan [78]. It is important to note that Shayan employs various data mining metrics to rank all assertions, subsequently selecting the top-ranked ones for verification engineers. Therefore, for a fair comparison, an equal number of top-ranked assertions from Shayan, matching the number of minimized assertions by IMMizer, has been considered. For instance, in the case of A-Team - Arbiter as shown in the first row of Table 10, where the number of minimized assertions equals 198, the first 198 top-ranked assertions from Shayan have

Table 12: Efficiency of IMMizer over Shayan

Assertion Miners	**#Assertions**	**Improvement in Mutant Detection(%)**
A-Team [42] (Arbiter)	198	equivalent
DDG [65] (Arbiter)	199	5
Assertion Miner [66] (Arbiter)	258	45
Assertion Miner [66] (LBDR)	304	25

been included in the comparison.

Table 12, presents these results. The number of assertions has been reported in the column labeled '*#Assertions*', while the efficiency of the method over Shayan has been detailed in the column '*Improvement in Mutant Detection (%)*'. For A-Team [42], the top 198 assertions ranked by Shayan have been compared with 198 minimized assertions by IMMizer. After performing mutant analysis, both methods were able to detect the same number of injected mutants. In the case of DDG [65], IMMizer detected 5% more injected mutants compared to Shayan. Moreover, for the assertion miner proposed in [66], IMMizer detected 25% and 45% more mutants for LBDR and Arbiter, respectively, compared to Shayan. These results demonstrate that while IMMizer can minimize the number of initial assertions, it also surpasses other tools like Shayan in detecting mutants.

5.4 Conclusions

In this chapter, two innovative methods, Dominance and IMMizer, were presented to enhance the assertion-based verification domain in terms of automatic assertion evaluation and minimization. Dominance, a data mining-based method, evaluates the quality of assertions and consolidates them into a unified set, while IMMizer significantly minimizes the number of assertions by identifying redundant and inconsistent ones.

Experimental results indicate that the Dominance algorithm can select high-quality assertions capable of detecting nearly the same percentage of mutants as the initial set, with significantly fewer assertions. On the other hand, the experimental results of IMMizer demonstrate its remarkable ability to minimize assertions from various assertion miners without compromising design behavior coverage. Furthermore, compared to other assertion minimization approaches (*e.g.*, data mining-based methods), IMMizer exhibits superior accuracy in mutant detection while utilizing fewer assertions. Moreover, IMMizer enhances assertions readability by eliminating unnecessary propositions, thereby reducing memory overhead on the system.

6 Automatic Generation of Assertions for Security Verification

This chapter aims to utilize assertion-based verification and tap into the potential of assertions within the realm of security verification. To achieve this goal, a security-based assertion miner, based on the paper [152], is proposed in section 6.1. This miner is capable of automatically generating a set of security assertions tailored for RISC-V processors to detect security vulnerabilities such as hardware Trojans. Furthermore, in section 6.2, a method named ADAssure is introduced to enhance the safety and security of autonomous vehicles [153]. ADAssure utilizes assertions for debugging and bug localization of autonomous driving control algorithms in AVs. This methodology has been implemented and tested on a real autonomous vehicle, IseAuto [154].

6.1 Automatic Generation of Assertions for Processors: RISC-V Case Study

In this section, an automated security-based assertion miner for RISC-V processors is proposed [152]. To the best of the author's knowledge, this is the first automated security-based assertion miner that can extract security assertions with the guidelines of ISA characteristics of security properties, directly from the simulation trace of the processor [152]. This extracted security assertion set is employed in the security verification process to uncover vulnerabilities, such as hardware Trojans, embedded within the design. It is noteworthy that the proposed method is expandable to any processor family and is not limited to RISC-V. In the following subsections, the proposed method is elaborated in more detail.

6.1.1 Preliminaries

In this subsection, the definitions used in this chapter are briefly explained. Furthermore, the definitions presented in Chapters 2 and 4 are referred to in Chapter 6.

Definition 19 *A **security property** in this study is defined as a critical security aspect of the processor that neglecting consideration of it can lead to security vulnerabilities in the processor. These security properties are usually presented in the specification of the processor [152, 155, 156].*

Definition 20 *A **security assertion** in this study checks the consistency between the defined behaviors in the security properties (Definition 19) and the actual implementation when it faces an attack [152]. A security assertion is a composition of propositions through temporal operators that must hold or must become true during the execution of the design [42, 152]. Typically, a security assertion is divided into two parts: the left side, named antecedent, and the right side, called consequent [42, 152].*

The general structure of a security assertion in PSL is like $always(antecedent \rightarrow consequent)$, which implies that the consequent will hold whenever the antecedent occurs [109, 152].

6.1.2 Background

In this subsection, the related concepts and background used in this chapter are briefly explained.

RISC-V Instruction Set Architecture

RISC-V provides a flexible instruction set for various general and application-specific scenarios. Around a set of basic mandatory instructions for arithmetic, control flow and memory instructions, called the base instruction set (*e.g.*, RV32I for the 32bit base instruction set) [152, 155, 157], various instruction set extensions can be appended (*e.g.*, multiply/divide, floating point numbers, vector operations).

In this work, the RV32I base instruction set is used, but the proposed method is independent of the specific ISA or utilized instruction set extensions. The RV32I base instruction set defines a 32bit architecture around 32 general-purpose registers `x0` to `x32` (with `x0` being constant 0). More information on the RISC-V instruction set, and its various extensions can be found in Volume 1 [155] and further details on the privileged architecture details, especially CSRS, can be found in Volume 2 [156] of the RISC-V Specification, respectively.

Threat model: MicroRV32 Platform

Amongst the many available open-source implementations, the MicroRV32 platform [158], implemented in the open-source Hardware Description Language SpinalHDL has been chosen. Through the modern SpinalHDL language, it is possible to prototype modifications in the data path quickly, while keeping control over the generated Verilog or VHDL description [152]. The platform is synthesizable for FPGAs and ASICs, while providing a lightweight and robust microarchitecture. MicroRV32 features a configurable multi-cycle processor compliant to RV32IMC, meaning it is capable of the aforementioned RISC-V 32-bit base instruction set (I), the multiply/divide extension (M) and the compressed instruction extension (C) [152]. Within the platform, a set of peripherals enables the interaction with the outside environment, similar to other microcontroller units.

In this work, the threat model environment consists of a processor based on MicroRV32 embedded in a SoC, with various peripherals and a memory hierarchy [152].

Attack Model

To evaluate the proposed method, Hardware Trojans (HTs) have been implemented based on the following details. The attacker targets the RISC-V RTL code, aiming to disrupt normal operations of IPs and cause damages to the IP design house, *e.g.*, financial losses for any reason [152]. Specifically, the attacker intends to add three Hardware Trojans to the processor: two of them alter the control unit's functionality, while one focuses on the memory and illegal access to it [152]. The attacker possesses knowledge of the design modules and implementation. HTs typically consist of a trigger and a payload [152, 159]. The trigger is the condition activating the Trojan, while the payload executes the malicious function [152, 159]. Triggers can be of different types like Always-On, Conditional, or Time-Based, with payloads causing diverse corruptions like Data and Control Flow Manipulation, Denial-of-Service (DoS), etc [152, 159]. The implemented HTs use conditional triggers, activating under specific rare conditions, and their payload manipulates program data and control flow. Trojan 1 triggers a specific input combination in the control flow. Its payload alters the execution flow and causes incorrect computation in specific part of control unit [152]. Trojan 2 activates through a specific sequence of control signals, initiating illegal memory access with a payload involving unauthorized memory access [152]. Trojan 3 is triggered by an improbable combination of input conditions, leading to the alteration of the opcode signal and manipulation of update registers. Its payload executes incorrect instructions, disrupting the normal program flow, potentially compromising system integrity, and

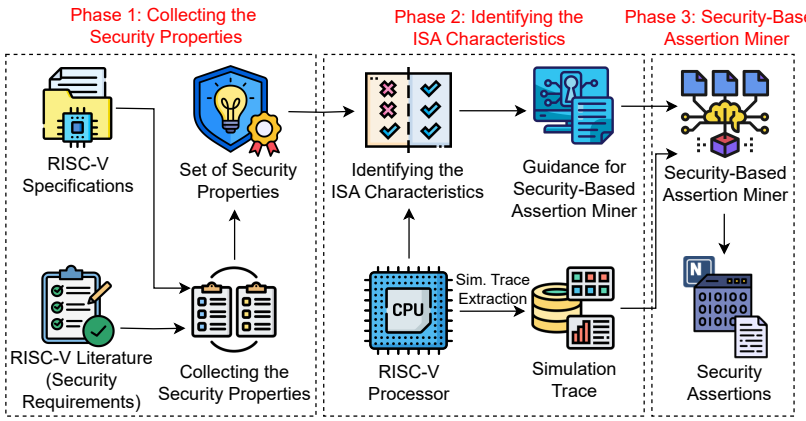


Figure 14: Overview of the proposed method [152]

enabling the attacker to control the instruction sequence [152]. In this study, the HTs have been implemented so that they will be activated in very rare conditions, making their detection difficult.

6.1.3 Methodology

Fig. 14 provides an overview of the proposed method, which is structured in three key phases [152]. These phases are 1- Collecting the Security Properties, 2- Identifying the ISA Characteristics, and 3- Security-Based Assertion Miner [152].

In the first phase, leveraging RISC-V specifications [155, 156] and already introduced security properties from literature [88, 89, 160–162], a set of RISC-V security properties (Definition 19) is collected [152]. This set, along with the RISC-V implementation, becomes the input for the second phase. In the second phase, the security properties of the processor are translated to the instruction set architecture [152]. The main aim of this phase is to systematically identify corresponding instruction sets and the associated signals related to each security property.

Based on this identification, a guideline is formulated for the next phase. This guide assists the security-based assertion miner in the third phase [152]. Operating with the simulation trace generated from the RISC-V design and the guidance report, the security-based assertion miner automatically mines a set of security assertions (Definition 20). These assertions encompass the behaviors outlined in the security properties identified in the initial phase [152]. Further details for each phase are elaborated in the subsequent subsections.

Collecting the Security Properties

After reviewing the RISC-V specifications [155, 156] and its security properties from literature [88, 89, 160–162], a set of security properties (Definition 19) have been collected for the purpose of this work. These collected security properties are outlined in Table 13. However, it is noteworthy that the proposed method can seamlessly extend to encompass other security properties [152]. This versatility enables the method to cater not only to the collected set in the case study benchmark (RISC-V) but also to the diverse security properties of other processors.

In this study, security properties have been collected from the most important categories pertinent to processor security, namely Memory Access, Control flow, and

Table 13: Collected Security Properties [152]

Security Properties	Security Property Type
1: Calculation of memory address and memory data is correct	Memory Access
2: Jumps update the program counter correctly	Control Flow
3: Jumps update the link register correctly	Update Register
4: Addition with register value and immediate value results in correct result in the correct target register	Update Register
5: Load immediate value into the upper 20 bits in the correct target register	Update Register
6: Adds the immediate as upper 20 bits to the program counter and puts it into the correct target register	Update Register

updating Registers [152]. In papers [88, 89, 160–162] which represent the latest studies on processor security properties, these categories have been reported as critical areas requiring scrutiny in the security verification process of processors. Consequently, as depicted in Table 13, security properties corresponding to each of these categories have been delineated [152]. More details for these security properties are as follows [152]:

- Security property 1: The first security property of Table 13 describes the correctness and relation between the current load or store instruction and the resulting memory address and data on the memory interface. For example, if an HT corrupts the address maliciously to redirect the data, the mismatch between the intended address and the actual address should become visible.

- Security property 2: The second security property ensures, that the jump instructions redirect the control flow correctly, *i.e.*, the change in program counter reflects the provided register and immediate values accordingly. In this case, an HT could redirect the control flow to malicious code before returning to the original application's code.

- Security property 3: As jump instructions are used to call functions and return to the code calling a function, the storage of this return address can similarly be tampered with. Thus, the third security property ensures, that the return address saved on jump instructions is correct.

- Security property 4: As control flow and memory access instructions prepare addresses and values through arithmetic and logic operations, their correctness is vital for security. To avoid information leakage through HT, the fourth security property assures that the `ADDI` (add immediate) instruction saves the correct result only into the correct target register. As an example, an HT could enable writing to the target register and a temporary register, in order to leak the information in a different subroutine.

- Security property 5: Similarly, the fifth security property ensures the correctness for the `LUI` (load upper immediate) instruction, that the accordingly extended immediate value is stored into the correct target register.

- Security property 6: Lastly, the sixth security property addresses the `AUIPC` (add upper immediate and program counter) instruction. The security property ensures the correct arithmetic and storage in the correct target register. As this instruction is utilized to prepare target addresses, an HT can potentially redirect the control flow to a different part of the code.

These security properties serve as the inputs for the next phase, *i.e.*, Identifying the ISA Characteristics.

Identifying the ISA Characteristics

In order to identify which parts of the ISA specification contribute to a security property, an example is utilized that covers security properties 2 and 3. Consider the RISC-V instruction `JAL rd, offset`, *Jump And Link*. This instruction manipulates the control flow of the application, jumping to a new location in the program by setting the *Program Counter* (PC) to an `offset` encoded as an immediate value (*i.e.*, `PC = offset`) [152]. As a second part, the instruction will store the value of program counter of the instruction after the `JAL` instruction (`rd = PC + 4`). We can consider possible violation of security if, for example, a Hardware Trojan introduces are change of the offset under specific system conditions. This could lead to a possible attack in which an unwanted execution of different code occurs. Therefore, for the `JAL` instruction, we can denote two possible high-level properties [152]:

- Jumps update the PC correctly according to the `offset` passed in the instruction immediate.
- Jumps update the register `rd` with the correct PC (*i.e.*, `rd = PC+4`).

Consequently, the security-based assertion miner would need to find potential security assertions covering the signals mentioned in the high-level properties. In a more general sense, these steps can be abstracted as follows [152]:

1. Identifying instructions affected by threat model (*e.g.*, control flow integrity affects the instructions for branch, jump, and address manipulation).
2. Identifying each part of functional behavior contributing to instructions (*e.g.*, jump instruction changes PC and a register based on PC value).
3. Formulating high-level property reflecting parts of the functional behavior (*e.g.*, resulting addresses for store or load operations have to be consistent with the initial instruction and register values).
4. Utilizing the high-level property to identify the affected signals in the processor and its interfaces. This includes the relation between instruction bits and the observable behavior on results and interfaces.

In this phase, alongside the guidance report created based on the identified ISA characteristics, a simulation trace is generated from the RISC-V design. These inputs are utilized by the security-based assertion miner in the next phase.

Security-Based Assertion Miner

In this phase, the details of the proposed security-based assertion miner are elaborated. As illustrated in Fig. 15, the proposed miner is comprised of four primary steps: 1) Signal Identification and preprocessing of Simulation Trace, 2) Association Rule Mining, 3) Time Notation, and 4) Assertion Conversion [152].

During the Signal Identification and preprocessing step, after identifying the RISC-V signals that are associated with the identified security properties, the exhaustive simulation trace of the RISC-V undergoes preprocessing to prepare the data. Subsequently, in the Association Rule Mining step, the proposed algorithm is applied to the preprocessed simulation trace to mine all association rules (Definition 9) derived from the simulation trace [152].

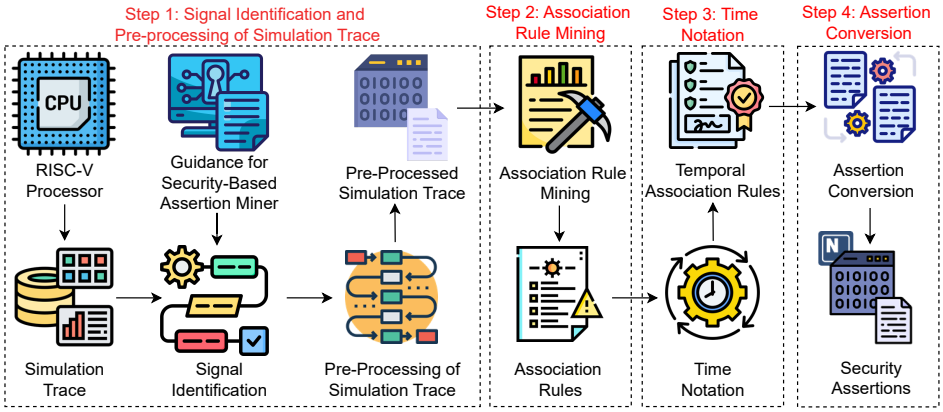


Figure 15: Proposed Security-Based Assertion Miner [152]

Afterward, the association rules obtained from the second step are passed to the third step, *i.e.,* Time Notation. Here, the extracted association rules are integrated with the concept of time to generate appropriate time-integrated rules (temporal association rules) in the form of $next[\mathcal{N}]$ pattern (Definition 5) [152]. These rules serve as the input for the Assertion Conversion step. Consequently, the Assertion Conversion step transforms the rules from the previous step into assertions, rendering them ready for utilization in the security verification process of the RISC-V design [152].

Algorithm 4 presents the detailed process of the first three steps of the security-based assertion miner [152]. In this algorithm, $\mathcal{ST}$ denotes the simulation trace and $\mathcal{ST}'$ is the preprocessed simulation trace, while f and t represent the simulation trace's outputs and input values.

In the following subsections, each phase of the security-based assertion miner is discussed in more detail.

Signal Identification and preprocessing of Simulation Trace

Lines 6 to 8 of the Algorithm 4 are related to the Signal Identification and preprocessing step of the security-based assertion miner.

In this phase of the security-based assertion miner, at first, the RISC-V simulation trace and guidance report generated from the second phase of the method (Identifying the ISA Characteristics) are processed for signal identification [152]. In signal identification, the assertion miner prunes all the signals of the simulation trace that are not relevant to the identified security properties. Once signals associated with the specified security properties are identified, the simulation trace undergoes preprocessing [152].

To preprocess the simulation trace, all the identified output of the simulation trace is moved $\mathcal{N}$ records above its original position (line 8 of the Algorithm 4). However, the identified inputs of the simulation trace remain as they are [152]. Traditional association rule mining algorithms (*e.g.*, apriori [132]) cannot typically mine the rules in the form of *next[$\mathcal{N}$]*. Because of this reason, and also since the corresponding output of input variables in a sequential hardware design may occur in the simulation trace $\mathcal{N}$ time instants later, this preprocessing needs to be performed. This ensures the correct alignment of outputs with their corresponding inputs, allowing for accurate temporal analysis and also mining patterns for different $\mathcal{N}$ time instants (clock cycles) [152].

Algorithm 4 Security-Based Assertion Miner [152]

1: **Inputs:** $\mathcal{N}$, $\mathcal{ST}$, min_supp;
2: **Output:** $next[\mathcal{N}] = antecedent \rightarrow next[\mathcal{N}]consequent$;
3: $\mathcal{R} = antecedent \rightarrow consequent$;
4: $L_1 = \{frequent\ \ 1-itemsets \in \mathcal{ST}'\}$;
5: $K = 2$;
6: $\mathcal{ST}' \leftarrow$ **prune_nonrelevant_security_signals**$(\mathcal{ST})$
7: **forall** $f \in \mathcal{ST}'$ **do**
8: $\quad \mathcal{ST}' = \mathbf{MoveUp}(f(\mathcal{N}))$;
9: **while** L_{k-1} $!= \varnothing$ **do**
10: $\quad C_k =$ **generate_candidate_itemsets**(L_{k-1});
11: $\quad L_k =$ **prune_infrequent_itemsets**(C_k, min_supp);
12: $\quad k = k+1$;
13: **foreach** $frequent\ \ itemset\ L_i \in L$ **do**
14: $\quad$ **foreach** $subset\ \ S\ \ of\ \ L_i$ **do**
15: $\quad\quad$ **if** $(S\ != \varnothing)$ && $(S\ !=\ \ L_i)$ **then**
16: $\quad\quad\quad confidence = support(L_i)\ \ /\ \ support(S)$;
17: $\quad\quad\quad$ **if** $confidence >= min\_conf$ **then**
18: $\quad\quad\quad\quad \mathcal{R} \leftarrow$ **association_rule**$(S => L_i)$;
19: return $\mathcal{R}$;
20: **if** $(\mathcal{R}.antecedent == (t \in \mathcal{ST}'))$ and $(\mathcal{R}.consequent == (f \in \mathcal{ST}'))$ **then**
21: $\quad next[\mathcal{N}] \leftarrow$ **label**$(\mathcal{R})$;
22: **else**
23: $\quad$ **Discard**$(\mathcal{R})$;

Association Rule Mining

After preprocessing the simulation trace in the previous step, the resulting preprocessed simulation trace is subsequently fed into lines 9 to 19 of Algorithm 4 to mine association rules [152]. Applying these lines of the algorithm to the preprocessed simulation trace provides us with a set of association rules in the form of $antecedent \rightarrow consequent$.

In lines 9 to 12 of Algorithm 4, frequent itemsets (Definition 8) of various sizes (1-itemsets, 2-itemsets, etc.) are generated iteratively until the list of the frequent itemsets is empty. Specifically, the algorithm mines frequent itemsets whose support values (Definition 10) exceed the min_supp value (Definition 11), while pruning the others [152]. In line 10 of the algorithm, C_k is the candidate itemsets of size *k* that are generated by combining frequent (*k*-1)-itemsets and L_k in line 11 of the algorithm is the set of frequent *k*-itemsets. In this algorithm, 1-itemsets consist of individual variables of simulation trace, 2-itemsets are pairs of variables, etc.

After mining the frequent itemsets and adding them to the L_k list, in lines 13 to 19 of the algorithm, the association rules are extracted from the list of frequent itemsets [152]. To clarify these lines of the algorithm, let's consider an example where the list of frequent itemsets is equal to 4-itemsets of {A, B, C, D}. To generate association rules from this frequent itemset, we consider all non-empty subsets of it. These subsets are 1-itemsets of {A}, {B}, {C}, {D}, 2-itemsets of {A, B}, {A, C}, {A, D}, {B, C}, {B, D}, {C, D}, as well as the 3-itemsets of {A, B, C}, {A, B, D}, {A, C, D}, {B, C, D}, and 4-itemset of {A, B, C, D}. Afterward, for each non-empty subset $\mathcal{Y}$, an association rule of the form $\mathcal{Y} => \mathcal{X} - \mathcal{Y}$ is generated. By doing so, the algorithm considers all possible combinations of items in the frequent itemset to identify significant associations between different sets of items [152]. For example, if $\mathcal{X}$

$= \{A, B, C, D\}$ and $\mathcal{Y} = \{A, B, C\}$, then the rule $\{A, B, C\} => \{D\}$ is generated. The antecedent of the rule ($\mathcal{Y}$) is the subset $\{A, B, C\}$, and the consequent of the rule ($\mathcal{X} - \mathcal{Y}$) is the set difference between the frequent itemset $\{A, B, C, D\}$ and the antecedent $\{A, B, C\}$, which is $\{D\}$. This process is repeated for each non-empty subset of the frequent itemsets list, yielding a set of association rules. Finally, we evaluate each association rule for its confidence, pruning those that are below the min_conf threshold (Definition 13) [152].

Increasing the min_supp value results in fewer assertions that describe more general design behavior, while decreasing the min_supp value leads to assertions covering rare design behavior (corner cases) [152]. These corner cases are important as attackers can consider them for performing any corruption in the design. Similarly, raising the min_conf value produces fewer but more valid assertions [152]. Valid assertions refer to assertions that will not be violated during the simulation with different attack scenarios. The utilization of these values in the security-based assertion miner facilitates an effective vulnerability detection process. In this study, min_supp and min_conf are set to 0.01 and 1, respectively, as we aim to discover corner cases while achieving high vulnerability detection (details are presented in subsection 6.4.1) [152].

At this point, with the completion of the association rule mining, these rules serve as the fundamental components of the Time Notation step.

Time Notation

In the previous step, the method provides us a set of rules in the general form of $antecedent \rightarrow consequent$. In this step, the method integrates the concept of time into the association rules generated in the association rule mining step, leading to a set of temporal association rules in the form of $antecedent \rightarrow next[\mathcal{N}]consequent$ [152]. Lines 20 to 23 in Algorithm 4 describe the details of the Time Notation step. If the antecedent value matches an input in the simulation trace, and the consequent value has already been moved to another record in the simulation trace, the rule is labeled as a *next* temporal association rule. Otherwise, other mined rules are discarded [152].

Assertion Conversion

In this step, the mined temporal association rules are transformed into temporal security assertions (Definition 20) in SVA language [136] using the labels assigned in the Time Notation step [152]. While the security-based assertion miner provides assertions in the SVA language, this section elucidates the general structure and format of temporal mined rules in PSL [109] for enhanced comprehension. Consequently, the output of the Time Notation step for temporal association rules labeled as *next[$\mathcal{N}$]* is transformed into the PSL format of "*always(antecedent $\rightarrow$ next[$\mathcal{N}$]consequent)*" [152].

6.2 Automatic Generation of Assertions for Autonomous Vehicles: IseAuto Case Study

The development of ADAssure is motivated by three main objectives. Firstly, it seeks to furnish Autonomous Driving (AD) system designers with a methodology to detect and rectify vulnerabilities that correspond to the design of AD algorithms. Secondly, in light of the dynamic nature of autonomous vehicle systems, it strives to establish a structured methodology conducive to consistent, flexible, and repeatable testing practices. Lastly, it aims to facilitate unit testing, enabling the testing of individual components of the autonomous system independently of other dynamic factors influencing autonomous control.

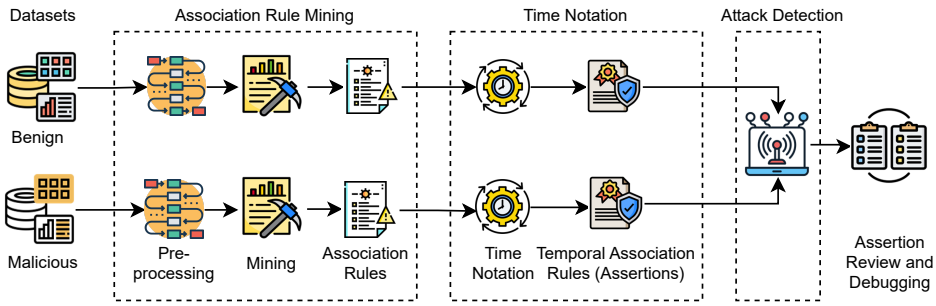


Figure 16: Different Phases of Assertion Generation

The underpinnings of the ADAssure methodology are rooted in the analysis of vehicle dynamics and sensing data, which guides the creation of assertions regarding the vulnerability of AD control algorithms. This analysis involves conducting a sensitivity analysis of vehicle dynamics data (*e.g.*, velocity, yaw, and steering angle), sensor data (*e.g.*, lateral and longitudinal movement), and visualization of the trajectory of the AD system. This process aids in identifying crucial parameters for creating assertions concerning the AD control algorithms. Designers of AD control systems can utilize these assertions to identify and locate vulnerabilities within the control model and develop mechanisms to test and rectify errors. The ADAssure methodology comprises three principal phases: Autonomous Driving Data Collection, Association Rule Generation, and Assertion Review and Debugging. In the following, each of these phases is elaborated in further detail.

6.2.1 Autonomous Driving Data Collection

This phase involves generating data from either real-world system or simulation environment. The advantage of utilizing a simulation environment lies in its ability to automate driving scenarios or design them to test specific conditions, such as cyber-attacks or corner cases. The output data is structured according to predefined metrics, which may include vehicle dynamics parameters (yaw angle, velocity, etc.), sensing data (position co-variance, point-cloud, etc.), and safety parameters (distance-to-collision, etc.). The AD data is formatted in a way that can be interpreted by analytical tools, with the chosen format of .csv in this study.

6.2.2 Association Rule Generation Phase

The objective of this phase is to analyze the data generated in the preceding phase and generate a set of association rules, which will be translated into assertions during the Assertion Review and Debugging phase. This phase comprises three main steps, as illustrated in Fig. 16: a) Association Rule Mining, b) Time Notation, and c) Attack Detection.

The association rule mining is applied to both benign and malicious datasets, resulting in two distinct sets of association rules. Subsequently, these sets of rules are processed through the Time Notation step to integrate temporal information, yielding temporal association rules (assertions) in the form of $next[\mathcal{N}]$ and $before[\mathcal{N}]$ patterns. The $next[\mathcal{N}]$ type of rule is defined in the general form of $\mathcal{X} \rightarrow next[\mathcal{N}]\mathcal{Y}$. This rule signifies that when $\mathcal{X}$ occurs, after $\mathcal{N}$ time instants, $\mathcal{Y}$ will occur. Here, $\mathcal{N}$ represents a positive integer value. Furthermore, the $before[\mathcal{N}]$ rule is defined in the general

Algorithm 5 Association rule mining & time notation

1: **Inputs:** $\mathcal{N}$, $\mathcal{D}$;
2: **Outputs:** $next[\mathcal{N}] = antecedent \rightarrow next[\mathcal{N}]consequent$;
3: $before[\mathcal{N}] = antecedent \rightarrow before[\mathcal{N}]consequent$;
4: $\mathcal{R} = antecedent \rightarrow consequent$; ▷ Initialization and Preprocessing
5: **forall** $f \in \mathcal{D}$ **do**
6: $\mathcal{D}' = \mathbf{MoveUp}(f(\mathcal{N}))$;
7: $\mathcal{R} \leftarrow \mathbf{apriori}(\mathcal{D}')$; ▷ Mining
8: **if** $(\mathcal{R}.antecedent == (t \in \mathcal{D}'))$ and $(\mathcal{R}.consequent == (f \in \mathcal{D}'))$ **then** ▷ Time Notation
9: $next[\mathcal{N}] \leftarrow$ **label**$(\mathcal{R})$;
10: **if** $(\mathcal{R}.antecedent == (f \in \mathcal{D}'))$ and $(\mathcal{R}.consequent == (t \in \mathcal{D}'))$ **then**
11: $before[\mathcal{N}] \leftarrow$ **label**$(\mathcal{R})$;

form of $\mathcal{X} \rightarrow before[\mathcal{N}]\mathcal{Y}$. This rule indicates that whenever $\mathcal{X}$ occurs, $\mathcal{Y}$ should have occurred $\mathcal{N}$ time instants before that. In the 'Attack Detection' step these temporal association rules are compared and ultimately detect attacks and anomalies within the datasets. The following sections present a more in-depth discussion of each step.

Association Rule Mining

This step primarily serves two purposes: preprocessing the datasets and consequently mining association rules from the preprocessed data. In order to mine the association rules, the apriori algorithm [132] was adapted and enhanced to mine temporal rules capable of detecting attacks at various time instances during AV operation. Algorithm 5 demonstrates the details of the Association Rule Mining and Time Notation steps. In this algorithm, $\mathcal{D}$ represents the dataset and $\mathcal{D}'$ is the preprocessed dataset, while f and t denote the dataset's features and target values. To prepare the dataset for mining the $next[\mathcal{N}]$ and $before[\mathcal{N}]$ temporal patterns, all the features of the dataset are moved $\mathcal{N}$ records above their original position (Line 6). However, the target of the dataset remains unchanged. Subsequently, the apriori algorithm is applied to the preprocessed dataset to mine a set of association rules. The output of this phase comprises association rules in the general form of $antecedent \rightarrow consequent$. These association rules are then forwarded to the Time Notation step.

Time Notation

Here, the method incorporates the concept of time into the association rules generated in the previous step, producing a set of temporal association rules. The method identifies the temporal pattern ($next[\mathcal{N}]$ or $before[\mathcal{N}]$) associated with each extracted rule and accordingly assigns the appropriate time label. If the value of the antecedent matches a target value in the dataset, and the consequent value has already been moved to another record in the dataset, the rule is labeled as a $next$ temporal association rule (Line 9). Conversely, if the antecedent of a rule mined in the association rule mining step aligns with a dataset feature that has been moved to another record, and the consequent of the rule aligns with the target value of the dataset, this rule is labeled as a $before$ temporal association rule (Line 11). The mined rules are in the forms of $antecedent \rightarrow next[\mathcal{N}]consequent$, and $antecedent \rightarrow before[\mathcal{N}]consequent$, serving as assertions for debugging the autonomous driving system.

Attack Detection

This step is dedicated to the identification of rules indicating potential attacks on the AV. The assumption is that the rule sets extracted from benign and malicious datasets

should exhibit similarity under normal conditions, devoid of any AV attacks. Any disparity observed between these rule sets indicates an anomaly within the autonomous vehicle. In accordance with this assumption, the temporal association rules (assertions) generated during the time notation phase are categorized into two distinct sets.

The first category encompasses rules solely mined from the malicious dataset, without corresponding counterparts in the benign dataset. Any rule exclusively extracted from the malicious dataset, lacking a counterpart in the benign dataset, indicates an attack. These rules reveal abnormal behavior in the malicious dataset, contrasting with different behavior observed in the corresponding time instance of the benign dataset. Subsequently, these rules are classified as attacks.

The second category encompasses similar rules mined from both benign and malicious datasets, but with different values for *minimum support* (`min_supp`) and *minimum confidence* (`min_conf`). Discrepancies in these values indicate that, while the mined rules are similar, anomalies and abnormal behaviors exist between the datasets. The apriori algorithm utilizes these two metrics (*i.e.*, `min_supp` and `min_conf`). The `min_supp` value serves as the threshold and minimum value, determined by the expert to ascertain whether a rule occurs frequently in the dataset or not [77, 163]. The `min_conf` represents the minimum value that is selected by the expert and is an indication of how often a rule has been found to be true [132, 133]. Increasing the `min_supp` value results in fewer association rules describing more general behavior of the autonomous vehicle, while decreasing it leads to rules covering rare behaviors (corner cases). Similarly, raising the `min_conf` value generates fewer but more valid rules. Valid rules refer to association rules that remain consistent across various attack scenarios, including corner cases. These values incorporated into ADAssure facilitate an efficient attack detection process. The second category of rules aids the ADAssure in effectively identifying corner cases and the attacks that rarely occur on the AV. These rare attacks manifest behavior closely resembling normal vehicle operation but are malicious and can potentially cause AV failure.

6.2.3 Assertion Review and Debugging

During this phase, the association rules generated from the preceding phase are reviewed alongside an analysis of control behavior and individual data parameters to formulate assertions. Trajectory maps of the AD system and graphs illustrating the sensitivity of data parameters in benign and cyber-attack scenarios are compared with the anomalous behavioral patterns identified by the association rule generation phase. Leveraging expertise from algorithm designers and safety validation engineers aids in identifying parameters capable of uniquely demonstrating algorithm vulnerabilities within the system. Once assertions regarding system vulnerabilities are established, debugging efforts concentrate on control flow analysis. As the assertion aids in pinpointing the specific module, the static analysis can focus on the control flow of the substituent functions within the module. For instance, a local-planning module may encompass 15 diverse algorithms, each containing multiple methods or functions. As AD algorithm code comprises differential equations, debugging can suggest optimizations to implement mitigation mechanisms against identified vulnerabilities.

6.3 Autonomous Driving Control Algorithm

To evaluate the methodology, the focus is on an AD control algorithm employed in a real-world AD ride-hailing service. The AD pipeline comprises four pivotal modules: localization, perception, planning, and control. In this study, the localization and

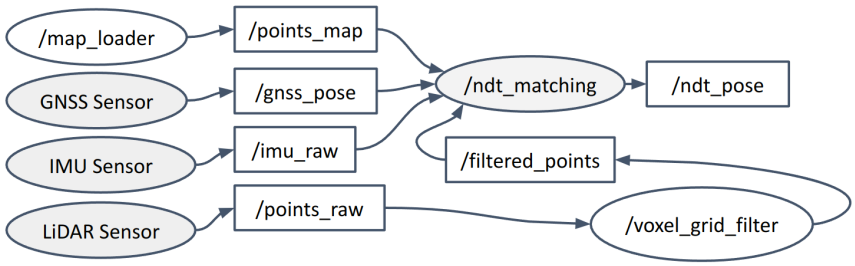


Figure 17: Localization Algorithm Flow within AD System

planning modules are considered.

6.3.1 Localization Module

This module provides accurate information concerning the vehicle's position and orientation. Employing a Normal Distributions Transform (NDT) matching search algorithm, it identifies the best matching position based on sensor perception. It utilizes input from the Inertial Measurement Unit (IMU) and the point cloud generated by the LiDAR. Subsequently, it attempts to match the points from the current scan to a grid of probability functions extracted from the map. NDT matching algorithms can further leverage the GNSS sensor, offering initial approximate localization estimates on geo-referenced maps to avoid sudden errors in localization calculations that could result in failures. Fig. 17 illustrates the flow of the localization algorithm within the AD system.

6.3.2 Planning Module

To initiate mission planning within the AD system, the global planner initially generates a global reference path utilizing a vector (road network) map. The function of the global planner is to delineate a route between the mission's starting and goal positions on the road map. Subsequently, the local-planner generates smooth and obstacle-free trajectories within the operational local domain, adhering to the global route. The local-planner encompasses various modules, as depicted in Fig. 18: trajectory generation, trajectory evaluation, intention and trajectory estimator, object-tracker, and behavior selection (decision making) [164]. The trajectory generation module produces alternative tracks, referred to as roll-outs, parallel to the main path defined by the global planner. Subsequently, the trajectory evaluation module assesses all potential roll-outs and inputs data from the AV's sensed-data to estimate costs. The behavior selector then directs the AV to motion on a roll-out based on the least-cost trajectory.

6.4 Experimental Results – RISC-V and ADAssure

In this section, the experimental results concerning the security-based assertion mining for the RISC-V processors and debugging and bug localization of AD control algorithms of AVs (*i.e.*, ADAssure) are presented. Subsection 6.4.1 outlines the findings regarding the security-based assertion miner, while subsection 6.4.2 delves into the results for ADAssure.

6.4.1 Experimental Results – RISC-V

For the experimental evaluation, the security-based assertion miner is utilized to generate security assertions in the form of SVA. As shown earlier, in Fig. 15, the security-based

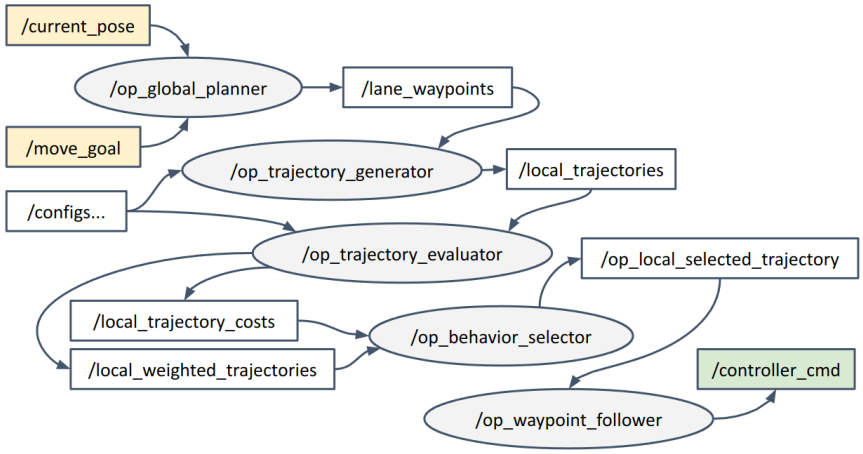


Figure 18: Abstract Local Planning Algorithm Flow within AD System

assertion miner requires a simulation trace generated from a processor. In this case study, the processor embedded in the open-source MicroRV32 platform [158] is utilized. For generating the simulation trace, a software application on the processor that is centered around checksum calculations for embedded systems has been executed. It features different types of control flow, loops, arithmetic operations, and interaction with the memory as well as the available peripherals [152]. Hence, a simulation trace with 10000 records has been generated that activates various parts of the microarchitecture to provide a diverse data set for security assertion mining. The generated assertions are then evaluated by including Hardware Trojans in the processor's microarchitecture together with the generated security properties, to verify them. Notably, in the experiments the values for the minimum support (Definition 11) and minimum confidence (Definition 13) have been set to 0.01 and 1, respectively [152]. These values allow us to guide the security-based assertion miner in mining fewer yet more valid assertions, effectively considering the corner cases of designs. Moreover, $\mathcal{N}$ has been set to 2 for the $next[\mathcal{N}]$ pattern, but it can be adjusted to other values [152].

Table 14 exhibits the detailed experimental results about the assertions that are associated with any of the security properties that have been presented in the Collecting the Security Properties section [152]. The proposed security-based assertion miner generated a total of 4036 security assertions. It should be noted that while the total number of generated assertions is 4036, some of them overlap so that they contain the signals of the design that are related to several security properties [152]. According to the experimental results in Table 14, 870 and 1490 security assertions are related to security properties 1 and 2, which means that these numbers of assertions can cover the behaviors that have been described for these two security properties [152]. The results indicate that for security properties 3 and 4, 1522 and 2026 security assertions have been mined, respectively. This figure for both the security properties 5 and 6 is 1898 assertions [152].

Table 15 presents the experimental results on the Trojan detection [152]. The column '#Security Assertions Detecting Trojans' presents the number of security assertions that could detect any of the Trojans. For Trojan 1, 16 assertions detected it and 64 and 176 security assertions detected Trojans 2 and 3, respectively [152].

Table 16 presents a comparative analysis between the proposed security-based

Table 14: Detailed Experimental Results on Six Different Security Properties [152]

Total Number of Assertions	Number of Assertions Covering the Security Properties					
	Security Property 1	Security Property 2	Security Property 3	Security Property 4	Security Property 5	Security Property 6
4036	870	1490	1522	2026	1898	1898

Table 15: Experimental Results on Detected Trojans [152]

Trojans	#Security Assertions Detecting Trojans	Trojan Detection
Trojan 1	16	✓
Trojan 2	64	✓
Trojan 3	176	✓

✓: Trojan has been detected.

assertion miner and HARM, the latest assertion miner in the literature [72, 152]. HARM is explicitly presented as a tool that can be employed in the context of security verification for Trojan detection, making it a relevant tool for the comparison. While HARM generated an extensive set of 16073 assertions, contributing to prolonged security verification process, the proposed method yielded a more compact and accurate set of 4036 assertions [152]. Among all the mined assertions by HARM, a total of 397 assertions could detect the Trojans, while the proposed method is this chapter accomplished Trojan detection with 256 assertions [152]. The ratio of security assertions that detect Trojans to the total number of assertions demonstrates that our mined security assertions are more effective in Trojan detection. These assertions exhibited a superior effectiveness with a ratio of 6.3%, in contrast to HARM's 2.4% [152]. Considering the fact that there are only three Trojans, and as mentioned in the Attack Model section, the probability of their activation is very rare, 6.3% shows promise. These findings underscore the efficiency of the proposed method in producing a smaller yet more potent and accurate set of security assertions. Furthermore, the security-based assertion miner demonstrated significantly shorter execution time, completing the assertion mining process in approximately 5 minutes, compared to HARM's duration of over an hour [152].

6.4.2 Experimental Results – ADAssure

To evaluate the influence of corner cases on AD system behavior using the ADAssure methodology, the datasets comprising corner case scenarios obtained from both simulation and real-world driving scenarios of the target AD system have been utilized. The first corner case scenario dataset encompasses three distinct cyber-security attacks on the AD system conducted in a simulation environment. Given our focus on the planning and localization algorithms, a low-fidelity simulation environment provided by `Autoware.AI`, coupled with the `OpenPlanner 2.5` planning algorithm has been employed. The second corner case scenario dataset involves a Global Positioning System (GPS) spoofing event that occurred on the AD system during its operation in the urban road network of a capital city.

Table 16: Comparison of the proposed method with HARM [152]

Assertion Miner	Total number of Security Assertions	#Security Assertions Detecting Trojans	Ratio of Security Assertions Detecting Trojans to the Total Number of Assertions (%)	Execution Time
Proposed Method	4036	256	6.3	5min30sec
HARM [72]	16073	397	2.4	74min31sec

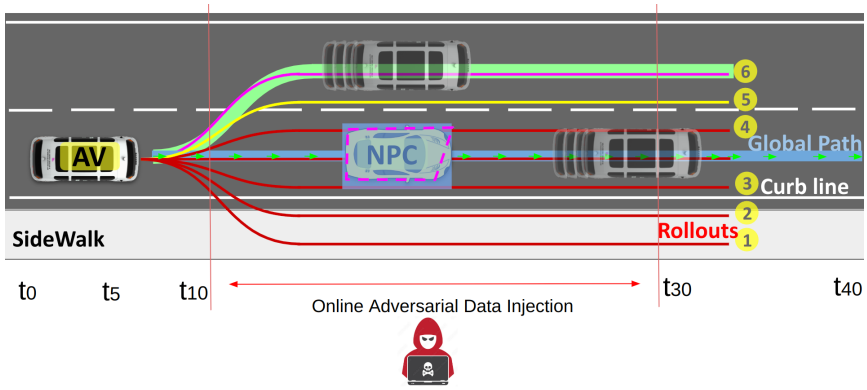


Figure 19: The threat model used for conducting the attack cases

AD Control System Datasets

In the following, details of the datasets utilized in this study are presented:

Cyber-security Corner Case Dataset

Within this dataset, three attacks were carried out on the target AD vehicular system while attempting an overtaking maneuver. These attacks are categorized as follows: 1) Lateral Position Offset Attack, 2) Longitudinal Position Offset Attack, and 3) Message Time-Delay. In the lateral and longitudinal position offset attack, an attacker injects malicious data input into the lateral or longitudinal position while the AD vehicular system is executing the overtaking maneuver (Fig. 19). This attack could be executed through GPS spoofing or interception and manipulation of the localization sensor data. For the message time-delay attack, the attacker introduces a delay into the `current_pose` (lateral and longitudinal) sensor messages reaching the AD control pipeline. The malicious data is injected between the 21m and 67m marks of the AV journey (traveled distanced). Each attack has been executed 300 times, encompassing a range of different attack parameters. The lateral and longitudinal attacks introduced deviations ranging from 0.16% to 1.0%, equivalent to approximately 20cm to 1m. The message time-delay introduced delays of 0.3%, 0.6%, and 1.0% second, as a message is transmitted every 20ms, this range demonstrates a delay of 15 to 50 messages. Overall, the dataset encompasses over 1500 scenario runs of both attacks and benign safety cases.

GPS Spoofing Real-World AV Dataset

The sensor data from the AD ride-hailing service is transmitted to an edge server through a logging node, where it stores the AD System data in a database. While operating in the vicinity of the city's port area, the AD vehicle experienced a loss of localization due to a GPS spoofing incident, which also impacted other GPS-enabled platforms.

This GPS spoofing occurred intermittently over the preceding months. The dataset utilized in this study originates from the logging system of the AD ride-hailing service.

AD System Data

The simulation and real-world datasets have been structured to output data as shown in Table 17.

Table 17: AD System Data

AD Data Type	Description
AV_X	Longitudinal Position of the AD System as to the HD Map
AV_Y	Lateral Position of the AD System as to the HD Map
AV_Steer	Steering Angle of the AD System
AV_Vel	Velocity of the AD System
AV_Yaw	Orientation of the AD System based on its centre of gravity
Roll-out_Num	Current Lane according to the lane selector of the AD Control Algorithm
DTC	Distance to collision of the AD vehicular system to the overtaking vehicle
Position Co-variance	GPS position co-variance
Altitude	Altitude derived from the GPS

Table 18: ADAssure Assertion Generation phase results

Dataset		Assertion			Execution Time
Name	#Records	Total	$\#Next[\mathcal{N}]$	$\#Before[\mathcal{N}]$	
Longitude	412	5	3	2	1ns
Latitude	356	7	7	0	1ns
Delay	417	5	3	2	1ns
GNSS	16	5	4	1	1ns

Results and Analysis

To evaluate the effectiveness of the ADAssure methodology, six attack types and their corresponding safety (benign) scenarios have been chosen. These attack types encompass each of the aforementioned attacks with varying levels of noise (lateral and longitudinal position offsets, delay message).

Automated Analysis

Employing the ADAssure methodology on the three attack types resulted in three distinct sets of assertions, corresponding to each attack type. The results of the assertion generation phase are detailed in Table 18. The minimum support threshold (`min_supp`) was set at 0.01, while the minimum confidence threshold (`min_conf`) was 1. Furthermore, $\mathcal{N}$ has been set to 2 for the $next[\mathcal{N}]$ and $before[\mathcal{N}]$ patterns, but it can be adjusted to other values. Notably, the method exhibits swift execution times. Within the analysis of the three attacks of the cybersecurity corner case dataset, the assertions identified two patterns of anomalous AD behavior. Firstly, extreme steering angles of 20° and -20°, accompanied by sudden lane transitions. Secondly, multiple lane-transitions coupled with extreme steering angle and sudden changes in vehicular velocity. This behavior indicates the impact of cyber activity on the smoothness of the initiation of the overtaking maneuver, resulting in erratic movements and, in some cases, a collision event. The assertions generated from the GNSS spoofing dataset identified alterations in altitude and position co-variance, which were consistent with significant changes in GPS coordinate values and resultant changes in altitude.

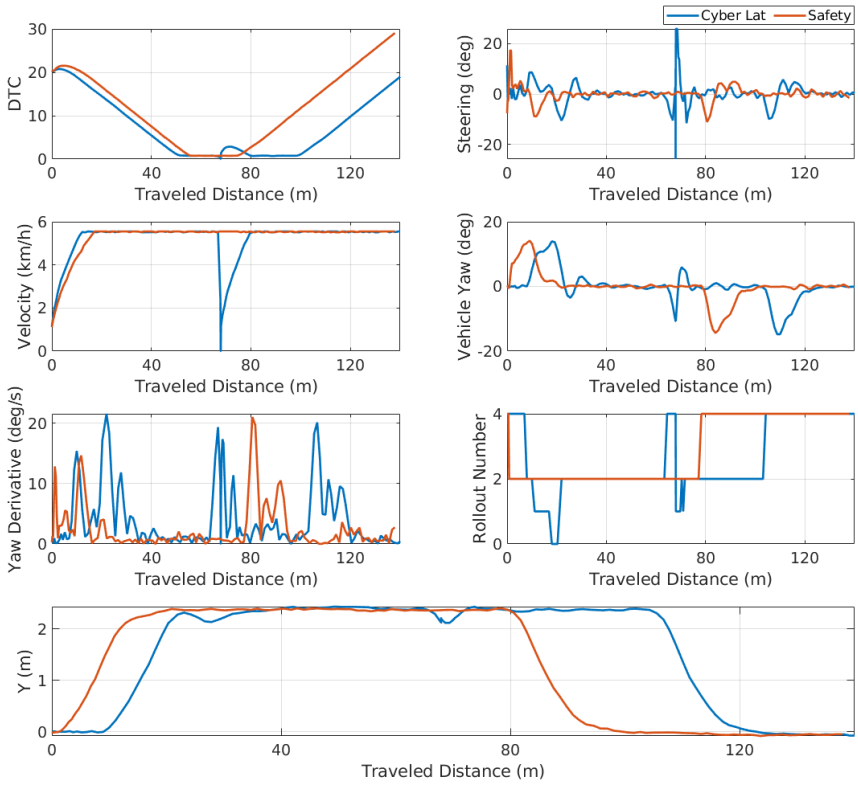


Figure 20: Lateral position offset attack vehicle parameters

Assertion Review and Debugging

The patterns identified in the association rules allow us to infer that the Yaw angle and angular velocity serve as good indicators to illustrate the impact of cyber-attacks. During the injection of position offset attacks, the vehicle's orientation exhibits dramatic action; in some circumstances, the vehicle sometimes appears to spin. As depicted in Fig. 20, the Yaw (angle) of the vehicle during the Lateral Position Offset Attack shows sharp fluctuations, reaching 15deg/sec from the 15 meters mark of the AV journey. This dynamic behavior of the vehicle is a characteristic also observed in both the longitudinal position offset (Fig. 21) and delay message attack (Fig. 22). The results for the velocity parameter indicate that it only reflects immediate collision of the vehicle, and it does not support early detection of anomalous vehicle behavior. Assertion 1 contends that the AD system should prevent movements that exceed the physical limitations of the steering model.

Assertion 1: To determine the vulnerability of the yaw angle and momentum, we can derive the assertion: *AV.displacement_of_yaw_angle* > *max_yaw_angle_threshold && time* < *time_threshold*.

The roll-out transition, steer, and distance-to-collision parameters exhibit discernible changes during a cyber-attack. Manipulating the lateral and longitudinal position alters the vehicle's position on the map, thereby inducing greater transitions between roll-outs,

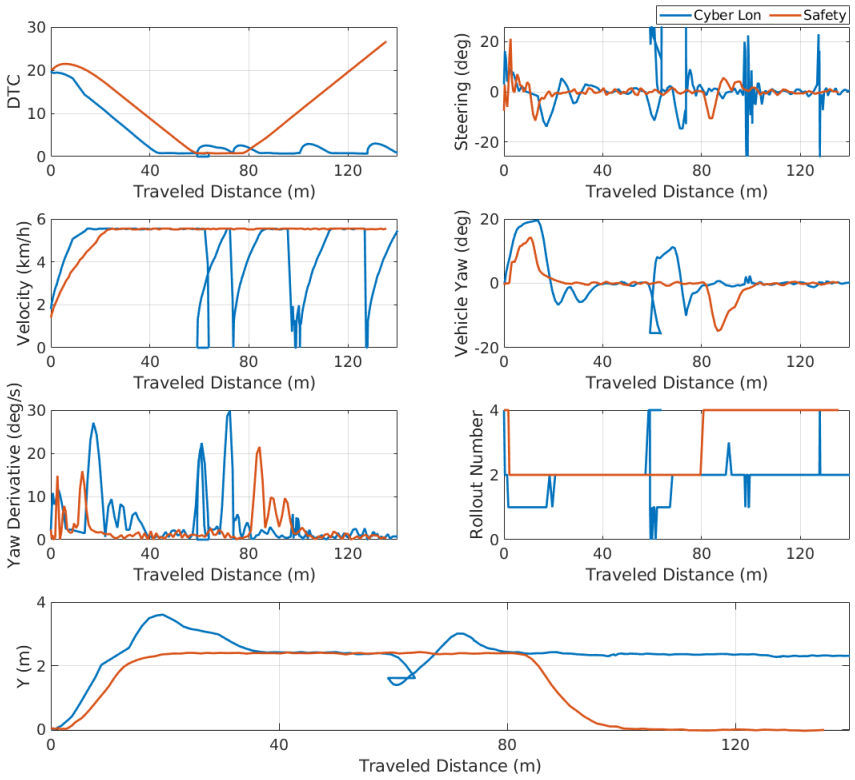


Figure 21: Longitudinal position offset attack vehicle parameters

which represent the effective position of the vehicle on the road. The frequency of these transitions affects the smoothness of the steering angle. Regarding the distance-to-collision parameter, it is observed that the effect of the attack is most prominent during the overtaking maneuver, particularly during the cut-in process, when the vehicle cuts-in front of the passing vehicle (NPC). Assertion 2 contends that when the vehicle transitions across multiple roll-outs, demonstrates a 180° steering angle, and approaches within less than 0.5m of the passing vehicle, it signifies behavior affected by the cyber attack.

Assertion 2: To identify vehicle dynamic changes from cyber-attack: *AV.x* − *NPC.x* < *distance_threshold && AV.lane_transition* > *max_transition_number && AV.steer_angle* ∉ *[min, max]_steer_angle*.

Assertion 3 contends with behavior observed in the longitudinal position offset (Fig. 21) wherein the AV collides with the passing vehicle and subsequently accelerates to its previous set-point.

Assertion 3: To identify collisions, we can derive the assertion: $|AV.v_k - AV.v_{k+1}| > threshold$.

Assertion 3 can also serve to identify anomalies in GPS data. The GNSS spoof-

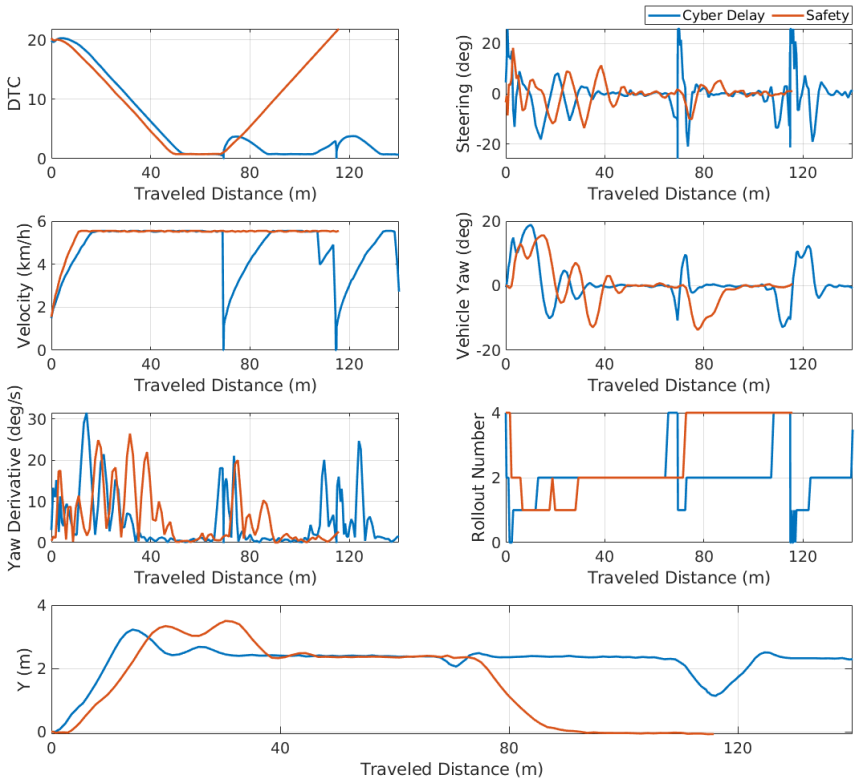


Figure 22: Delay message attack vehicle parameters

ing attack exhibits a considerable deviation in the altitude and position co-variance parameters. Assuming that velocity data originates from two sources, a wheel sensor measurement and calculated by deriving the position from GPS data, the results from both sources should closely align. In the case of a GNSS spoofing attack, the deviation in the position co-variance would lead to a spike in velocity (calculated by deriving position from GPS data), thus violating assertion 3.

For our specific AD system, the threshold for assertion 1 is a yaw angle displacement of 15° within a 1-second duration. The threshold for assertion 2 is identified as a distance between the AV and passing vehicle of less than 0.5m, a lane transition exceeding 1 roll-out, and a steering angle beyond the bounds of 20 and -20°. It is important to note that these values are applicable for a low-speed AV ride-hailing service, and designers of different classes of vehicles must calculate values consistent with their specific application.

Solvable bugs originate from various points in the controller. One common issue arises from incorrect or imprecise saturation values of the control signal, leading to excessive acceleration or a high steering angle in the vehicle. This phenomenon is clearly depicted in Fig. 21, where a signal overshoot results in multiple lane changes by the vehicle. Another notable example, as evident in Fig. 20, 21 and 22 is the absence of a fallback plan. A distinct indication of a collision occurs when the vehicle speed suddenly drops to 0ms^{-1} and then rapidly accelerates back to the reference point, thus violating Assertion 3. A robust controller should incorporate a fallback plan to address such scenarios, indicating a bug in the functional design of the controller. In such scenarios,

the vehicle should recognize that it can no longer adhere to the global trajectory and switch to emergency mode.

The primary objective of identifying unexpected behaviors is to debug the controller. Upon analyzing the experimental results, a violation of Assertion 1 can be associated with a bug within the `/ndt_pose` module (refer to Fig. 17), while a violation of Assertion 2 can be backpropagated to the module `/op_trajectory_evaluator`. Likewise, a violation of assertion 3 can be backpropagated to the modules of `/op_trajectory_generator` and `/op_behavior_selector` (refer to Fig. 17). To precisely pinpoint the violation of assertion 3 to a specific function, we abstracted from the local_planner algorithm and its substituent lane_rule algorithm, the `getClosestWaypointNumber` method. This method selects the next waypoint to follow in the global trajectory and returns an exception to be handled as a different driving behavior (*e.g.*, there was a crash, emergency mode activated).

In the case of a GNSS attack, the NDT localization algorithm fails to detect the deviation in position co-variance, attributed to the normal vector pointing in the same direction. Debugging efforts prioritize optimizing the NDT localization by leveraging visual odometry to maintain the local position at short-distances until the disturbance source is resolved.

6.5 Conclusions

In this chapter, with the aim of studying assertion-based verification in the context of security and security verification, two innovative methods were presented.

In section 6.1, a method was introduced for generating security assertions tailored to a RISC-V processor. The method involves systematically analyzing design specifications and requirements to pinpoint critical security properties, neglecting which could result in design vulnerabilities. By identifying the pertinent security properties and the associated design signals, the proposed security-based assertion miner is applied to the processor's simulation trace. The miner automatically outputs security assertions. The experiments demonstrate that these assertions proficiently encapsulate the behavior described by identified security properties, effectively detecting injected Hardware Trojans within the design.

On the other hand, in section 6.2, a method called ADAssure was presented for debugging and bug localization of autonomous driving control algorithms in AVs. ADAssure comprises three phases 1) AD Data collection 2) Assertion Rule Generation and 3) Assertion Review and Debugging. The concept behind ADAssure is to mine association rules from AD data, which can be transformed into assertions to detect the vulnerabilities in the system. Evaluation of ADAssure using various cyber-security datasets from both simulation and real-world scenarios revealed that the ADAssure could provide three assertions on the vulnerability of the `OpenPlanner 2.5` AD planning algorithm. These assertions provide valuable guidance to algorithm designers and safety engineers in pinpointing specific modules within the planning algorithm for debugging purposes.

7 Conclusions and Future Directions

In the current technological landscape, hardware designs and embedded systems are progressively growing in complexity with each iteration. This complexity often introduces bugs and errors that can compromise the functionality of these designs. To address this challenge and verify the correctness of the functionality of hardware designs, various functional verification techniques are available, including static verification, dynamic verification, and assertion-based verification methods.

In this thesis, with the aim of enhancing the assertion-based verification techniques, several innovative methods were proposed. These solutions and methods are applicable in the contexts of both functional verification and security verification and can be utilized by both academicians and engineers.

In this thesis, an automatic assertion miner, ARTmine, was proposed, utilizing data mining and association rule mining techniques for generating assertions. The experimental results demonstrated that ARTmine effectively generates accurate assertion sets, outperforming leading assertion miners, and enhancing the functional verification of hardware designs.

Furthermore, with the aim of enhancing the security verification of hardware designs, in this thesis, the utilization of assertions in the context of security verification was studied. To this end, for the first time, an automatic security-based assertion miner was presented that is able to generate a set of security assertions. While in this thesis, the proposed miner examined Hardware Trojan detection on RISC-V processors, it is not limited to only this type of processor and is applicable to other types of processors and various hardware designs. Additionally, it is not limited to only the detection of Hardware Trojans and can be utilized for the detection of other kinds of security vulnerabilities and attacks. Moreover, the application of security assertions was studied in the context of autonomous vehicles, and a method called ADAssure was proposed for automatically generating security assertions for debugging and bug localization of autonomous driving control algorithms within AVs.

Given the fact that the generated assertions for functional verification can suffer from redundancy and inconsistency and also the importance of evaluating the quality of assertions and selecting the best and highest-quality ones for utilization in the verification process, two novel techniques were proposed. For the first time, a data mining-based solution called Dominance was introduced, able to evaluate the quality of assertions and select the highest-quality ones, which can provide more design behavior coverage. Moreover, Dominance can integrate various rankings of other evaluation methods and provide a unique set of high-quality assertions. Furthermore, for the first time, in this thesis, an innovative method named IMMizer was proposed for identifying redundant and inconsistent assertions and minimizing them. The experimental findings showcased that IMMizer can significantly reduce and minimize the number of assertions, while the minimized set can provide the same design behavior coverage as the initial assertions did. Although in this thesis Dominance and IMMizer were studied in the context of functional verification, they can easily be utilized in the realm of security verification as well.

Overall, this PhD thesis, by proposing several novel solutions and methods, significantly enhanced both the functional verification and security verification of hardware designs. For future research studies, considering the importance and utilization of quantum hardware designs and circuits in recent years, and the considerable efficiency of the proposed solutions in this thesis, it is remarkable to apply them to the context of quantum hardware verification. Additionally, the proposed solutions can be studied and

applied to broader security and security verification domains for analyzing the detection of various types of security vulnerabilities and attacks. Last but not least, as hardware test techniques such as Built-in Self-Test (BIST), Memory Built-in Self-Test (MBIST), and fuzz testing approaches are struggling with huge amounts of test vectors, the data mining-based solutions presented in this PhD thesis are applicable to significantly optimize the test vectors in these kinds of testing techniques.

List of Figures

List of Tables

References

[1] H. Kopetz and W. Steiner, *Real-time systems: design principles for distributed embedded applications*. Springer Nature, 2022.

[2] P. Marwedel, *Embedded system design: embedded systems foundations of cyber-physical systems, and the internet of things*. Springer Nature, 2021.

[3] S. Cuenca-Asensi, A. Martínez-Álvarez, F. Restrepo-Calle, F. R. Palomo, H. Guzmán-Miranda, and M. A. Aguirre, "Soft core based embedded systems in critical aerospace applications," *Journal of Systems Architecture*, vol. 57, no. 10, pp. 886–895, 2011.

[4] G. Dere, "Biomedical applications with using embedded systems," in *Data acquisition-recent advances and applications in biomedical engineering*, IntechOpen, 2021.

[5] C. Salloum, M. Elshuber, O. Höftberger, H. Isakovic, and A. Wasicek, "The across MPSoC - a new generation of multi-core processors designed for safety-critical embedded systems," vol. 37, 09 2012.

[6] M. R. H. Iman and P. Yaghmaie, "A software control flow checking technique in multi-core processors," *International Journal of Embedded Systems*, vol. 13, no. 2, pp. 136–147, 2020.

[7] H. Witharana, Y. Lyu, S. Charles, and P. Mishra, "A survey on assertion-based hardware verification," *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–33, 2022.

[8] T. Grimm, D. Lettnin, and M. Hübner, "A survey on formal verification techniques for safety-critical systems-on-chip," *Electronics*, vol. 7, no. 6, p. 81, 2018.

[9] L. C. Cordeiro, E. B. de Lima Filho, and I. V. Bessa, "Survey on automated symbolic verification and its application for synthesising cyber-physical systems," *IET Cyber-Physical Systems: Theory & Applications*, vol. 5, no. 1, pp. 1–24, 2020.

[10] O. Hasan and S. Tahar, "Formal verification methods," in *Encyclopedia of Information Science and Technology, Third Edition*, pp. 7162–7170, IGI global, 2015.

[11] H. Li, Q. Liu, and J. Zhang, "A survey of hardware trojan threat and defense," *Integration*, vol. 55, pp. 426–437, 2016.

[12] T. Kropf, *Introduction to Formal Hardware Verification: Methods and Tools for Designing Correct Circuits and Systems*. Berlin, Heidelberg: Springer-Verlag, 1st ed., 1999.

[13] J. Zhang, F. Yuan, L. Wei, Z. Sun, and Q. Xu, "VeriTrust: Verification for hardware trust," in *Proc. of the 50th Annual Design Automation Conference*, pp. 1–8, 2013.

[14] W. Chen, S. Ray, M. Abadir, J. Bhadra, and L.-C. Wang, "Challenges and trends in modern SoC design verification," *IEEE Design Test*, vol. PP, pp. 1–1, 08 2017.

[15] H. Foster, "The 2020 Wilson Research Group Functional Verification Study." https://blogs.sw.siemens.com/verificationhorizons/2020/11/05/part-1-the-2020-wilson-research-group-functional-verification-study/, 2020. Retrievd on March 18, 2024.

[16] T. Ghasempouri, J. Raik, K. Paul, C. Reinbrecht, S. Hamdioui, and M. Taouil, "A security verification template to assess cache architecture vulnerabilities," in *Proc. of the 2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, pp. 1–6, 2020.

[17] X. Guo, R. G. Dutta, Y. Jin, F. Farahmandi, and P. Mishra, "Pre-silicon security verification and validation: A formal perspective," in *Proc. of the 52nd annual design automation conference*, pp. 1–6, 2015.

[18] F. Erata, S. Deng, F. Zaghloul, W. Xiong, O. Demir, and J. Szefer, "Survey of approaches and techniques for security verification of computer systems," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 19, no. 1, pp. 1–34, 2023.

[19] A. Nahiyan, M. Sadi, R. Vittal, G. Contreras, D. Forte, and M. Tehranipoor, "Hardware trojan detection through information flow security verification," in *Proc. of the 2017 IEEE International Test Conference (ITC)*, pp. 1–10, IEEE, 2017.

[20] W. Khan, M. Kamran, S. R. Naqvi, F. A. Khan, A. S. Alghamdi, and E. Alsolami, "Formal verification of hardware components in critical systems," *Wireless Communications and Mobile Computing*, vol. 2020, pp. 1–15, 2020.

[21] R. Tao, J. Yao, X. Li, S.-W. Li, J. Nieh, and R. Gu, "Formal verification of a multiprocessor hypervisor on arm relaxed memory hardware," in *Proc. of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pp. 866–881, 2021.

[22] D. Große, G. Fey, and R. Drechsler, "Enhanced formal verification flow for circuits integrating debugging and coverage analysis," *Electronic Communications of the EASST*, vol. 62, 2013.

[23] R. Drechsler and G. Fey, "Formal verification meets robustness checking—techniques and challenges," in *Proc. of the 13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pp. 4–4, IEEE, 2010.

[24] H. Riener and G. Fey, "Faust: A framework for formal verification, automated debugging, and software test generation," in *Proc. of the International SPIN Workshop on Model Checking of Software*, pp. 234–240, Springer, 2012.

[25] G. Fey, "Assessing system vulnerability using formal verification techniques," in *Proc. of the International Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, pp. 47–56, Springer, 2011.

[26] M. Loghi, T. Margaria, G. Pravadelli, and B. Steffen, "Dynamic and formal verification of embedded systems: A comparative survey," *International Journal of Parallel Programming*, vol. 33, pp. 585–611, 2005.

[27] J. Kapinski, J. V. Deshmukh, X. Jin, H. Ito, and K. Butts, "Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques," *IEEE Control Systems Magazine*, vol. 36, no. 6, pp. 45–64, 2016.

[28] A. Kamkin, "Simulation-based hardware verification with time-abstract models," in *Proc. of the 2011 9th East-West Design Test Symposium (EWDTS)*, pp. 43–47, 2011.

[29] K. Datta and P. P. Das, "Assertion based verification using HDVL," in *Proc. of the 17th International Conference on VLSI Design. Proceedings.*, pp. 319–325, IEEE, 2004.

[30] K. Peterson and Y. Savaria, "Assertion-based on-line verification and debug environment for complex hardware systems," in *Proc. of the 2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No. 04CH37512)*, vol. 2, pp. II–685, IEEE, 2004.

[31] Y. Li, W. Wu, L. Hou, and H. Cheng, "A study on the assertion-based verification of digital IC," in *Proc. of the 2009 Second International Conference on Information and Computing Science*, vol. 2, pp. 25–28, IEEE, 2009.

[32] M. W. Anwar, M. Rashid, F. Azam, A. Naeem, M. Kashif, and W. H. Butt, "A unified model-based framework for the simplified execution of static and dynamic assertion-based verification," *IEEE Access*, vol. 8, pp. 104407–104431, 2020.

[33] K. Alatoun, B. Shankaranarayanan, S. M. Achyutha, and R. Vemuri, "SoC trust validation using assertion-based security monitors," in *Proc. of the 2021 22nd International Symposium on Quality Electronic Design (ISQED)*, pp. 496–503, IEEE, 2021.

[34] R. Drechsler, *Advanced formal verification*. Springer, 2004.

[35] G. Fey and R. Drechsler, "Improving simulation-based verification by means of formal methods," in *Proc. of the ASP-DAC 2004: Asia and South Pacific Design Automation Conference 2004 (IEEE Cat. No.04EX753)*, pp. 640–643, 2004.

[36] N. Bombieri, F. Fummi, and G. Pravadelli, "Hardware design and simulation for verification," in *Formal Methods for Hardware Verification* (M. Bernardo and A. Cimatti, eds.), (Berlin, Heidelberg), pp. 1–29, Springer Berlin Heidelberg, 2006.

[37] L. Zhao, S. Chen, and Y. Wang, "Simulation-based hardware verification with a graph-based specification," *Mathematical Problems in Engineering*, vol. 2018, p. 10, 2018.

[38] A. Kamkin and M. Chupilko, "Survey of modern technologies of simulation-based verification of hardware," *Programming and Computer Software*, vol. 37, pp. 147–152, 05 2011.

[39] Y. Tao, "An introduction to assertion-based verification," in *Proc. of the 2009 IEEE 8th International Conference on ASIC*, pp. 1318–1323, 2009.

[40] H. Sohofi and Z. Navabi, "Assertion-based verification for system-level designs," in *Proc. of the Fifteenth International Symposium on Quality Electronic Design*, pp. 582–588, IEEE, 2014.

[41] S. Lammermann, J. Ruf, T. Kropf, W. Rosenstiel, A. Viehl, A. Jesser, and L. Hedrich, "Towards assertion-based verification of heterogeneous system designs," in *Proc. of the 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pp. 1171–1176, IEEE, 2010.

[42] A. Danese, N. D. Riva, and G. Pravadelli, "A-TEAM: Automatic template-based assertion miner," in *Proc. of the 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2017.

[43] S. Germiniani, *A complete assertion-based verification framework from the edge to the cloud*. PhD thesis, University of Verona, Verona, Italy, 2023.

[44] L. Pierre, F. Pancher, R. Suescun, and J. Quévremont, "On the effectiveness of assertion-based verification in an industrial context," in *Formal Methods for Industrial Critical Systems* (C. Pecheur and M. Dierkes, eds.), (Berlin, Heidelberg), pp. 78–93, Springer Berlin Heidelberg, 2013.

[45] P. Yeung and K. Larsen, "Practical assertion-based formal verification for SoC designs," in *Proc. of the 2005 International Symposium on System-on-Chip*, pp. 58–61, 2005.

[46] P. Gurha and R. R. Khandelwal, "Systemverilog assertion based verification of AMBA-AHB," in *Proc. of the 2016 International Conference on Micro-Electronics and Telecommunication Engineering (ICMETE)*, pp. 641–645, 2016.

[47] A. M. Gharehbaghi, B. H. Yaran, S. Hessabi, and M. Goudarzi, "An assertion-based verification methodology for system-level design," *Computers Electrical Engineering*, vol. 33, no. 4, pp. 269–284, 2007. Hardware/Software System on Chip Co-design: Approach and Application.

[48] B. Niemann and C. Haubelt, "Assertion-based verification of transaction level models," in *Proc. of the Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, 2006.

[49] A. Ghofrani, F. Javaheri, and Z. Navabi, "Assertion based verification in TLM," pp. 509 – 513, 10 2010.

[50] A. Habibi, S. Tahar, A. Samarah, D. Li, and O. Mohamed, "Efficient assertion based verification using TLM," in *Proc. of the Design, Automation & Test in Europe Conference*, vol. 1, pp. 1–6, 2006.

[51] L. Ferro, L. Pierre, Y. Ledru, and L. du Bousquet, "Generation of test programs for the assertion-based verification of TLM models," in *Proc. of the 2008 3rd International Design and Test Workshop*, pp. 237–242, 2008.

[52] T. Ghasempouri, J. Malburg, A. Danese, G. Pravadelli, G. Fey, and J. Raik, "Engineering of an effective automatic dynamic assertion mining platform," in *Proc. of the 2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 111–116, 2019.

[53] F. Erata, S. Deng, F. Zaghloul, W. Xiong, O. Demir, and J. Szefer, "Survey of approaches and techniques for security verification of computer systems," *J. Emerg. Technol. Comput. Syst.*, vol. 19, jan 2023.

[54] H. Witharana, Y. Lyu, and P. Mishra, "Directed test generation for activation of security assertions in RTL models," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 26, jan 2021.

[55] B. Wile, J. Goss, and W. Roesner, *Comprehensive Functional Verification: The Complete Industry Cycle*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[56] N. Bombieri, F. Fummi, and G. Pravadelli, "At-speed functional verification of programmable devices," in *Proc. of the 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2004. DFT 2004. Proceedings.*, pp. 386–394, 2004.

[57] M. Boulé and Z. Zilic, *Generating Hardware Assertion Checkers: For Hardware Verification, Emulation, Post-Fabrication Debugging and On-Line Monitoring*. Springer, 2008.

[58] A. Danese, F. Filini, T. Ghasempouri, and G. Pravadelli, "Automatic generation and qualification of assertions on control signals: A time window-based approach," in *VLSI-SoC: Design for Reliability, Security, and Low Power* (Y. Shin, C. Y. Tsui, J.-J. Kim, K. Choi, and R. Reis, eds.), (Cham), pp. 193–221, Springer International Publishing, 2016.

[59] M. Jenihhin, J. Raik, R. Ubar, and A. Chepurov, "On reusability of verification assertions for testing," in *Proc. of the 2008 11th International Biennial Baltic Electronics Conference*, pp. 151–154, 2008.

[60] X. T. Ngo, J.-L. Danger, S. Guilley, Z. Najm, and O. Emery, "Hardware property checker for run-time hardware trojan detection," in *Proc. of the 2015 European Conference on Circuit Theory and Design (ECCTD)*, pp. 1–4, 2015.

[61] *Property Generation*, pp. 75–97. Dordrecht: Springer Netherlands, 2008.

[62] S. Katz, O. Grumberg, and D. Geist, "Have I written enough properties? A method of comparison between specification and implementation," in *ACM CHARME*, pp. 280–297, 1999.

[63] S. Hertz, D. Sheridan, and S. Vasudevan, "Mining hardware assertions with guidance from static analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 6, pp. 952–965, 2013.

[64] S. Vasudevan, D. Sheridan, S. Patel, D. Tcheng, B. Tuohy, and D. Johnson, "Goldmine: automatic assertion generation using data mining and static analysis," in *Proc. of the 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pp. 626–629, 2010.

[65] J. Malburg, T. Flenker, and G. Fey, "Property mining using dynamic dependency graphs," in *Proc. of the 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 244–250, Jan 2017.

[66] G. Martino and G. Fey, "Syntax-guided enumeration of temporal properties," in *Proc. of the 2019 Forum for Specification and Design Languages (FDL)*, pp. 1–8, 2019.

[67] M. Soeken, U. Kühne, M. Freibothe, G. Fe, and R. Drechsler, "Automatic property generation for the formal verification of bus bridges," in *Proc. of the 14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pp. 417–422, 2011.

[68] J. Malburg, T. Flenker, and G. Fey, "Generating good properties from a small number of use cases," in *Proc. of the IEEE International Verification and Security Workshop*, 07 2016.

[69] F. Rogin, T. Klotz, G. Fey, R. Drechsler, and S. Rülke, "Advanced verification by automatic property generation," *IET Comput. Digit. Tech.*, vol. 3, no. 4, pp. 338–353, 2009.

[70] F. Rogin, T. Klotz, G. Fey, R. Drechsler, and S. Rülke, "Automatic generation of complex properties for hardware designs," in *Proc. of the Conference on Design, Automation and Test in Europe*, DATE '08, (New York, NY, USA), p. 545–548, Association for Computing Machinery, 2008.

[71] M. R. Heidari Iman, J. Raik, M. Jenihhin, G. Jervan, and T. Ghasempouri, "A methodology for automated mining of compact and accurate assertion sets," in *Proc. of the 2021 IEEE Nordic Circuits and Systems Conference (NorCAS)*, pp. 1–7, 2021.

[72] S. Germiniani and G. Pravadelli, "HARM: A hint-based assertion miner," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4277–4288, 2022.

[73] S. Germiniani and G. Pravadelli, "Exploiting clustering and decision-tree algorithms to mine ltl assertions containing non-boolean expressions," in *Proc. of the 2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, 2022.

[74] S. Vasudevan, L. Liu, and S. Hertz, *A Comparative Study of Assertion Mining Algorithms in GoldMine*, pp. 609–645. Cham: Springer International Publishing, 2019.

[75] D. Sheridan, L. Liu, and S. Vasudevan, "Goldmine : Automatic assertion generation and coverage closure in design validation," 2011.

[76] M. R. Heidari Iman, J. Raik, G. Jervan, and T. Ghasempouri, "IMMizer: An innovative cost-effective method for minimizing assertion sets," in *Proc. of the 2022 25th Euromicro Conference on Digital System Design (DSD)*, pp. 1–8, 2022.

[77] M. R. Heidari Iman, J. Raik, M. Jenihhin, G. Jervan, and T. Ghasempouri, "An automated method for mining high-quality assertion sets," *Microprocessors and Microsystems*, vol. 97, p. 104773, 2023.

[78] T. Ghasempouri and G. Pravadelli, "On the estimation of assertion interestingness," in *Proc. of the 2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 325–330, Oct 2015.

[79] T. Ghasempouri, S. Payandeh Azad, B. Niazmand, and J. Raik, "An automatic approach to evaluate assertions' quality based on data-mining metrics," in *Proc. of the 2018 IEEE International Test Conference in Asia (ITC-Asia)*, pp. 61–66, 2018.

[80] R. Hariharan, T. Ghasempouri, B. Niazmand, and J. Raik, "From RTL liveness assertions to cost-effective hardware checkers," in *Proc. of the 2018 Conference on Design of Circuits and Integrated Systems (DCIS)*, pp. 1–6, IEEE, 2018.

[81] D. Pal, S. Offenberger, and S. Vasudevan, "Assertion ranking using RTL source code analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 8, pp. 1711–1724, 2020.

[82] G. Fey, T. Ghasempouri, S. Jacobs, G. Martino, J. Raik, and H. Riener, "Design understanding: From logic to specification," in *Proc. of the 2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 172–175, 2018.

[83] Y. Lyu and P. Mishra, "System-on-chip security assertions," *CoRR*, vol. abs/2001.06719, 2020.

[84] P. Bhamidipati, S. M. Achyutha, and R. Vemuri, "Security analysis of a system-on-chip using assertion-based verification," in *Proc. of the 2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 826–831, 2021.

[85] A. Ayalasomayajula, N. Farzana, M. Tehranipoor, and F. Farahmandi, "Automatic asset identification for assertion-based SoC security verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2024.

[86] C. Wang, Y. Cai, Q. Zhou, and H. Wang, "ASAX: Automatic security assertion extraction for detecting hardware trojans," in *Proc. of the 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 84–89, 2018.

[87] S. Germiniani, A. Danese, and G. Pravadelli, "Automatic generation of assertions for detection of firmware vulnerabilities through alignment of symbolic sequences," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 2, pp. 728–739, 2022.

[88] C. S. Chuah, C. Appold, and T. Leinmueller, "Formal verification of security properties on RISC-V processors," in *Proc. of the 2023 21st ACM-IEEE International Symposium on Formal Methods and Models for System Design (MEMOCODE)*, pp. 159–168, 2023.

[89] R. Zhang and C. Sturton, "Transys: Leveraging common security properties across hardware designs," in *Proc. of the 2020 IEEE Symposium on Security and Privacy (SP)*, pp. 1713–1727, 2020.

[90] C. Deutschbein, A. Meza, F. Restuccia, R. Kastner, and C. Sturton, "Isadora: Automated information flow property generation for hardware designs," in *Proc. of the 5th Workshop on Attacks and Solutions in Hardware Security*, ASHES '21, p. 5–15, Association for Computing Machinery, 2021.

[91] N. Dong, R. Guanciale, M. Dam, and A. Lööw, "Formal verification of correctness and information flow security for an in-order pipelined processor," in *Proc. of the Formal Methods in Computer-Aided Design (FMCAD)*, pp. 247–256, 2023.

[92] C. Harper, G. Chance, A. Ghobrial, S. Alam, T. Pipe, and K. Eder, "Safety validation of autonomous vehicles using assertion checking," *arXiv preprint arXiv:2111.04611*, 2021.

[93] C. J. Harper, G. Chance, A. Ghobrial, S. Alam, A. G. Pipe, and K. I. Eder, "Safety validation of autonomous vehicles using assertion-based oracles," *ArXiv*, vol. abs/2111.04611, 2021.

[94] M. Schwammberger, C. J. Harper, G. V. Alves, G. Chance, A. G. Pipe, and K. I. Eder, "Integrating formal verification and simulation-based assertion checking in a corroborative v&v process," *ArXiv*, vol. abs/2208.05273, 2022.

[95] A. Domenici, A. Fagiolini, and M. Palmieri, "Integrated simulation and formal verification of a simple autonomous vehicle," in *Proc. of the SEFM Workshops*, 2017.

[96] M. W. Anwar, M. Rashid, F. Azam, A. Naeem, M. Kashif, and W. H. Butt, "A unified model-based framework for the simplified execution of static and dynamic assertion-based verification," *IEEE Access*, vol. 8, pp. 104407–104431, 2020.

[97] S. Azzopardi, C. Colombo, and G. Pace, "A model-based approach to combining static and dynamic verification techniques," in *Proc. of the International Symposium on Leveraging Applications of Formal Methods*, pp. 416–430, Springer, 2016.

[98] L.-W. Kim and J. D. Villasenor, "Dynamic function verification for system on chip security against hardware-based attacks," *IEEE Transactions on Reliability*, vol. 64, no. 4, pp. 1229–1242, 2015.

[99] H. D. Foster, A. C. Krolnik, and D. J. Lacey, *Assertion-based design*. Springer Science & Business Media, 2004.

[100] G. Pravadelli, D. Quaglia, S. Vinco, F. Fummi, *et al.*, "Semiformal assertion-based verification of hardware/software systems in a model-driven design framework," in *Handbook of Hardware/Software Codesign*, pp. 1–38, Springer Netherlands, 2017.

[101] C. Seger, "An introduction to formal hardware verification," tech. rep., CAN, 1992.

[102] T. F. Melham, *Higher Order Logic and Hardware Verification*. USA: Cambridge University Press, 1st ed., 2009.

[103] C. Kern and M. R. Greenstreet, "Formal verification in hardware design: a survey," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 4, p. 123–193, apr 1999.

[104] G. A. Alvarez and F. Cristian, "Simulation-based testing of communication protocols for dependable embedded systems," *The Journal of Supercomputing*, vol. 16, pp. 93–116, 2000.

[105] M. Lajolo, L. Lavagno, M. Rebaudengo, M. S. Reorda, and M. Violante, "Automatic test bench generation for simulation-based validation," in *Proc. of the eighth international workshop on Hardware/software codesign*, pp. 136–140, 2000.

[106] H. Foster *et al.*, "Applied assertion-based verification: An industry perspective," *Foundations and Trends® in Electronic Design Automation*, vol. 3, no. 1, pp. 1–95, 2009.

[107] A. Dahan, D. Geist, L. Gluhovsky, D. Pidan, G. Shapir, Y. Wolfsthal, L. Benalycherif, R. Kamidem, and Y. Lahbib, "Combining system level modeling with assertion based verification," in *Proc. of the Sixth international symposium on quality electronic design (isqed'05)*, pp. 310–315, 2005.

[108] G. Di Guglielmo, L. Di Guglielmo, F. Fummi, and G. Pravadelli, "On the use of assertions for embedded-software dynamic verification," in *Proc. of the 2012 IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pp. 330–335, IEEE, 2012.

[109] "Standard for property specification language (PSL), ieee standard," p. 1–184, 2012.

[110] T. Ghasempouri, J. Raik, C. Reinbrecht, S. Hamdioui, and M. Taouil, "Survey on architectural attacks: A unified classification and attack model," *ACM Comput. Surv.*, vol. 56, sep 2023.

[111] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *Proc. of the 2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1–19, 2019.

[112] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *Proc. of the 27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), pp. 973–990, USENIX Association, Aug. 2018.

[113] A. Danese, T. Ghasempouri, and G. Pravadelli, "Automatic extraction of assertions from execution traces of behavioural models," in *Proc. of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 67–72, 2015.

[114] M. Bertasi, G. Di Guglielmo, and G. Pravadelli, "Automatic generation of compact formal properties for effective error detection," in *Proc. of the ACM/IEEE CODES+ISSS*, pp. 1–10, 2013.

[115] J. S. Bradbury, J. R. Cordy, and J. Dingel, "ExMAn: A generic and customizable framework for experimental mutation analysis," in *Proc. of the Second Workshop on Mutation Analysis (Mutation 2006 - ISSRE Workshops 2006)*, pp. 4–4, 2006.

[116] Y. Zhang and A. Mesbah, "Assertions are strongly correlated with test suite effectiveness," in *Proc. of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, (New York, NY, USA), p. 214–224, Association for Computing Machinery, 2015.

[117] T. Xie, N. Tillmann, J. de Halleux, and W. Schulte, "Mutation analysis of parameterized unit tests," in *Proc. of the 2009 International Conference on Software Testing, Verification, and Validation Workshops*, pp. 177–181, 2009.

[118] T. Zhao, E. Yurtsever, J. A. Paulson, and G. Rizzoni, "Formal certification methods for automated vehicle safety assessment," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 232–249, 2023.

[119] B. Y. Ekren, S. S. Heragu, A. Krishnamurthy, and C. J. Malmborg, "Simulation based experimental design to identify factors affecting performance of AVS/RS," *Computers & Industrial Engineering*, vol. 58, no. 1, pp. 175–185, 2010.

[120] J. Wang, H. Huang, K. Li, and J. Li, "Towards the unified principles for level 5 autonomous vehicles," *Engineering*, vol. 7, no. 9, pp. 1313–1325, 2021.

[121] A. Soteropoulos, M. Mitteregger, M. Berger, and J. Zwirchmayr, "Automated drivability: Toward an assessment of the spatial deployment of level 4 automated vehicles," *Transportation Research Part A: Policy and Practice*, vol. 136, pp. 64–84, 2020.

[122] T. Nguyen and D. Ničković, "Assertion-based monitoring in practice – checking correctness of an automotive sensor interface," *Science of Computer Programming*, vol. 118, pp. 40–59, 2016. Formal Methods for Industrial Critical Systems (FMICS'2014).

[123] G. Chance, A. Ghobrial, S. Lemaignan, T. Pipe, and K. Eder, "An agency-directed approach to test generation for simulation-based autonomous vehicle verification," in *Proc. of the 2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*, pp. 31–38, 2020.

[124] C. Medrano-Berumen and M. I. Akbaş, "Abstract simulation scenario generation for autonomous vehicle verification," in *Proc. of the 2019 SoutheastCon*, pp. 1–6, 2019.

[125] J. Lygeros, D. N. Godbole, and S. Sastry, "Verified hybrid controllers for automated vehicles," *IEEE transactions on automatic control*, vol. 43, no. 4, pp. 522–539, 1998.

[126] Y. Zhou, Y. Sun, Y. Tang, Y. Chen, J. Sun, C. M. Poskitt, Y. Liu, and Z. Yang, "Specification-based autonomous driving system testing," *IEEE Transactions on Software Engineering*, 2023.

[127] E.-Y. Kang, D. Mu, L. Huang, and Q. Lan, "Model-based verification and validation of an autonomous vehicle system," *arXiv preprint arXiv:1803.06103*, 2018.

[128] R. Passerone, D. Cancila, M. Albano, S. Mouelhi, S. Plosz, E. Jantunen, A. Ryabokon, E. Laarouchi, C. Hegedűs, and P. Varga, "A methodology for the design of safety-compliant and secure communication of autonomous vehicles," *IEEE Access*, vol. 7, pp. 125022–125037, 2019.

[129] M. R. Heidari Iman, G. Jervan, and T. Ghasempouri, "ARTmine: Automatic association rule mining with temporal behavior for hardware verification," in *2024 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1–6, 2024.

[130] C. Antunes and A. L. Oliveira, "Temporal data mining: an overview," 2001.

[131] S. Bilqisth and K. Mustofa, "Determination of temporal association rules pattern using apriori algorithm," *IJCCS (Indonesian Journal of Computing and Cybernetics Systems)*, vol. 14, p. 159, 04 2020.

[132] J. Han, M. Kamber, and J. Pei, "6 - mining frequent patterns, associations, and correlations: Basic concepts and methods," pp. 243–278, 2012.

[133] M. Shahin, M. R. Heidari Iman, M. Kaushik, R. Sharma, T. Ghasempouri, and D. Draheim, "Exploring factors in a crossroad dataset using cluster-based association rule mining," *Procedia Computer Science*, vol. 201, pp. 231–238, 2022. Proc. of the 13th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 5th International Conference on Emerging Data and Industry 4.0 (EDI40).

[134] M. R. H. Iman, P. Chikul, G. Jervan, H. Bahsi, and T. Ghasempouri, "Anomalous file system activity detection through temporal association rule mining," in *Proc. of the 9th International Conference on Information Systems Security and Privacy (ICISSP)*, pp. 733–740, 2023.

[135] C. Czepa, A. Amiri, E. Ntentos, and U. Zdun, "Modeling compliance specifications in linear temporal logic, event processing language and property specification patterns: a controlled experiment on understandability," *Software & Systems Modeling*, vol. 18, 12 2019.

[136] C. B. Spear, *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*. Springer, 2nd ed., 2010.

[137] T. A. Budd and F. G. Sayward, "Users guide to the pilot mutation system," techreport 114, Yale University, New Haven, Connecticut, 1977.

[138] T. Budd, R. Lipton, R. Demillo, and F. Sayward, "The design of a prototype mutation system for program testing (PDF)," *Proc. of the International Workshop on Managing Requirements Knowledge*, vol. 0, p. 623, 01 1978.

[139] T. Flenker, J. Malburg, G. Fey, S. Avramenko, M. Violante, and M. S. Reorda, "Towards making fault injection on abstract models a more accurate tool for predicting RT-level effects," in *Proc. of the 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 533–538, 2017.

[140] W. Wong, *Mutation Testing for the New Century*. 01 2001.

[141] A. J. Offutt and K. N. King, "A fortran 77 interpreter for mutation analysis," *ACM SIGPLAN Notices*, vol. 22, p. 177–188, jul 1987.

[142] A. J. Offutt and J. Pan, "Automatically detecting equivalent mutants and infeasible paths," *Software Testing, Verification and Reliability*, vol. 7, no. 3, pp. 165–192, 1997.

[143] R. Guderlei, R. Just, C. Schneckenburger, and F. Schweiggert, "Benchmarking testing strategies with tools from mutation analysis," in *Proc. of the 2008 IEEE International Conference on Software Testing Verification and Validation Workshop*, pp. 360–364, 2008.

[144] U. Repinski, H. Hantson, M. Jenihhin, J. Raik, R. Ubar, G. Di Guglielmo, G. Pravadelli, and F. Fummi, "Combining dynamic slicing and mutation operators for esl correction," in *Proc. of the 2012 17th IEEE European Test Symposium (ETS)*, pp. 1–6, 2012.

[145] Abrishami, "ISCAS'89." https://sportlab.usc.edu/~msabrishami/benchmarks.html. Retrievd on July 25, 2023.

[146] D. Pal, "Goldminer." https://bitbucket.org/debjitp/goldminer/src/master/example/. Retrievd on July 22, 2023.

[147] B. Niazmand, S. Payandeh Azad, and R. Hariharan, "Project Bonfire Network-on-Chip." https://github.com/Project-Bonfire/Bonfire_handshake, 2017. Retrievd on July 20, 2023.

[148] S. Bouker, R. Saidi, S. Ben Yahia, and E. Mephu Nguifo, "Mining undominated association rules through interestingness measures," *International Journal on Artificial Intelligence Tools*, vol. 23, no. 04, p. 1460011, 2014.

[149] S. Bouker, R. Saidi, S. B. Yahia, and E. M. Nguifo, "Ranking and selecting association rules based on dominance relationship," in *Proc. of the 2012 IEEE 24th International Conference on Tools with Artificial Intelligence*, vol. 1, pp. 658–665, 2012.

[150] S. L. Harris and D. M. Harris, "2 - combinational logic design," in *Digital Design and Computer Architecture* (S. L. Harris and D. M. Harris, eds.), pp. 54–106, Boston: Morgan Kaufmann, 2016.

[151] C. Deutschbein and C. Sturton, "Mining security critical linear temporal logic specifications for processors," in *Proc. of the 2018 19th International Workshop on Microprocessor and SoC Test and Verification (MTV)*, pp. 18–23, 2018.

[152] M. R. H. Iman, S. Ahmadi-Pour, R. Drechsler, and T. Ghasempouri, "Processor vulnerability detection with the aid of assertions: RISC-V case study," TechRxiv, July 2024. DOI: `http://dx.doi.org/10.36227/techrxiv.172101134.45466090/v1`.

[153] A. Roberts, M. R. Heidari Iman, M. Bellone, T. Ghasempouri, J. Raik, O. Maennel, M. Hamad, and S. Steinhorst, "ADAssure: Debugging methodology for autonomous driving control algorithms," in *2024 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1–6, 2024.

[154] "IseAuto – Autonomous Vehicle." https://autolab.taltech.ee/portfolio/iseauto/.

[155] A. Waterman, K. Asanovic, *et al.*, "The RISC-V instruction set manual volume i: Unprivileged isa," *Document Version*, vol. 20191213, 2019.

[156] A. Waterman, K. Asanovic, *et al.*, "The RISC-V instruction set manual, volume ii: Privileged architecture," *RISC-V Foundation*, 2019.

[157] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design RISC-V Edition*. The Morgan Kaufmann Series in Computer Architecture and Design, 2017.

[158] S. Ahmadi-Pour, V. Herdt, and R. Drechsler, "The MicroRV32 framework: An accessible and configurable open source RISC-V cross-level platform for education and research," *Journal of Systems Architecture*, vol. 133, p. 102757, 2022.

[159] S. Bhasin and F. Regazzoni, "A survey on hardware trojan detection techniques," in *Proc. of the 2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2021–2024, 2015.

[160] R. Zhang, N. Stanley, C. Griggs, A. Chi, and C. Sturton, "Identifying security critical properties for the dynamic verification of a processor," *SIGARCH Comput. Archit. News*, vol. 45, p. 541–554, apr 2017.

[161] B. Kumar, A. K. Jaiswal, V. S. Vineesh, and R. Shinde, "Analyzing hardware security properties of processors through model checking," in *Proc. of the 2020 33rd International Conference on VLSI Design and 2020 19th International Conference on Embedded Systems (VLSID)*, pp. 107–112, 2020.

[162] M. Hicks, C. Sturton, S. T. King, and J. M. Smith, "SPECS: A lightweight runtime mechanism for protecting software from security-critical processor bugs," in *Proc. of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, (New York, NY, USA), p. 517–529, Association for Computing Machinery, 2015.

[163] M. Zaki, "Scalable algorithms for association mining," *IEEE Transactions on Knowledge and Data Engineering*, 2000.

[164] H. Darweesh, E. Takeuchi, and K. Takeda, "Openplanner 2.0: The portable open source planner for autonomous driving applications," in *Proc. of the 2021 IEEE Intelligent Vehicles Symposium Workshops*, 2021.

Acknowledgements

I express my sincere gratitude to my parents and brothers for their unwavering support, love, and encouragement throughout my life and also during my PhD journey. Their support made pursuing my PhD dream possible, and I am deeply grateful for all the love they have shared with me.

I also extend my gratitude to my supervisors, Dr. Tara Ghasempouri and Dr. Gert Jervan, for providing me with the invaluable opportunity to embark on this PhD journey and for sharing their time and experience with me over the past four years.

My heartfelt appreciation goes out to my dear friends Behnoosh, Babak, Nzamba, Saghar, Tatiana, Roya, and Sara. Their unwavering support, kindness, and empathy have been a constant source of strength and encouragement for me throughout these years. I am immensely proud and fortunate to have such incredible friends, and I am forever grateful for each and every one of them.

I also express my gratitude to Tallinn University of Technology and the Department of Computer Systems for fostering a conducive educational environment throughout my PhD journey. Additionally, I am thankful to the beautiful country of Estonia and its hospitable and kind people for welcoming me and making my stay here enriching and memorable.

I would also like to express my appreciation to Professor Rolf Drechsler for extending his gracious invitation to visit his esteemed research group. Collaborating with such a brilliant team has been an enriching experience, and I am grateful for the opportunity he provided me. Moreover, special thanks go to all my collaborators and co-authors who have contributed to my publications; their expertise has been invaluable in the success of our joint endeavors.

Furthermore, I am deeply grateful to the opponents and committee members of my PhD defense for their constructive comments and valuable feedback on my thesis. Their insights and expertise have played a pivotal role in refining and shaping my PhD thesis.

I am also grateful for the financial support provided by the European Union through the European Social Fund in the frame of the "Information and Communication Technologies (ICT) programme" and the Estonian Research Council grants PSG837. Their funding support has been instrumental in facilitating my research endeavors over the past four years.

Last but not least, I extend my gratitude to all those who have supported and assisted me during my PhD journey, helping me realize my childhood dream.

Abstract
Enhancing Assertion-Based Verification in Hardware Designs through Data Mining Algorithms

In the field of functional hardware verification, there exists a range of techniques including static verification, dynamic verification, and assertion-based verification methods. While static and dynamic approaches show promise in functional verification, assertion-based verification techniques have garnered increasing interest. They offer the combined benefits of both static and dynamic techniques. Assertions play a pivotal role in assertion-based verification and are fundamental to its methodology. Given their significance in functional and even security verification, considerable efforts have been made in the literature to create high-quality assertion sets that offer comprehensive design behavior coverage.

These efforts can be broadly categorized into two main approaches: manual and automatic assertion creation. Manual assertion definition is time-consuming and prone to errors. Moreover, for large-scale designs, manual assertion definition necessitates a deep understanding of the design under verification, increasing the cost of assertion creation. To address these challenges, various studies have focused on automatic assertion generation. While these techniques have mitigated the limitations of manual assertion definition, they still exhibit some shortcomings. These include the generation of a high number of assertions by automatic assertion miners, the production of redundant and inconsistent assertions, and sometimes the lengthy execution time required by existing assertion miners for mining assertion sets. Furthermore, there remain limitations in the state-of-the-art in terms of evaluating the quality of assertions and analyzing whether they offer comprehensive design behavior coverage. These shortcomings and limitations result in high costs in the functional verification of designs under verification.

Additionally, in recent years, there have been preliminary efforts to incorporate assertions into security verification practices. While there have been notable contributions in this field, there is still a need for further study and improvement. Previously, assertions were primarily generated for functional verification purposes, focusing on verifying the functionality of designs. However, there is now a demand to enhance these functional assertions into security assertions, enabling their use in the security verification of designs.

This PhD thesis endeavors to enhance both functional verification and security verification processes for designs under verification. In pursuit of this goal, several innovative methods and solutions, many of which are introduced for the first time, are proposed. To enhance functional verification and address the limitations of existing automatic assertion miners, this thesis presents ARTmine, an automatic assertion miner that efficiently generates accurate assertion sets within a significantly reduced execution time. The assertions produced by ARTmine offer extensive coverage of design behavior, surpassing leading assertion miners in the field with minimal redundancy and inconsistency.

On the other hand, this thesis introduces two novel techniques aimed at enhancing the security verification of designs under verification. Firstly, an automatic security-based assertion miner is introduced for the first time, capable of effectively generating security assertion sets to detect inserted Hardware Trojans in the designs. Furthermore, to address security verification in the domain of autonomous vehicles, an innovative method named ADAssure is presented. ADAssure automatically generates security assertions with the objective of debugging and localizing bugs in autonomous driving

control algorithms amidst potential attacks.

Furthermore, this thesis introduces two innovative methods aimed at enhancing the current assertion evaluation techniques in the state-of-the-art and addressing the shortcomings of automatic assertion miners in producing redundant and inconsistent assertions. Firstly, a novel data mining-based method named Dominance is introduced for the first time. It not only evaluates the quality of assertions and selects those offering high design behavior coverage but also analyzes various rankings of other evaluation methods to present a unified set of high-quality assertions. Secondly, for the first time, a novel method named IMMizer is proposed to minimize the redundancy and inconsistency of assertions. IMMizer effectively detects redundant and inconsistent assertions and minimizes them, ensuring that the design behavior coverage of the minimized set equals that of the initial assertion set. While Dominance and IMMizer are examined in the context of functional verification and functional assertions in this thesis, they are readily applicable in the realm of security verification and security assertions.

In summary, this PhD thesis introduces several innovative methods and solutions, most of them for the first time, effectively enhancing the fields of functional verification and security verification. These methods include: 1) ARTmine, an automatic assertion miner; 2) an automatic security-based assertion miner; 3) ADAssure, a novel method for bug localization and debugging of autonomous driving control algorithms in autonomous vehicles; 4) Dominance, a data mining-based method for assertion evaluation; and finally, 5) IMMizer, a technique for minimizing the assertions generated by automatic assertion miners.

Kokkuvõte
Andmekaeve algoritmide kasutamine riistvarasüsteemide väidete-põhise verifitseerimise parendamiseks

Riistvara funktsionaalseks verifitseerimiseks on olemas mitmed erinevad tehnikad, sealhulgas staatilised, dünaamilised ja väidete-põhised lähenemised. Kuigi nii staatilised kui ka dünaamilised meetodid on näidanud häid tulemusi, on väidete-põhised meetodid, mis kombineerivad staatiliste ja dünaamiliste tehnikate eelised, pälvimas üha suuremat tähelepanu. Arvestades väidete-põhise lähenemise olulisust nii funktsionaalse kui ka turvalisuse verifitseerimise jaoks, on tehtud viimasel ajal olulisi jõupingutusi kõrge kvaliteediga väitekomplektide loomiseks, mis tagaksid disainide ulatusliku katvuse.

Need jõupingutused võib laias laastus jaotada kaheks: manuaalne ja automaatne väidete loomine. Manuaalne väidete loomine on aeganõudev ja ka aldis vigadele. Lisaks nõuab suurte disainide jaoks manuaalne väidete loomine detailset arusaamist verifitseeritavast disainist, suurendades sedasi väidete loomisega seotud kulusid. Eelpool mainitud probleemide lahendamiseks on hakatud arendama automaatseid väitekaevureid. Kuigi nimetatud lähenemine leevendab mõningaid manuaalse väidete loomisega seotud probleeme, siis sellele vaatamata on ka sellel meetodil omad puudused. Nende hulka kuuluvad näiteks automaatselt genereeritavate väidete suur kogus, liiaste ja vastuoluliste väidete hulk ning vahel ka genereerimisele kuluv aeg. Lisaks ei ole isegi kõige kaasaegsemad meetodid jätkuvalt väga head väidete kvaliteedi hindamisel ja nende analüüsimisel, et veenduda disaini katvuses. Seetõttu on verifitseerimisega seotud kulud aga endiselt kõrged.

Viimastel aastatel on tehtud olulisi jõupingutusi ka disainide turvalisuse verifitseerimiseks. Kuigi selles valdkonnas on märkimisväärseid arenguid, on uurimistöö vajadus veel endisel suur. Seetõttu on väidete-põhiseid lähenemisi, mis siiani olid suunatud ainult funktsionaalsuse verifitseerimiseks, edasi arendatud ka turvalisusega seotud aspektide verifitseerimiseks.

Käesoleva doktoritöö eesmärgiks on edendada nii funktsionaalse kui ka turvalisuse verifitseerimise meetodeid. Eesmärgi saavutamiseks on välja pakutud mitmeid uuenduslikke meetodeid ja lahendusi. Funktsionaalse verifitseerimise parendamiseks ja olemasolevate automaatsete väitekaevurite piirangute leevendamiseks tutvustatakse väitekaevurit ARTmine, mis genereerib täpseid väitekomplekte oluliselt kiiremini, kui senised meetodid. ARTmine'i poolt toodetud väited pakuvad ulatuslikku disaini katvust, ületades seniseid väitekaevureid väiksema liiasuse ja vastuolulisusega.

Samuti tutvustab käesolev doktoritöö kahte uut tehnikat, mille eesmärk on parendada disainide turvalisuse verifitseerimist. Esmakordselt esitletakse automaatset turvalisusele suunatud väitekaevurit, mis suudab tõhusalt genereerida verifitseerimiseks vajalikke väitekomplekte, et tuvastada disainides peituvaid Trooja hobuseid. Täiendavalt pakutakse välja meetod nimega ADAssure, et lahendada turvalisuse verifitseerimise väljakutseid autonoomsete sõidukite valdkonnas. ADAssure genereerib turvalisuse väiteid autonoomsete juhtimisalgoritmide juures nii veaotsinguks kui ka vigade lokaliseerimiseks võimalike rünnakute tingimustes.

Lisaks tutvustab käesolev doktoritöö kahte innovaatilist meetodit, millede eesmärgiks on parendada olemasolevaid väidete hindamise tehnikaid, et lahendada automaatsete väitekaevurite poolt toodetud üleliigsete ja vastuoluliste väidete probleem. Esmakordselt tutvustatakse andmekaevel põhinevat meetodit nimega Dominance. See meetod hindab väidete kvaliteeti ja valib välja need väited, mis pakuvad suurimat disaini katvust. Samuti analüüsitakse ka teiste hindamismeetodite tulemusi, et esitada ühtne kõrgkvaliteediline

väitekomplekt. Lisaks pakutakse välja meetod nimega IMMizer, mis on suunatud väidete üleliigsuse ja vastuolulisuse vähendamisele. IMMizer tuvastab ja ellimineerib üleliigsed ja vastuolulised väited, tagades samas, et minimeeritud väitekomplekti katvus vastab algse väitekomplekti omale. Kuigi Dominance'i ja IMMizerit on käesolevas doktoritöös välja pakutud funktsionaalse verifitseerimise kontekstis, siis on võimalik neid kasutada ka turvalisuse verifitseerimise valdkonnas.

Kokkuvõtvalt tutvustab käesolev doktoritöö mitmeid innovaatilisi meetodeid ja lahendusi, millest enamik on esitatud esmakordselt, ja edendab seeläbi funktsionaalse ja turvalisuse verifitseerimise valdkondi. Pakutud meetodid hõlmavad: 1) ARTmine'i, automaatset väitekaevurit; 2) automaatset turvalisusel põhinevat väitekaevurit; 3) ADAssure'i, innovaatilist meetodit autonoomsete juhtimisalgoritmide veaotsinguks ja vigade lokaliseerimiseks autonoomsetes sõidukites; 4) Dominance'i, andmekaeve meetodil põhinevat meetodit väidete hindamiseks; ja lõpuks 5) IMMizerit, meetodit, mis minimeerib automaatsete väitekaevurite poolt genereeritud väited.

Appendix 1

I

MRH. Iman, G. Jervan, and T. Ghasempouri, "ARTmine: Automatic Association Rule Mining with Temporal Behavior for Hardware Verification," *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Valencia, Spain, 2024, pp. 1-6.

ARTmine: Automatic Association Rule Mining with Temporal Behavior for Hardware Verification

Mohammad Reza Heidari Iman, Gert Jervan and Tara Ghasempouri
Department of Computer Systems, Tallinn University of Technology, Tallinn, Estonia
{mohammadreza.heidari, gert.jervan, tara.ghasempouri}@taltech.ee

***Abstract*—Association rule mining is a promising data mining approach that aims to extract correlations and frequent patterns between items in a dataset. On the other hand, in the realm of assertion-based verification, automatic assertion mining has emerged as a prominent technique. Generally, to automatically mine the assertions to be used in the verification process, we need to find the frequent patterns and correlations between variables in the simulation trace of hardware designs. Existing association rule mining methods cannot capture temporal behaviors such as *next[N], until*, and *eventually* that hold significance within the context of assertion-based verification. In this paper, a novel association rule mining algorithm specifically designed for assertion mining is introduced to overcome this limit. This algorithm powers ARTmine, an assertion miner that leverages association rule mining and temporal behavior concepts. ARTmine outperforms other approaches by generating fewer assertions, achieving broader design behavior coverage in less time, and reducing verification costs.**

***Index Terms*—verification, assertion-based verification, automatic assertion mining, data mining, association rule mining**

I. Introduction

Functional verification ensures that a system's design meets its specification before manufacturing, by identifying and rectifying design errors [1]. Among the various methods to ensure system correctness, Assertion-Based Verification (ABV) has emerged as one of the most popular solutions for checking the design functionality [2].

Assertions are Boolean expressions that define the design's behavior [1]. Traditionally, verification engineers manually defined assertions [2]. However, the manual definition of assertions is a time-consuming task and requires human expertise and a profound understanding of the design's functionality [3]. Therefore, some studies have been conducted to automatically mine the assertions [4–9].

In the literature, there are several works on the automatic assertion mining of digital designs. The work in [4] proposed an assertion miner that employs dynamic dependency graphs to extract the design's signal relations and generate assertions. Another approach presented in [5] is a syntax-guided enumeration assertion miner. The proposed methods in [6] and [7] are techniques that extract assertions using several templates in the form of Finite State Machines (FSMs). HARM [10] and its extended version [11] are hint-based assertion miners that generate Linear Temporal Logic (LTL) assertions from simulation traces. The GoldMine introduced in [8] can generate assertions for a given Register-Transfer Level (RTL) design by leveraging formal verification and static code analysis. Finally, the method described in [12] extracts assertions by transforming sequential designs into pseudo-combinational designs.

This work was supported by the Estonian Research Council grants PSG837.

While existing assertion miners show promise, they do have some drawbacks that require improvement. Assertion miners in [4–6] suffer from redundancy (*i.e.,* assertions that describe the same design behavior) and inconsistency (*i.e.,* assertions that contradict each other), requiring additional tools like IMMizer [13] for resolution. Certain miners such as [5], [10], and [11] generate excessive assertions, necessitating tools like Dominance [14] and Shayan [15] for selecting the best assertions among all the generated ones. Readability issues arise with complex antecedents in miners like the work in [4]. Moreover, high execution times in the works such as [16] incur costs in the verification process.

On the other hand, association rule mining algorithms in the realm of data mining tend to neglect the incorporation of mining temporal information [17]. The primary focus of these algorithms such as FP-growth and Eclat lies in handling static data, which remains unchanged over time [18]. Nevertheless, they typically employ various data mining metrics to generate a limited yet accurate set of association rules [18]. These algorithms continue to demonstrate efficiency in swiftly extracting valuable information from big datasets within a short timeframe [18]. Furthermore, while some research has explored temporal association rule mining [19, 20], none of these approaches can capture temporal behaviors such as *next[N], until*, and *eventually* that are vital for ABV.

This paper introduces an innovative association rule mining algorithm to address the aforementioned drawbacks. This algorithm is capable of extracting rules that incorporate the concept of time, which is necessary in the context of ABV. Based on this algorithm, an automatic assertion miner called ARTmine is proposed to generate *next[N], until*, and *eventually* temporal behaviors from hardware designs. Remarkably, the data mining literature lacks dedicated algorithms for mining these temporal behaviors [20, 21], making the algorithm presented in this paper a unique and specific solution for assertion mining.

ARTmine efficiently produces readable assertions in significantly less time compared to other approaches. It generates a compact assertion set, minimizing redundancy and inconsistency while effectively covering the design behavior. As a result, ARTmine reduces verification costs and time.

The contributions of this paper are listed as follows:

- A new association rule mining algorithm is proposed to create an assertion miner called ARTmine, which can automatically generate the most important temporal patterns in ABV, *i.e., next[N], until*, and *eventually*, in a shorter amount of time in comparison with the proposed methods in the literature;
- ARTmine integrates suitable data mining metrics (*e.g.,* min_supp, and min_conf) that aid it in generating

979-8-3503-4859-0/DATE24/© 2024 EDAA

fewer assertions, prioritizing only the most accurate and valid ones;
- ARTmine achieves broader design behavior coverage than the state-of-the-art through an efficient algorithm capable of analyzing all relationships among all hardware design variables with minimal overhead.

The paper is organized as follows: The preliminaries are presented in section II and the proposed method is elaborated in section III. Section IV presents the experimental results and finally, Section V concludes the paper.

II. Preliminaries

In this section, we briefly explain the definitions and concepts used in this paper.

Definition 1: An ***atomic proposition*** is a logic formula that does not contain logical connectives [6]. Examples of *atomic propositions* are such as a1 = True, and b1 = False.

Definition 2: A ***proposition*** is a composition of *atomic propositions* through logical connectives [6]. An example for *proposition* is a1 = True && b1 = False && c1 = True.

Definition 3: An ***assertion*** is a composition of propositions through temporal operators that must hold or must become true during the execution of the design [6]. Typically, an assertion is divided into two parts: the left side, named *antecedent*, and the right side, called *consequent* [6]. The general structure of an assertion in Property Specification Language (PSL) is like $always(antecedent \rightarrow consequent)$, which implies that the consequent will hold whenever the antecedent occurs [22].

Definition 4: A ***simulation trace*** consists of the values of the variables of hardware designs that have been stored as records of data for different time instants (clock cycles) during the execution of the designs [10].

Definition 5: ***Temporal pattern next[N]***: In PSL, *next[N]* temporal pattern will be in the form of: $always(antecedent \rightarrow next[N]\ consequent)$ [22]. This means that when antecedent occurs, after N time instant (clock cycle), consequent will occur [22]. N is an integer value and N > 0.

Definition 6: ***Temporal pattern until***: In PSL, *until* temporal pattern will be in the form of: $always(antecedent\ until\ consequent)$ [22]. This means that the antecedent is true and holds up until the time that the consequent happens [22].

Definition 7: ***Temporal pattern eventually***: In PSL, this pattern is in the form of: $always(antecedent \rightarrow eventually!\ consequent)$ [22]. This means that there exists a future time instant (clock cycle) where the consequent of assertion finally holds [22].

Definition 8: ***Frequent itemsets*** refer to a set of variables in simulation trace that occur with a frequency, indicating significant relations/associations between the variables.

Definition 9: An ***Association Rule (AR)*** is defined as an implication of the form X → Y where X, Y ⊆ I, with X ∩ Y = Ø, and I is a set of items [18, 20, 21]. X and Y are called *frequent itemsets*.

Definition 10: ***Support*** is a metric in association rule mining that indicates how frequently an itemset appears in the dataset [18]. This value is between 0 and 1. For the rule X → Y, the value of support is calculated with the following formula [18, 23]:

$$Supp(X \rightarrow Y) = P(X \cup Y) \quad (1)$$

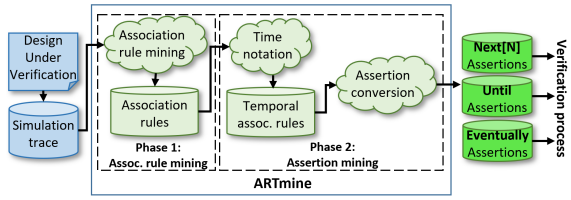


Figure 1: General flow of ARTmine

In (1), $P(X \cup Y)$ is the probability where $X \cup Y$ indicates that a record contains both X and Y, that is the union of itemsets X and Y.

Definition 11: The **min_supp** value is the threshold and a minimum value for support to decide whether an itemset is frequent (*i.e.*, occurs frequently in the simulation trace) or not [18]. If the frequency of the itemset is more than this threshold, the itemset is considered a frequent itemset [18]. A higher value of min_supp leads to generating commonly occurring (general) ARs, while a lower value of min_supp leads to generating rarely occurring ARs (corner cases) [18].

Definition 12: ***Confidence*** is an indication of how often the rule has been found to be true [24]. For the rule X → Y, this value is calculated with the following formula [18, 24]:

$$Conf(X \rightarrow Y) = P(Y|X) \quad (2)$$

It evaluates the degree of certainty of the detected association rule. This is taken to be the conditional probability $P(Y|X)$, that is the probability that a record containing X also contains Y. This value is between 0 and 1.

Definition 13: The **min_conf** is the minimum value for confidence [18]. The higher value of min_conf leads to fewer but more accurate and valid association rules [18].

III. Proposed Methodology

Fig. 1 depicts the general flow of ARTmine, taking a simulation trace (Definition 4) of the Design Under Verification (DUV) as input. The resulting output comprises a set of temporal assertions, encompassing *next[N]* (Definition 5), *until* (Definition 6), and *eventually* (Definition 7). These temporal assertions are then utilized in the verification process. As illustrated in Fig. 1, the first phase of ARTmine is *Association rule mining*. Consequently, phase 2 is *Assertion mining* which includes two sub-phases named *Time notation* and *Assertion conversion*.

During the *Association rule mining* phase, at first, the simulation trace of the hardware design undergoes preprocessing to prepare the data. Subsequently, a procedure is applied to the preprocessed data to mine all association rules (Definition 9) derived from the simulation trace.

In the second phase, *Assertion mining*, the association rules obtained from the first phase are passed to the *Time notation* step. Here, these extracted association rules are integrated with the concept of time to generate appropriate time-integrated rules (temporal association rules). These rules serve as the input for the *Assertion conversion* step. Consequently, the *Assertion conversion* step transforms the rules from the previous step into assertions, rendering them ready for utilization in the verification process. In the following subsections, each phase of the method has been discussed in more detail.

A. Association Rule Mining

The primary objective of this phase is to first preprocess the simulation trace and second, mine the association

rules from the preprocessed data. Conventional association rule mining algorithms (*e.g.,* Apriori, and FP-growth [18]) typically lack the capability to incorporate temporal considerations required for extracting crucial temporal patterns in ABV. To address this limitation and introduce a time-aware approach for association rule mining in ABV, we propose Algorithm 1. This algorithm outlines the entire procedure in the *Association rule mining* phase.

Lines 1 to 10 of the Algorithm 1 handle its initialization. 'N' denotes the time instant used for mining the *next[N]* pattern, and $\mathcal{ST}$ represents the simulation trace from which we aim to mine association rules. The minimum support threshold is min_supp (Definition 11) and the minimum confidence threshold is min_conf (Definition 13). In this algorithm, $\mathcal{FI}$ signifies a set of frequent itemsets (Definition 8). α and β represent the values of a variable at consecutive time instants in the simulation trace (*e.g.,* time instants t1 and t2). The output of the algorithm is a set of association rules ($\mathcal{R}$) (Definition 9). Subsequently, lines 11 to 14 encompass the preprocessing steps for mining association rules for the *next[N]* pattern. In Algorithm 1, lines 16 to 24 handle the preprocessing for the *until* pattern, while lines 26 to 30 accomplish the same for the *eventually* pattern. After preprocessing the $\mathcal{ST}$ for each pattern, lines 32 to 49 perform the tasks of identifying frequent itemsets in the simulation trace and then mining the association rules.

Preprocessing of Simulation Trace – next[N]:

In Algorithm 1, lines 11 to 13 demonstrate the preprocessing of the simulation trace for *next[N]* pattern. To clarify the algorithm's hypothesis, consider a rule in the form of $antecedent \rightarrow next[N]consequent$. In order to prepare the simulation trace for mining this temporal pattern, all the output of the simulation trace is moved N records above its original position. However, the inputs of the simulation trace remain as they are. The simulation trace is modified in this way since the corresponding output of input variables in a sequential hardware design may occur in the simulation trace N time instants later. This ensures correct alignment of outputs with their corresponding inputs, allowing for accurate temporal analysis and also mining patterns for different N time instants (clock cycles). In line 14, the modified simulation trace is stored for mining the *next[N]* pattern using lines 32 to 49 of the Algorithm 1.

Fig. 2 illustrates an example of preprocessing for *next[2]* pattern. The simulation trace in Fig. 2.1 is preprocessed by moving the output parts 2 time instants above their original positions, resulting in the modified simulation trace shown in Fig. 2.2. The figure uses T to represent the true value, and F to show the false value. The simulation trace consists of 5 records, divided into two categories: input variables and output variables. Each variable is assigned its corresponding value at each time instant. The last two records in Fig. 2.2 are marked as not available (NA) due to the absence of data after time instant t4 to be moved in front of these two records.

Notably, ARTmine primarily focuses on mining the essential temporal patterns such as *next[N]*, which hold significant importance in ABV. Nevertheless, the method is easily extensible to other temporal patterns like *before[N]*, due to the symmetry shared between these two patterns [25].

Preprocessing of Simulation Trace – until:

Time	Inputs			Outputs	
	v1	v2	v3	v4	v5
t0	001	01	F	11	T
t1	001	11	F	10	T
t2	101	00	T	01	T
t3	110	10	F	10	F
t4	000	01	T	11	F

(1)

moving 2 time instants

Preprocessed

Time	Inputs			Outputs	
	v1	v2	v3	v4	v5
t0	001	01	F	01	T
t1	001	11	F	10	F
t2	101	00	T	11	F
t3	110	10	F	NA	NA
t4	000	01	T	NA	NA

(2)

Figure 2: (1) Simulation trace (2) Preprocessed simulation trace

Algorithm 1 ARTmine – Association Rule Mining

1: Inputs: ▷ initialization
2: N = time instant;
3: simulation trace $\mathcal{ST}$;
4: minimum support threshold min_supp;
5: minimum confidence threshold min_conf;
6: Set of frequent itemsets $\mathcal{FI}$;
7: α = value of variable a in time t1;
8: β = value of variable a in time t2;
9: Output:
10: Set of association rules $\mathcal{R}$;
11: **case** (next[N]): ▷ preprocessing of next[N]
12: **for** all records of $\mathcal{ST}$:
13: **move** each output of $\mathcal{ST}$, N records to up;
14: **store**(moved $\mathcal{ST}$);
15: **go to** line 32;
16: **case** (until): ▷ preprocessing of until
17: $\mathcal{F} = (\beta - \alpha)/\alpha$;
18: **if** $\mathcal{F} = 0$ or $\mathcal{F} = undefined$: ▷ $\mathcal{F} = 0 \div 0 = undefined$
19: $\beta = 0$;
20: **if** $\mathcal{F} = \infty$:
21: $\beta = 0.5$;
22: **if** $\mathcal{F} = -1$:
23: $\beta = -1$;
24: **store**(modified $\mathcal{ST}$);
25: **go to** line 32;
26: **case** (eventually): ▷ preprocessing of eventually
27: **for** every input of the $\mathcal{ST}$ in time t:
28: **for** every output from time(t+1) to time(t+N):
29: **move** the output to the front of the input in time t;
30: **store**(moved $\mathcal{ST}$);
31: **go to** line 32;
32: **initialize** $\mathcal{FI}$ to be an empty set; ▷ association rule mining
33: **generate** frequent itemsets of size 1 and add them to $\mathcal{FI}$;
34: **while** $\mathcal{FI}$ is not empty do:
35: **generate** candidate itemsets Ck+1 of size k+1 by joining frequent itemsets of size k;
36: **for** each record in $\mathcal{ST}$:
37: **count** the support for each candidate itemset in Ck+1;
38: **prune** the candidate itemsets in Ck+1 that is not equal to the minimum support threshold min_supp;
39: **add** the remaining candidate itemsets to $\mathcal{FI}$;
40: **Increment** k;
41: **generate** association rules from the frequent itemsets in $\mathcal{FI}$:
42: **for** each frequent itemset X in $\mathcal{FI}$:
43: **generate** all non-empty subsets Y of X;
44: **for** each subset Y:
45: **generate** the rule Y => X-Y;
46: **calculate** the support and confidence of each rule;
47: **prune** the rules that do not meet the minimum confidence threshold min_conf;
48: **add** the remaining rules to $\mathcal{R}$;
49: **return** $\mathcal{R}$;

Lines 16 to 24 in Algorithm 1 detail the preprocessing procedure for the *until* pattern (Definition 6). To clarify the algorithm's hypothesis, consider a temporal pattern $antecedent \rightarrow until\ consequent$, where $antecedent$ comprises input variables from the simulation trace, and $consequent$ represents an output variable from the simulation trace. The preprocessing method explores the points in the simulation trace where the value of an input variable undergoes a change and subsequently identifies the corresponding output. This information is obtained through the implementation of lines 17 to 23 in Algorithm 1. In this

algorithm, α and β represent the values of a variable at two consecutive time instants (*e.g.*, t1 and t2).

The result of the equation on line 17 of the algorithm is stored in $\mathcal{F}$. Based on this value, we perform a mapping on the simulation trace as follows to explore the changes in variable values in the simulation trace according to the definition of *until* pattern:

- the first row of the simulation trace remains unchanged.
- if $\mathcal{F} = 0$ or $\mathcal{F} = undefined$, β is mapped to 0, indicating no change in the values of variables (α and β).
- if $\mathcal{F} = \infty$, β is mapped to 0.5, indicating a change from 0 (α) to 1 (β).
- if $\mathcal{F} = -1$, β is mapped to -1, indicating a change from 1 (α) to 0 (β).

The mapped values (*i.e.*, 0, 0.5, and -1) only serve as indicators for the tool to enhance the detection of changes in the simulation trace and facilitate the categorization of these distinct changes.

Although ARTmine focuses on mining the crucial temporal patterns in ABV, it can be readily extended to other patterns like *release* [25] with minor changes in its algorithm as *release* and *until* patterns share certain similarities. Specifically, the *until* pattern specifies that the consequent must hold until the antecedent becomes true, whereas the *release* pattern mandates that the consequent must persist continuously until the antecedent becomes true [25].

Preprocessing of Simulation Trace – eventually:

Algorithm 1 preprocesses the simulation trace for the *eventually* pattern in lines 26 to 31. To clarify the algorithm's hypothesis, consider a rule of the form $antecedent \rightarrow eventually!\ consequent$, where $antecedent$ comprises input variables from the simulation trace, and $consequent$ represents an output variable from the simulation trace. To prepare the concept of time for the *eventually* pattern according to Definition 7, for each row of inputs at time t of the simulation trace, all the outputs from time t+1 to t+n are moved to the front of the input at time t.

Association Rule Mining – next[N], until, eventually:

After preprocessing the simulation trace in the preceding steps to handle temporal patterns *next[N]*, *until*, and *eventually*, the resulting preprocessed simulation trace is subsequently fed into lines 32 to 49 of Algorithm 1 to mine association rules for these three patterns. This part of the algorithm is executed similarly across all pattern types.

In lines 32 to 40 of Algorithm 1, frequent itemsets ($\mathcal{FI}$) (Definition 8) of various sizes (1-itemsets, 2-itemsets, etc.) are generated iteratively until the $\mathcal{FI}$ list is empty. Specifically, the algorithm mines frequent itemsets whose support values (Definition 10) exceed the min_supp value (Definition 11), while pruning the others. In this algorithm, 1-itemsets consist of individual variables of simulation trace, 2-itemsets are pairs of variables, etc.

After mining the frequent itemsets and adding them to the $\mathcal{FI}$ list, in lines 41 to 49 of the algorithm, the association rules are extracted from the $\mathcal{FI}$ list. To clarify these lines of the algorithm, let's consider an example where $\mathcal{FI}$ is equal to 4-itemsets of {A, B, C, D}. To generate association rules from this frequent itemset, we consider all non-empty subsets of it. These subsets are 1-itemsets of {A}, {B}, {C}, {D}, 2-itemsets of {A, B}, {A, C}, {A, D}, {B, C}, {B, D}, {C, D}, as well as the 3-itemsets of {A, B, C}, {A, B, D}, {A, C, D}, {B, C, D}, and 4-itemset of {A, B, C, D}. Afterward, for each non-empty subset Y, we generate an association rule of the form Y => X-Y. By doing so, the algorithm considers all possible combinations of items in the frequent itemset to identify significant associations between different sets of items. For example, if X = {A, B, C, D} and Y = {A, B, C}, then we generate the rule {A, B, C} => {D}. The antecedent of the rule (Y) is the subset {A, B, C}, and the consequent of the rule (X-Y) is the set difference between the frequent itemset {A, B, C, D} and the antecedent {A, B, C}, which is {D}. This process is repeated for each non-empty subset of $\mathcal{FI}$, yielding a set of association rules. Finally, we evaluate each association rule for its support and confidence, pruning those that are below the min_conf threshold (Definition 13). In this part of the algorithm, ARTmine optimizes assertion generation by thoroughly inspecting non-empty subsets of itemsets, exploring diverse combinations, and selectively pruning those that are a subset of each other and also fall below the min_supp and min_conf thresholds. This process minimizes redundancy while ensuring the creation of valid and accurate assertion sets with minimal overhead.

Increasing the min_supp value results in fewer assertions that describe more general design behavior, while decreasing the min_supp value leads to assertions covering rare design behavior (corner cases). Similarly, raising the min_conf value produces fewer but more valid assertions. Valid assertions refer to assertions that will not be violated during the simulation with different scenarios. The utilization of these values in ARTmine facilitates an effective verification process. In this paper, min_supp and min_conf are set to 0.01 and 1, respectively, as we aim to discover corner cases while achieving high design behavior coverage (details are presented in Section IV).

At this point, with the completion of the association rule mining for all three patterns, these rules serve as the fundamental components of the *Assertion mining* phase.

B. Assertion Mining

This phase consists of two steps: *Time notation* and *Assertion conversion*, which are explained in the following:

1) Time Notation

In this step, the method integrates the concept of time into the *association rules* generated in the first phase, leading to a set of *temporal association rules*. Algorithm 2 covers the time notation process for all three temporal patterns, detailed in the following sections. The initial 7 lines of the algorithm handle its initialization. It involves $\mathcal{U}$ as a set of mined association rules, and $\mathcal{R}$, $\mathcal{R}'$, and $\mathcal{R}''$ as three distinct association rules.

Time Notation – next[N]:

After mining association rules in the first phase (section III-A), the method provides us a set of rules in the general form of $antecedent \rightarrow consequent$. In this step, ARTmine determines to which temporal pattern each extracted rule belongs. Subsequently, it assigns the corresponding time label to the rule using Algorithm 2.

Line 9 of the algorithm detects which rules are associated with the *next[N]* pattern. Subsequently, line 10 assigns the corresponding N to *next[N]* in the rule $antecedent \rightarrow next[N]\ consequent$. If the antecedent value matches an

Algorithm 2 ARTmine – Time Notation

1: $\mathcal{U}$ = set of mined association rules; ▷ initialization
2: $\mathcal{R}$ = $antecedent \rightarrow consequent$ in time instant t_i;
3: $\mathcal{R}'$ = $antecedent \rightarrow consequent$ in time instant t_j;
4: $\mathcal{R}''$ = $consequent \rightarrow antecedent$;
5: {$\mathcal{R}$, $\mathcal{R}'$} $\in \mathcal{U}$;
6: TID = time instant difference;
7: move_count = number of moved records in the preprocessing phase, it corresponds with N in the next[N] pattern;
8: **for** a rule $\mathcal{R}$ in $\mathcal{U}$: ▷ next[N] time notation
9: **if** ($antecedent$ == an input of preprocessed sim. trace) and (($consequent$ == an output of preprocessed sim. trace) and (move_flag == true)):
10: **label** $\mathcal{R}$ as **next**[move_count] temporal association rule;
11: **else**:
12: **discard** $\mathcal{R}$;
13: **if** (($\mathcal{F}$ = 0.5) or ($\mathcal{F}$ = -1)) and ($\mathcal{R}'' \in \mathcal{U}$): ▷ until time notation
14: **label** $\mathcal{R}$ as **until** temporal association rule;
15: **discard** $\mathcal{R}''$;
16: **for** rules in form of $\mathcal{R}$ and $\mathcal{R}'$ in $\mathcal{U}$: ▷ eventually time notation
17: **if** ($count(antecedent)$ == $count(consequent)$) and (move_flag == true):
18: **for** each corresponding $antecedent$ and $consequent$ in $\mathcal{R}$:
19: TID[$\mathcal{R}$] = time_instant[$antecedent$] - time_instant[$consequent$];
20: **for** each corresponding $antecedent$ and $consequent$ in $\mathcal{R}'$:
21: TID[$\mathcal{R}'$] = time_instant[$antecedent$] - time_instant[$consequent$];
22: **if** (TID[$\mathcal{R}$] < 0) and (TID[$\mathcal{R}'$] < 0) and (TID[$\mathcal{R}$] != TID[$\mathcal{R}'$]):
23: **label** $\mathcal{R}$ as **eventually** temporal association rule;

input in the simulation trace, and the consequent value has already been moved to another record of it ('move_flag == true'), the rule is labeled as a *next* temporal association rule. Otherwise, other mined rules are discarded.

Time Notation – until:

Lines 13 to 15 in Algorithm 2 describe the process of identifying the *until* pattern. If the value of $\mathcal{F}$ (computed in phase 1) is 0.5 or -1, and for this value, both $antecedent \rightarrow consequent$ and $consequent \rightarrow antecedent$ rules were obtained in the first phase, the rule is labeled as an *until* temporal association rule. The rule $consequent \rightarrow antecedent$ is then removed from the set of mined rules. Indeed, the essential criterion for identifying a rule as an *until* pattern is the presence of both the $antecedent \rightarrow consequent$ and $consequent \rightarrow antecedent$ rules in the set of mined rules of phase 1.

Time Notation – eventually:

Lines 16 to 23 in Algorithm 2 are for detecting the *eventually* pattern. To achieve this, ARTmine first calculates the occurrences of $antecedent$ and $consequent$ in the preprocessed simulation trace, denoted as $count(antecedent)$ and $count(consequent)$, respectively. If these two values are equal, the time instants of their occurrences are captured. Subsequently, the difference in time instants (TID in Algorithm 2) between $antecedent$ and $consequent$ is calculated. If the differences are negative and non-equivalent (line 22), the extracted association rule represents an *eventually* pattern and is labeled as an *eventually* temporal association rule (lines 23). In fact, the essential condition for identifying a rule as an *eventually* pattern is that the TIDs must be negative and non-equivalent.

2) Assertion Conversion

In this step, the mined temporal association rules are transformed into temporal assertions using the labels assigned in the *Time notation* step (section III-B1). ARTmine provides assertions in SVA syntax. However, in this section, we will focus on explaining the general format of temporal mined rules in PSL [22] as it is more understandable.

The output of the *Time notation* step for temporal association rules labeled as *next[N]* is converted to the PSL format of "*always(antecedent → next[N] consequent)*". Temporal association rules labeled as *until* are transformed into "*always(antecedent until consequent)*", and rules with the *eventually* label are changed to "*always(antecedent → eventually! consequent)*" in PSL. The generated assertion sets in this phase are now ready for the verification process.

IV. Experimental Results

ARTmine[1] has been implemented in Python and evaluated using several benchmarks developed in Verilog and SystemVerilog. The benchmarks include some of the ISCAS'89 designs from [26] and Arb2, Decoder, Multdiv, Controller, and Id_stage from the GoldMine repository in [27]. One benchmark is the Bridge that connects memory and IO. The Arbiter and LBDR benchmarks are vital components of an open-source project named NoC router Bonfire [28].

Table I: Description of injected mutants

Mutation operator types	List of operators
arithmetic operators	+, −, ×, /, %
relational operators	==, !=, >, <, >=, <=
logical operators	&&, ||
assignment operators	+=, − =, ×=, /=, %=, =
unary operators	+, −, ~, !
bitwise operators	≪, ≫, &, |, ∧
bitwise assignment operators	≪=, ≫=, & =, | =, ∧ =

Table II: Experimental results of ARTmine

Benchmarks	T_Len	Lines	I/O	#A-*N*	#A-*U*	#A-*E*	ET*N*	ET*U*	ET*E*
Arb2	100	28	6	8	0	0	0.001s	0.11s	0.05s
Id_stage	1K	813	82	1793	107	2	13s	6m11s	50s
Decoder	1K	426	4	369	20	1	0.24s	2m	38s
Controller	10K	788	57	748	72	1	12s	5m14s	37s
Multdiv	1K	559	15	348	79	4	49s	6m9s	3m12s
Arbiter	30K	245	22	105	55	3	10s	6m	3m
LBDR	80K	95	13	195	27	3	57s	7m3s	4m
Bridge	100K	196	16	79	36	1	7s	13m	7m
S27	1K	36	5	1	0	0	0.05s	20s	15s
S15850	1K	11247	227	9542	227	3	1m55s	16m42s	5m13s
S35932	1K	39481	355	1084	48	1	54s	9m11s	4m5s
S38417	1K	26190	134	1509	308	12	18s	15m	3m
S38584	1K	22734	342	5093	1460	8	1m10s	22m21s	8m14s

To evaluate the assertion sets, we have implemented an automatic mutant generator and injector following the details in [29]. We used a complete set of mutants that converted all operators and bits, injecting them into the RTL designs. Table I provides details on the injected mutants, including the types of mutated operators in the column 'Mutation operator types' and the changes made to the operators listed in each row of the 'List of operators' column. Additionally, all 0 and 1 bits have been changed to each other. Furthermore, in all the experiments for ARTmine the values for the minimum support (Definition 11) and minimum confidence (Definition 13) have been set to 0.01 and 1, respectively. These values allow the verification engineer to guide ARTmine in mining fewer yet more valid assertions, effectively considering the corner cases of designs. Moreover, N has been set to 2 for the *next[N]* pattern, but it can be adjusted to other values.

Table II shows experimental results of ARTmine, including the number of mined assertions for *next[N]*, *until*, and *eventually* patterns (columns '#A-*N*', '#A-*U*', and '#A-*E*'),

[1] https://github.com/MohammadRezaHeidariIman/ARTmine

Table III: Comparison of ARTmine with other assertion miners

Benchmarks	#Mutants	#Assertions			Mutant Detection (%)			Execution time		
		ARTmine	HARM	GoldMine	ARTmine	HARM	GoldMine	ARTmine	HARM	GoldMine
Arb2	10	8	12	9	100	100	100	0.161s	0.49s	2.4s
Id_stage	660	1902	105716	2790	90	80	45	7m14s	13m41s	18m11s
Decoder	400	390	780	418	69	53	40	2m38.24s	4m33s	4m11s
Controller	644	821	13672	1068	91	76	37	6m3s	9m17s	15m51s
Multdiv	1801	431	21620	700	96	88	54	10m10s	17m30s	13m23s
Arbiter	860	163	2017	113	100	66	68	9m10s	20m14s	26m12s
LBDR	632	225	480	100	100	39	18	12m	15m38s	16m10s
Bridge	560	116	1596	806	78	50	31	20m7s	1h15m	36m4s
S27	12	1	1	NS *	100	100	NS *	35.05s	0.56s	NS *
S15850	2642	9772	68174	NS *	80	60	NS *	23m50s	33m09s	NS *
S35932	3700	1133	159315	NS *	52	34	NS *	14m10s	38m18s	NS *
S38417	3022	1829	15889	NS *	68	40	NS *	18m18s	25m12s	NS *
S38584	2905	6561	145925	NS *	84	62	NS *	31m45s	41m37s	NS *

* GoldMine could not provide any solution (assertion) for this benchmark.

along with their corresponding execution times ('ET*N*', 'ET*U*', and 'ET*E*'). In this table, column 'T_Len' indicates the length of the simulation traces, and column 'Lines' shows the lines of code for each benchmark. Furthermore, 'I/O' is the number of I/O in each benchmark. Experimental results show that ARTmine is capable of generating a reasonable number of assertions in a suitable amount of time, even for large-scale designs like the ISCAS'89 benchmarks.

Table III presents the efficiency of ARTmine in contrast to the most popular assertion miners in the literature, *i.e.*, HARM [10] and GoldMine [8]. The column '#Mutants' represents the number of injected mutants for each benchmark. The column '#Assertions' compares the number of assertions generated by ARTmine, HARM, and GoldMine, and the column 'Mutant Detection (%)' presents the percentage of the detected mutants. To mine the assertions for HARM, and GoldMine, we ran the tools available on their repositories in [10] and [27]. In our experiments, all assertion miners used the same simulation traces and designs, and mutant injection was performed similarly for all of them.

As can be seen, ARTmine has generated significantly fewer assertions than the other tools (in almost all cases), while being considerably more effective in mutant detection. The results indicate that HARM has generated an excessive number of assertions for most benchmarks, potentially prolonging the verification process. Conversely, GoldMine cannot handle ISCAS'89 benchmarks, as indicated by 'NS' (No Solution) in Table III. Unlike other tools, GoldMine only works with Verilog designs and cannot handle simulation traces or vcd files [8]. Moreover, it is limited to mining assertions from designs in the RTL format [8], while the ISCAS'89 benchmarks are implemented in the gate-level format, making GoldMine unable to mine any assertion for them. However, we employed ISCAS'89 benchmarks to evaluate our assertion miner with large-scale designs and compare it to the latest miner in the literature, HARM [10]. Moreover, ARTmine exhibits shorter execution times compared to HARM and GoldMine.

V. Conclusion

In this paper, we proposed an association rule mining algorithm that forms the foundation of ARTmine, an automatic assertion miner that efficiently generates accurate assertion sets encompassing *next[N], until*, and *eventually* temporal patterns. ARTmine outperforms other methods by detecting more injected mutants with fewer assertions.

References

[1] M. Boulé *et al.*, *Generating Hardware Assertion Checkers: For Hardware Verification, Emulation, Post-Fabrication Debugging and On-Line Monitoring.* Springer, 2008.

[2] X. T. Ngo *et al.*, "Hardware property checker for run-time hardware trojan detection," in *2015 ECCTD*, 2015, pp. 1–4.

[3] S. Katz *et al.*, "Have I written enough properties? A method of comparison between specification and implementation," in *ACM CHARME*, 1999, pp. 280–297.

[4] J. Malburg *et al.*, "Property mining using dynamic dependency graphs," in *ASP-DAC*, Jan 2017, pp. 244–250.

[5] G. Martino *et al.*, "Syntax-guided enumeration of temporal properties," in *2019 FDL*, 2019, pp. 1–8.

[6] A. Danese *et al.*, "A-team: Automatic template-based assertion miner," in *54th DAC conf.*, 2017, pp. 1–6.

[7] R. Hariharan *et al.*, "From rtl liveness assertions to cost-effective hardware checkers," in *2018 Conference on DCIS*, 2018, pp. 1–6.

[8] S. Vasudevan *et al.*, "Goldmine: automatic assertion generation using data mining and static analysis," in *DATE*, 2010, pp. 626–629.

[9] S. Hertz *et al.*, "Mining hardware assertions with guidance from static analysis," *IEEE TCAD*, vol. 32, no. 6, pp. 952–965, 2013.

[10] S. Germiniani *et al.*, "Harm: A hint-based assertion miner," *IEEE TCAD*, vol. 41, no. 11, pp. 4277–4288, 2022.

[11] ——, "Exploiting clustering and decision-tree algorithms to mine ltl assertions containing non-boolean expressions," in *VLSI-SoC*, 2022.

[12] M. R. Heidari Iman *et al.*, "A methodology for automated mining of compact and accurate assertion sets," in *NorCAS*, 2021, pp. 1–7.

[13] ——, "Immizer: An innovative cost-effective method for minimizing assertion sets," in *DSD*, 2022, pp. 1–8.

[14] M. R. Heidari Iman *et al.*, "An automated method for mining high-quality assertion sets," *MICPRO*, vol. 97, p. 104773, 2023.

[15] T. Ghasempouri *et al.*, "On the estimation of assertion interestingness," in *VLSI-SoC*, Oct 2015, pp. 325–330.

[16] A. Danese *et al.*, "Automatic extraction of assertions from execution traces of behavioural models," in *DATE*, 2015, pp. 67–72.

[17] S. M. Ghafari *et al.*, "A survey on association rules mining using heuristics," *WIREs DM & Knowl. Disc.*, vol. 9, no. 4, p. e1307, 2019.

[18] J. Han *et al.*, "6 - mining frequent patterns, associations, and correlations: Basic concepts and methods," 2012, pp. 243–278.

[19] A. Segura-Delgado *et al.*, "Temporal association rule mining: An overview considering the time variable as an integral or implied component," *WIREs DM & Knowl. Disc.*, vol. 10, no. 4, p. e1367, 2020.

[20] C. Antunes *et al.*, "Temporal data mining: an overview," 2001.

[21] S. Bilqisth *et al.*, "Determination of temporal association rules pattern using apriori algorithm," *IJCCS Journal*, vol. 14, p. 159, 04 2020.

[22] "Standard for property specification language (psl), ieee standard," p. 1–184, 2012.

[23] M. Shahin *et al.*, "Exploring factors in a crossroad dataset using cluster-based association rule mining," 2022, the 13th conf. on ANT.

[24] M. R. H. Iman *et al.*, "Anomalous file system activity detection through temporal association rule mining," in *9th ICISSP*, 2023, pp. 733–740.

[25] C. Czepa *et al.*, "Modeling compliance specifications in linear temporal logic, event processing language and property specification patterns: a controlled experiment on understandability," *Software & Systems Modeling*, vol. 18, 12 2019.

[26] "ISCAS'89," https://sportlab.usc.edu/~msabrishami/benchmarks.html.

[27] "Goldminer," https://bitbucket.org/debjitp/goldminer/src/master/example/.

[28] "Project Bonfire Network-on-Chip," https://github.com/Project-Bonfire/Bonfire_handshake, 2017.

[29] U. Repinski *et al.*, "Combining dynamic slicing and mutation operators for esl correction," in *ETS*, 2012, pp. 1–6.

Appendix 2

II

MRH. Iman, J. Raik, M. Jenihhin, G. Jervan, and T. Ghasempouri, "An Automated Method for Mining High-Quality Assertion Sets," *Microprocessors and Microsystems Journal*, Vol. 97, pp. 104773, (2023), DOI: https://doi.org/10.1016/j.micpro.2023.104773.

Microprocessors and Microsystems 97 (2023) 104773

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

An automated method for mining high-quality assertion sets

Mohammad Reza Heidari Iman *, Jaan Raik, Maksim Jenihhin, Gert Jervan, Tara Ghasempouri

Department of Computer Systems, Tallinn University of Technology, Tallinn, Estonia

ARTICLE INFO

Keywords:
Assertion-based verification
Automatic assertion mining
Assertion qualification
Data mining

ABSTRACT

Assertion-Based Verification (ABV) is one of the promising ways of functional verification. The efficiency of ABV largely depends on the quality of the assertions in terms of how accurately they capture the consistency between implementation and specification. To this end, several assertion miners have been developed to automatically generate assertions. However, existing automatic assertion miners typically generate a huge amount of assertions which can lead to overhead in the verification process. Assertion evaluation, on the other hand, has recently appeared to evaluate and select high-quality assertions among the huge generated assertion set. These methods typically measure the quality of an assertion based on different metrics. These metrics nonetheless, consider dissimilar and distinct aspects which lead to difficulties in deciding what metric should influence more in assertion evaluation. Thereby, to exceed the state-of-the-art, a flow is proposed in which an assertion miner and an assertion evaluator are introduced. The assertion miner is capable of generating a set of readable and compact assertions. The assertion evaluator instead estimates the quality of the assertion set with a data-mining-based algorithm called *dominance*. *Dominance* is able to analyze the outcome of different metrics to unify them. Experimental results present the effectiveness of the proposed flow by comparing them to the state-of-the-art.

1. Introduction

Digital systems become more complex with each generation [1]. Therefore, verifying that their behavior is correct has become a very challenging task. To this end, functional verification aims at guaranteeing that the design of a system satisfies its specification before manufacturing, by detecting and removing design errors [2]. Among all the solutions for ensuring the robustness of the systems, Assertion-Based Verification (ABV) has emerged as one of the most popular solutions for checking the design functionality [2].

An assertion is a Boolean expression that defines the behavior of designs [2,3]. Traditionally, assertions were defined manually [4,5]. However, the manual definition of assertions needs human expertise, it is costly and error-prone [4,5]. In general, to write an assertion set, a verification engineer needs creativity and a deep understanding of the design's functionality [6,7]. Therefore, some studies have been carried out to automatically mine the assertion sets [8–16].

Several works on assertion mining for digital designs have been proposed. The work in [8] is an automatic assertion miner which generates assertions using a dynamic dependency graph. They have extracted relations between signals of the design using simulation traces. The method in [9] is another approach that uses a syntax-guided enumeration assertion miner. Studies in [10,11] are techniques that extract assertions using several templates in the form of Finite State Machines (FSMs). The GoldMine tool is presented in [12,13]. The tool automatically generates assertions for a given Register-Transfer Level (RTL) design. This tool uses simulation traces, formal verification, and static code analysis. Finally, [14–16] combine a dynamic dependency graph and FSM to achieve the strength of both techniques for assertion extraction.

However, it is notable that in most of the assertion mining approaches, the number of generated assertions is considerably high, which can lead to redundancy in assertions, (*i.e.* assertions that describe the same behavior of the design), or inconsistent assertions, (*i.e.* assertions that describe a contradictory behavior of the design in comparison with another assertion). Consequently, these issues can cause exhaustive and non-compact assertion sets.

To this end, some studies have been performed to select a set of high-quality assertions from a large number of generated assertions [17–19]. The high-quality assertions are mainly the ones that can cover more of the design's behaviors. Analyzing the design behavior coverage is performed with the aid of mutants. In this regard, the assertions that detect more mutants can cover more behavior of the design. Obviously, selecting high-quality assertions and pruning the bad ones from the verification process can significantly reduce the time

* Corresponding author.
E-mail addresses: mohammadreza.heidari@taltech.ee (M.R. Heidari Iman), jaan.raik@taltech.ee (J. Raik), maksim.jenihhin@taltech.ee (M. Jenihhin), gert.jervan@taltech.ee (G. Jervan), tara.ghasempouri@taltech.ee (T. Ghasempouri).

https://doi.org/10.1016/j.micpro.2023.104773
Received 30 April 2022; Received in revised form 18 November 2022; Accepted 12 January 2023
Available online 20 January 2023

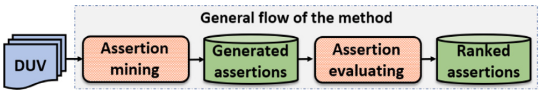


Fig. 1. General flow of the proposed method.

and cost of this process. As said, a bad assertion can typically refer to redundant assertions or inconsistent assertions.

In the context of assertion evaluation, studies have been performed to estimate the quality of assertions based on several proposed metrics. For instance, in [20], the assertions are evaluated and ranked based on two main metrics, *Importance*, and *Complexity*. The degree of *Importance* is higher in the assertions which are describing the output of the design and *Complexity* is related to the number of logical operators that are used in an assertion. In [13], the quality estimation is based on the number of propositions included in the antecedent of the assertion. In [17–19,21], mined assertions are mainly ranked according to data-mining-based metrics such as *Support* (*i.e.* their frequency of occurrences during simulation); *Correlation Coefficient* (*i.e.* correlation of the occurrence of an assertion to other assertions during simulation); and *IS* measure (*i.e.* assertions which have a low frequency of occurrence but highly correlated to other assertions), etc.

However, these assertion evaluators mainly calculate incompatible degrees of quality for one specific assertion. This is due to the fact that existing approaches include several metrics where each metric considers a particular aspect of the assertion. For instance, the approach in [18] calculates the degree of quality of each assertion based on three metrics, (*i.e.*, *Support*, *Correlation Coefficient* and *IS* measure). Nonetheless, the calculated degree of quality of each assertion according to these metrics can be different, *e.g.* assertion *A* is ranked as a high-quality assertion by *Support*, medium-quality, and low-quality by *Correlation Coefficient* and *IS* measure, respectively. Thereby, at this point, the critical concern is, which metric can dominate the other metrics for estimation of the assertion quality? Is assertion *A* considered as a high-quality assertion, medium-quality, or low-quality?

The aforementioned works are not able to determine the quality of assertions based on a unified metric to provide a unique set of evaluated assertions. This paper exceeds the state-of-the-art, first, by proposing an assertion miner which is able to generate a set of compact and readable assertions. Second, by presenting a data-mining-based algorithm called *dominance* for ranking and evaluating the mined assertions.

Fig. 1 demonstrates the overall workflow of the methodology. As can be seen, the method takes the Design Under Verification (DUV) as an input and produces two main outputs. The first output, *i.e. generated assertions* are related to the assertion miner and the second output *i.e. ranked assertions* is the outcome of the assertion evaluator.

The *assertion mining* phase works with a new algorithm for mining a compact assertion set. *Assertion evaluating* phase instead utilizes a data mining algorithm called *dominance* to evaluate and rank the assertions. *Dominance* is capable of calculating the degree of quality of each assertion in a unified way thus, filling in the gap of current approaches. Moreover, this assertion evaluator is not limited to the generated assertions by the proposed miner, in fact, it is able to evaluate the assertion's quality even if they are extracted by other miners *i.e.* third-party assertion miners.

Therefore, the contributions of this work are listed as follows:

- an innovative methodology is proposed for the automated mining of compact assertion sets that is based on a complete verification environment of exhaustive valid simulation traces;
- an assertion miner is introduced to automatically generate a human-readable assertion set from the simulation traces of hardware designs;
- a novel algorithm, called *dominance* is demonstrated, that assists verification engineers with unifying all different assertion qualification metrics, to provide a unique and integrated assertion set;
- the proposed *dominance* algorithm is able to select a set of high-quality assertions among a huge number of generated assertions. This leads to a reduction in the cost and time of the verification process;
- the proposed assertion miner yields more accurate assertion sets than existing approaches by achieving 100% mutant coverage for the considered benchmarks;
- an added external feature is integrated into the methodology such that it can communicate with users to provide a localized assertion set based on the user's need for further design analysis and debugging.

It should be noted that the paper is the extension of the work described in [22], and in this paper, we propose a new algorithm – *dominance* – for assertion evaluation.

The remainder of the paper is organized as follows: The related preliminary concepts are presented in Section 2. The proposed methodology is discussed in Section 3. Section 4 presents the experimental results and comparisons of the proposed method with the state-of-the-art. Finally, Section 5 concludes the paper.

2. Preliminaries

In this section, the definitions and concepts which are deployed in this article are briefly explained to assist the readers in better understating the method.

Definition 1. An ***atomic proposition*** is a logic formula that does not contain logical connectives [10]. A ***proposition*** is a composition of *atomic propositions* through logical connectives [10]. For example, a1 = True, b1 = False, and c1 = True are *atomic propositions*. Moreover, a1 = True && b1 = False && c1 = True is an example of *proposition*.

Definition 2. An ***assertion*** is a composition of propositions through some temporal operators. An *assertion* is typically divided into two parts, the left side of an implication which is called *antecedent*, and the right part of the implication which is called *consequent* [10].

An example of assertion in Linear Temporal Logic (LTL) [23] can be as '*always (antecedent1 |-> consequent1)*', which states whenever antecedent1 happens one clock cycle later consequent1 occurs.

Definition 3. Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items and $D = \{d_1, d_2, \ldots, d_m\}$ be a dataset, *i.e.*, a set of observations, called transactions, with respect the set of items *I*. Each element in *D* contains a subset of the items in *I*. An ***association rule*** is defined as an implication of the form $X \rightarrow Y$ where $X, Y \subseteq I$ and $X \cap Y = \varnothing$ [24]. X and Y are called itemsets.

As can be seen in Definitions 2 and 3, the structure of an assertion is similar to the structure of an association rule. In fact, the antecedent and consequent of an assertion in Definition 2 is the same as X and Y, respectively, in Definition 3.

3. Proposed methodology

In this section, we demonstrate the proposed methodology for automated assertion mining, as well as automated assertion evaluation.

Fig. 2 represents the main steps of the method. Regarding the assertion mining flow, two pre-processing steps are needed *i.e.*, *design extraction* and *environment generation*. In fact, these steps provide a suitable environment for *assertion mining* step. *Assertion evaluating* step instead gets the generated assertions by the proposed miner or assertion

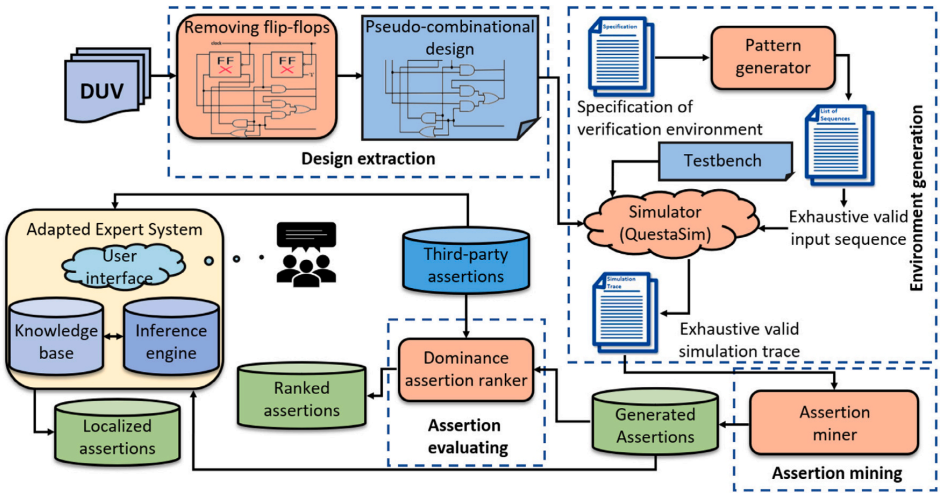


Fig. 2. Flow of the proposed methodology.

sets from the third-parties as inputs to evaluate the quality of the assertion set. Moreover, the figure shows an external feature that uses an adapted expert system [25,26] to communicate with verification engineers to provide a localized set of assertions based on their needs. The detail of each step is described in the following:

(A) *Design extraction*: This step deals with converting sequential designs to pseudo-combinational ones by removing flip-flops in order to enable a complete analysis of the assertion space within a single clock cycle. This step is performed for the immediate error detection, within a single clock cycle, thus preventing perturbing the errors in the system.

(B) *Environment generation*: In this step, with analyzing specifications of design under verification, a pattern generator is led to extract an exhaustive valid input sequence. Thereby, it prevents the flow to deal with the exhaustive input sequences, as well as with any invalid input combinations. This step is an essential part of the flow since generating an assertion set from the exhaustive valid input sequences provides a complete verification environment for the *assertion mining* step.

(C) *Assertion mining*: This step is in charge of mining the assertion set. An algorithm is presented that mines the assertion set directly from the output of the previous step.

(D) *Assertion evaluating*: This step is able to estimate the quality of assertions provided by the proposed assertion miner or by any other assertion miners in the state-of-the-art. The proposed algorithm, called *dominance*, ranks the assertions by integrating all the available assertion quality metrics in the literature to provide a unique integrated ranking framework for the assertion set.

Moreover, an adapted expert system has been used to communicate with users to provide a localized assertion set based on the user's requests. In the following, the description of these components is detailed.

3.1. Design extraction

In order to prepare the suitable inputs for the proposed assertion miner, the pseudo-combinational designs are derived-out from the original sequential designs. This is performed by removing the flip-flops and converting them to pseudo primary inputs and pseudo primary outputs [27]. *Design extraction* step in Fig. 2 represents this flow.

In the following, in the illustrative case study, the application of this conversion is presented. As a case study, we have considered an open-source 5-port NoC router Bonfire [27]. A high-level overview of this router is illustrated in Fig. 3.1. As shown in this figure, the router consists of an input buffer in the form of First-In-First-Out (FIFO), LBDR, Arbiter, crossbar, and an output buffer.

The router has 5 input/output ports, four of which (North — N, East — E, South — S, West — W) are connected to each cardinal direction. The last port (Local (L)) is connected to the local processing element. Besides, in the router, packets are sent in the form of flits, and each flit is composed of header flit, and tail flit, as well as body flit(s).

LBDR and Arbiter (demonstrated in a gray box in Fig. 3.1) are the control parts of the router and the rest are considered as the datapath. The control part controls all the main procedures of the router. Generally, the control part of any design is considered as the critical and the hard-to verify elements since the program flow of the design is maintained by it. For this reason, LBDR and Arbiter are specifically targeted for the proposed workflow.

Fig. 3.2 and 3.3 demonstrate these components after conversion to pseudo-combinational version. As shown here, this is performed by removing the flip-flops and converting their outputs and inputs to pseudo primary inputs and pseudo primary outputs, respectively. This allows deriving of assertions corresponding to a form A -> next C, where A is the antecedent and C is the consequent of the assertion.

In the next step, it is described how for the above components, a set of exhaustive valid simulation traces are generated to derive the complete verification environment for the assertion miner.

3.2. Environment generation

In this step, a pattern generator is developed to extract a set of exhaustive valid input sequences. Therefore, the specifications of the verification environment of LBDR and Arbiter, more specifically, ELBDR and SArbiter are studied, because they have one of the most connected signals. These specifications guide the pattern generator to extract the set of exhaustive valid input sequences. These sequences are then fed to the simulator to produce a set of exhaustive valid simulation traces. *Environment generation* step in Fig. 2 demonstrates this flow.

As shown in Fig. 3.2 and Fig. 3.3, ELBDR's output port signals are the N, W, S, and L signals, while for SArbiter, request and grant signals exist for the N, E, W, and L. In ELBDR, 3 bits are considered for the flit_id (FLIT_TYPE), and 4 bits for the destination address (DEST_ADDR). Moreover, an Eempty bit that is coming from the East input buffer is considered for ELBDR. The other inputs of ELBDR (iN, iW, iS, and iL) are one bit. For SArbiter, there is one bit input for any

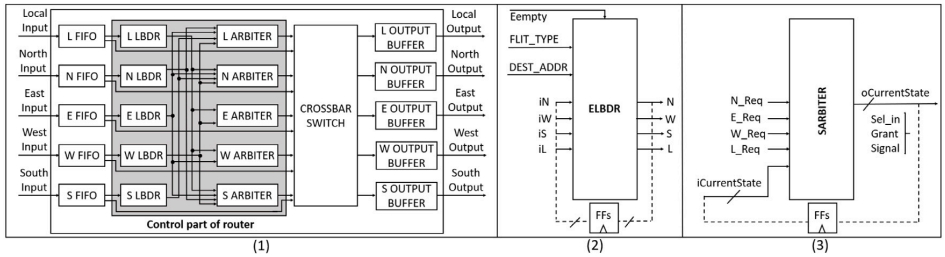


Fig. 3. (1) High-level overview of the router (2) ELBDR pseudo-combinational design (3) SArbiter pseudo-combinational design.

of the N_Req, E_Req, W_Req, L_Req, as well as 5 bits for any of the iCurrentState and oCurrentState.

Based on the specifications, two different conditions should be considered for ELBDR to lead the pattern generator for generating the exhaustive valid input sequences:

- if Eempty is equal to 1 (East input is empty):
  - the input ports, (*i.e.* iN, iW, iS and iL) can have any values,
  - flit_id (FLIT_TYPE in Fig. 3.2) can have any value,
  - destination address (DEST_ADDR) can have any value.
- if Eempty is equal to 0 (East input is not empty):
  - iN, iW, iS and iL must be one-hot (*i.e.* at the same time only one of the inputs could be 1),
  - flit_id (FLIT_TYPE) is one-hot (*e.g.* Header = 001, Body = 010, Tail = 100),
  - the outputs of ELBDR will always be one-hot (assuming that the routing algorithm is XY and we can only go from East to North, South, West, or Local. And they cannot be active at the same time).

Initially, the number of sequences for the exhaustive simulation trace was equivalent to $2^{12} = 4096$ for ELBDR due to the fact that the number of input bits is equal to 12. After feeding the pattern generator based on the specifications, the number of sequences for the simulation trace is reduced to 2144. As a result, the invalid conditions from the exhaustive input sequences are removed and filtered out.

Similarly, for SArbiter, the specifications are studied to guide the pattern generator. The specification is as follows:

- only one input can be granted at the same time. Hence, the grant vector (iCurrentState) state is defined as one-hot,
- N_Req, E_Req, W_Req, and L_Req can have any values (because SArbiter can get requests from different inputs except the South port).

Initially, the number of sequences for the exhaustive simulation trace was equivalent to $2^9 = 512$ for SArbiter due to the fact that the number of input bits is equivalent to 9. After guiding the pattern generator according to the specification, the number of sequences for the simulation trace is reduced to 80.

It is notable that this section adapted the idea of producing exhaustive valid input sequences from the work in [27]. However, the proposed methodology markedly improved the work in [27], as it developed an automatic tool to generate the valid exhaustive sequences based on the specifications. On the contrary, the work in [27], manually applies a set of filters on the exhaustive input sequence to gain the valid one. Thereby, the proposed method considerably reduced the time and cost in terms of memory requirement in comparison with [27].

The generation of exhaustive valid simulation trace is significantly pivotal since it prevents the assertion miner from being in non-realistic conditions and only the functionally of feasible values are retained. Thereby, the miner generates a complete assertion set that is based on valid scenarios. In the following, the assertion miner's algorithm is detailed.

3.3. Assertion mining

As demonstrated in *assertion mining* step of Fig. 2, the assertion miner generates a set of assertions from the output of the previous step, *i.e.* exhaustive valid simulation trace. This step is one of the fundamental components of the proposed methodology since it is in charge of discovering the relations between the primary inputs and primary outputs of the design. The assertions are generated according to the proposed Algorithm 1. As it can be seen, the algorithm is divided into three phases.

Phase 1 starts by reading the sequences of simulation trace to discover the similarities among them. Note that N in the algorithm indicates the number of sequences in the simulation trace file. Besides, input() and output() methods show the input and output parts of a sequence. $length()$ method indicates the length of a sequence. In this phase, all the sequences are compared. For every comparison, each bit in a sequence is compared with its corresponding bit in the next sequence. Lines 5 to 8 of Algorithm 1 describe how this comparison is performed. If both bits are equal, the value of the bit is kept. Otherwise, it is considered as a 'don't care' status and the result is stored in the variable 'result1'. Note that the comparisons in which all the input or output bits are determined as a 'don't care' are not analyzed in the next phase. Lines 11 and 12 show the input and output parts of a sequence.

Phase 2 aims at finding similarities among the outcomes of phase 1. At first, the input parts of all the results of phase 1 are compared (looking at Fig. 4 to distinguish the input part and output part of a sequence). If the bits in the input part are equivalent, the output part of the sequences is checked.

To achieve this, similar to phase 1, each bit is compared with its corresponding bit in the other sequence. If they are equal, the value of the bit is stored in the variable 'result2'. Otherwise, 'don't care' status is considered. On the contrary, according to lines 29 and 30 of Algorithm 1, if a sequence is not equal to any other sequences, it will go to the next phase of the algorithm.

Note that in both phase 1 and 2, there might be some results where their input/output parts all are 'don't care'. In such cases, those results should be discarded. This is described in lines 13, 14, 35, and 36 of the algorithm.

Finally, phase 3 decides which logic operator should be inserted among the variables of Result2. The mechanism is as follows: A logical AND is inserted between the bits of the input part of a Result2. Moreover, it checks if a bit in the output part is equal to its corresponding bit in another generated Result2. In this condition, an OR will be added between those cases.

Fig. 4 demonstrates several examples from C17 [28] design for each phase of algorithm 1. As shown in this figure, phase 1 starts by reading and comparing all the sequences of the simulation trace. For a better

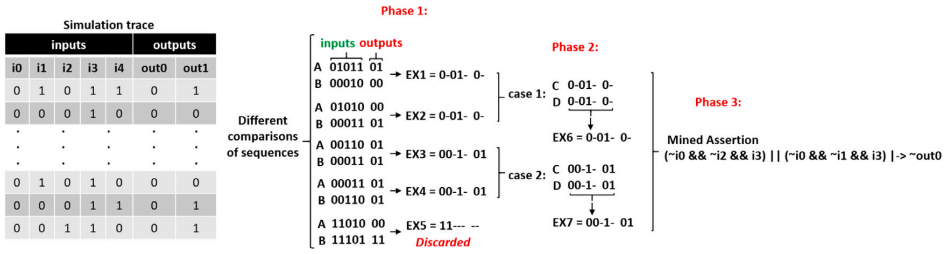


Fig. 4. Three phases of the assertion miner's algorithm.

Algorithm 1 Assertion Miner

```
1: index = 0                                                    ▷ Phase 1
2: for i in 1 to N-1 do
3:     for j in i+1 to N do
4:         for k in 1 to length(seq) do
5:             if (seq[i,k] == seq[j,k]) then
6:                 Result1[index,k] = seq[i,k]
7:             else
8:                 Result1[index,k] = 'don't care'
9:         index = index + 1
10: for i in 1 to length(Result1) do
11:     in = input(Result1[i])
12:     out = output(Result1[i])
13:     if (in.allBits[i] == 'don't care') or (out.allBits[i] == 'don't care') then
14:         discard(Result1[i])
15: index = 0                                                   ▷ Phase 2
16: for i in 1 to length(Result1)-1 do
17:     cnt = 0
18:     for j in i+1 to length(Result1) do
19:         if (input(Result1[i]) == input(Result1[j])) then
20:             input(Result2[index]) = input(Result1[i])
21:             for k in 1 to length(output(Result1[i])) do
22:                 if (output(Result1[i])[k] == output(Result1[j])[k] then
23:                     output(Result2[index])[k] = output(Result1[i][k])
24:                 else
25:                     output(Result2[index])[k] = 'don't care'
26:             index = index + 1
27:         else
28:             cnt = cnt + 1
29:             if (cnt == length(Result1)-i)) then
30:                 Result2[index] = Result1[i]
31:                 index = index + 1
32: for i in 1 to length(Result2) do
33:     in = input(Result2[i])
34:     out = output(Result2[i])
35:     if (in.allBits[i] == 'don't care') or (out.allBits[i] == 'don't care') then
36:         discard(Result2[i])
37: temp = ' '                                                  ▷ Phase 3
38: index = 0
39: for i in 1 to length(Result2)-1 do
40:     temp = AND(temp, input(Result2[i]))
41:     for j in i+1 to length(Result2) do
42:         for k in 1 to length(output(Result2[i])) do
43:             if (output(Result2[i])[k] == output(Result2[j])[k] then
44:                 temp = OR(temp, AND(input(Result2[j])))
45:     Result3[index] = temp
46:     index = index + 1
```

illustration, five different examples are shown for this phase. Each pair of sequences is shown by A and B.

After applying the first phase of Algorithm 1, the EX1, EX2, EX3, EX4 and EX5 are calculated which are equal to 0-01- 0-, 0-01- 0-, 00–1 01, 00–1 01 and, 11— –, respectively. In these examples, the first part of bits (*e.g.,* 0-01- in EX1) indicates the input bits, and the second part (*e.g.,* 0- in EX1), indicates the output bits. Moreover, each bit represents a variable name of the simulation trace. For instance, in 0-01- 0-, the first bit represents the i0 variable, the second one represents the i1 variable, and finally 0- represent out0 and out1 variables, respectively.

In these examples, EX1 and EX2 create Case 1, and EX3 and EX4 create Case 2. Subsequently, Case 1 and Case 2 go to phase 2 since each pair has the same inputs. On the contrary, EX5 will be discarded since its output is equal to '-', *i.e.*, 'don't care'.

As said and as shown in Fig. 4, Case 1 and Case 2 are entered into phase 2. Since the input of both cases is equivalent, their outputs should be analyzed according to the algorithm. As a result, EX6 and EX7 are calculated.

Finally, in phase 3 the mined assertions have been generated. Going back to the illustrative example in Fig. 4, phase 3 combines EX6 and EX7 on the basis of finding similar output bits. Since the first bit in the output part of both EX6 and EX7 are equal to 0, their input bits have been ORed with each other *i.e.*, $0-01- \;||\; 00-1-$. To this end, with looking at the name of the variables in the simulation trace it is clear that the extracted assertion is equal to $(\sim i0 \&\& \sim i2 \&\& i3) \;||\; (\sim i0 \&\& \sim i1 \&\& i3) |-> \sim out0$.

It should be noted that the proposed algorithm uses 'don't care' like some tools such as ABC [29]. However, tools like ABC are able to only minimize the logic, and how much the minimized logic can cover the same behavior as the original logic is not an important issue for these tools. For instance, according to our analysis, ABC cannot consider all the corner cases of the design behavior.

3.4. Assertion evaluating

This step is performed to evaluate the quality of assertion sets based on an algorithm called *dominance*, which was defined in the concept of data mining in [30,31]. Note that this algorithm for the first time is adopted and used in the assertion evaluation context by the proposed method.

According to Fig. 2, the input of this step is an assertion set, which can be provided by the proposed assertion miner or third-parties. The output of this phase is the evaluation of the quality of assertion sets.

In the literature, there exist approaches for evaluating the quality of assertions. However, each method considers different metrics for such an evaluation. For instance, some studies show that if the occurrence of an assertion during simulation is higher than the others, it typically covers the design's behavior more. Consequently, if an error takes place in the design, this assertion is more probable to detect the error [17–19,21]. Another work assumes that if an assertion has more number of propositions (*Definition* 1) than another one, it is more likely to be a higher quality assertion thus, it should be ranked higher [13].

Nevertheless, applying any of these different metrics for assertion evaluation generally leads to different results. For instance, assertion *A* is determined as a higher quality assertion than assertion *B* according

Table 1
Relation between the rules and measures (Ω).

Rules	Support	CC	IS
r1: a→d	0.20	0.67	0.02
r2: b→c	0.10	0.50	0.00
r3: c→a	0.10	0.50	0.02
r4: a→b	0.20	0.40	0.10
r5: b→d	0.20	0.33	0.02
r6: d→c	0.20	0.33	0.10
r7: c→b	0.10	0.20	0.01
r8: b→a	0.10	0.17	0.02

to the metric in [17–19,21] however, this result is vice versa according to the metric in [13]. The reason is that any of these metrics mainly considers one specific aspect of the quality of assertions. Thereby, verification engineers deal with the issue to select high-quality assertions for the verification process.

The *dominance* algorithm aims at filling this gap and provides a unique ranking for an assertion. In the following, we have formulated the details of this algorithm.

3.4.1. Dominance algorithm

We have considered any of the assertions in an assertion set as an *association rule* (*Definition* 3) and $\mathcal{R}$ is the set of all mined rules: $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$.

Any of these rules are evaluated according to a set $\mathcal{M}$ of different measures such as *Support, CC, IS* and etc. Therefore, $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$. $\Omega = (\mathcal{R}, \mathcal{M})$ creates a table (Table 1) that consists of the rules in $\mathcal{R}$ and measures in $\mathcal{M}$. Moreover, the value of the measure m for the rule r has been denoted by $r[m]$ such that $r \in \mathcal{R}$ and $m \in \mathcal{M}$.

As discussed before, the evaluation of each measure is different from the others and this can lead to different rankings subsequently, different quality evaluations for the rules. For example, according to the *Support* measure, r_1, r_4, r_5, and r_6 are the best rules and according to the CC r_1, and regarding the *IS* measure, r_4, and r_6 are the best rules. To resolve this issue with *dominance*, we define it in two levels, which are *value dominance* and *rule dominance*:

Definition 1. value dominance: Considering two values of a measure m from the set $\mathcal{M}$ corresponding to two rules r and r' from the set $\mathcal{R}$, we say that $r[m]$ dominates $r'[m]$, denoted by $r[m] \succeq r'[m]$, iff $r[m]$ is preferred to $r'[m]$. if $r[m] \succeq r'[m]$ and $r[m] \neq r'[m]$ then, $r[m]$ strictly dominates $r'[m]$, denoted by $r[m] \succ r'[m]$. Therefore, as a result, the dominated values will be kept and the others will be removed.

Definition 2. rule dominance: Considering two rules r and $r' \in \mathcal{R}$, we define the *dominance* relationship according to the set $\mathcal{M}$ of measures as follows:

- r dominates r', denoted by $r \succeq r'$, iff $r[m] \succeq r'[m]$, $\forall\, m \in \mathcal{M}$.
- If $r \succeq r'$ and $r' \succeq r$, *i.e.*, $r[m] = r'[m]$, $\forall\, m \in \mathcal{M}$ then r and r' are *equivalent*, denoted by $r \equiv r'$.
- If $r \succeq r'$ and $\exists\, m \in \mathcal{M}$ so that $r'[m] \succ r[m]$, then r' is *strictly dominated* by r, denoted by $r \succ r'$.

Moreover, the strict *dominance* relationship fulfills the following properties:

- **irreflexive**: $r \not\succ r$, *i.e.*, $r \succ r$ is false for each $m \in \mathcal{M}$,
- **transitive**: $\forall\, r, r'$ and$r'' \in \mathcal{R}$, if $r \succeq r'$ and $r' \succeq r''$ then $r \succeq r''$.

As a result of *rule dominance*, the dominated rules will be kept and the other rules will be removed. In other words, according to *dominance* algorithm, if a rule r dominates a rule r', this means that r is equal to or better than r' for all measures in the set $\mathcal{M}$.

As an illustrative example, looking at Table 1, the rule r_3 strictly dominates r_2, because $r_3[Support] \succeq r_2[Support]$, $r_3[CC] \succeq r_2[CC]$, and $r_3[IS] \succ r_2[IS]$. Thereby, *dominance* keeps the rule r_3 and removes the rule r_2.

Table 2
Dominance results.

Rules	Support	CC	IS
r1: a→d	0.20	0.67	0.02
r4: a→b	0.20	0.40	0.10

By applying the *dominance* algorithm on all the rules in Table 1, the result will be equal to Table 2. This means no rule in $\mathcal{R}$ dominating r_1 or r_4. Hence, the aforementioned rules are considered as high-quality rules to be kept.

As explained in Section 2, the structures of an assertion (Definition 2) and an association rule (Definition 3) are similar. Thus, the *dominance* is adaptable in the context of assertion evaluation. To this end, *dominance* is capable of assisting verification engineers with comparing assertions based on the value of each metric to evaluate them in a unified manner. Section 4 demonstrates the efficiency of the *dominance* algorithm for evaluating the quality of assertions.

3.5. Adapted expert system

In this phase, a feature is presented that provides the methodology with the ability to communicate with the users. In this way, verification engineers can request a specific set of assertions related to a specific module or variable under analysis that meets their needs and interests. The difference between this feature and the *dominance* algorithm is that *dominance* can integrate other rankings metrics and provides a unique one based on its algorithm, however, the adapted feature is able to localize the assertions according to the users' needs.

The proposed feature adapts an expert system [25] for the purpose of this work. In general, any expert system has two main components which are a knowledge base and an inference engine [26]. The former consists of *Facts* and the latter is based on a set of *Expert-System-Rules*. With the help of the *Facts* stored in the knowledge base and *Expert-System-Rules* in the inference engine, the expert system provides a selective set of assertions based on the users' needs.

The details about the *Facts* and *Expert-System-Rules* are provided in the following. As the proposed expert system has been implemented in CLIPS language, (which is a specific tool and language for implementing expert systems) the descriptions and the Listings in the following subsections have been written in CLIPS syntax.

3.5.1. Facts and expert-system-rules in the adapted expert system

Definition 3. ***FACT***s, are a formal form of describing knowledge and information required to solve a specific problem. Generally, these *Facts* are gathered from the knowledge of the experts in the domain related to that specific problem. For instance, in the real world, *headache* and *fever* are two symptoms of flu that are reported by the experts in the domain *i.e.*, medical doctors. Thereby, to prepare the knowledge for an expert system, *headache* and *fever* are stored as two *Facts* for the detection of flu. We have adapted these concepts for the proposed flow.

In the proposed adapted expert system, these *Facts* are mainly extracted from the left part of the implication of each mined assertion. As an illustrative example, we can consider the generated assertion in phase 3 of Fig. 4.

As can be seen, the left part of the implication for this assertion is equivalent to $(\sim i0\&\& \sim i2\&\&i3) \parallel (\sim i0\&\& \sim i1\&\&i3)$. Correspondingly, the *Facts* related to this assertion has been presented in Listing 1.

In this listing, each *Fact* is written in CLIPS syntax and represented by i[n] which demonstrates the name of the variable i, and its corresponding bit "n". For instance, f1 means that the value for input variable i[0] is equal to 0. Similarly, for the other *Facts*, f2, f3, and f4 the variables and their values are i[2] = 0, i[3] = 1 and i[1] = 0, respectively.

Listing 1: Example of a Fact

```
(deffacts minedFacts
   (f1 !i[0]) (f2 !i[2]) (f3 i[3]) (f4 !i[1]))
```

Definition 4. ***Expert-System-RULE,*** the structure of an *Expert-System-Rule* is similar to the $IF-THEN$ statements. If the first part of an *Expert-System-Rule* (IF), matches with one or more stored *Facts* in the knowledge base of the expert system then the second part of the *Expert-System-Rule* ($THEN$) will be fired and the *Expert-System-Rule* will produce the result. For example, in the real world, IF the symptoms in a person are *headache* and *fever*, $THEN$ we can realize that the person has gotten flu. Accordingly, this concept has been adapted for the proposed flow.

Listing 2 demonstrates the *Expert-System-Rule* related to the assertion in phase 3 of Fig. 4 in CLIPS syntax. This Listing represents that IF (i[0] = 0 && i[2] = 0 && i[3] = 1) || (i[0] = 0 && i[1] = 0 && i[3] = 1) $THEN$ *out*0 is always 0. Thus, if the users are interested in selecting the assertions related to *out*0, this *Expert-System-Rule* causes the selection of the mentioned assertion.

Listing 2: Example of an Expert-System-Rule

```
(or(and(i[0]=0)(i[2]=0)(i[3]=1))(and(i[0]=0)(i
    [1]=0)(i[3]=1)) => ~out0)
```

The above described the basic components of an expert system and the process it is adapted for purpose of this work. In the following, we demonstrate further usage of this feature. According to Fig. 2, the adapted expert system can be fed by the assertions from the proposed assertion miner, as well as assertions provided by third-parties for a specific design. This can enhance the methodology with two advantages.

Firstly, in cases that the generated assertions by an assertion miner cannot completely cover all the functionality of the design, the adapted expert system can learn from what assertions have been stored by the third-parties in its knowledge base and generate a new set of assertions to increase the coverage. Secondly, the adapted expert system can reduce the redundancy of the assertion sets after the selection phase. Formula (1), formula (2), and formula (3) represent the reduction mechanism. As can be seen here, $m_1, m_2, \ldots, m_n$ is repeated in both formula (1) and formula (2) which is omitted in formula (3). With this straightforward mechanism, this feature can prevent this kind of redundancy.

$$(a_1 \;\&\&\; \ldots \;\&\&\; a_n) \;\|\; \ldots \;\|$$
$$(m_1 \;\&\&\; \ldots \;\&\&\; m_n) \;|->\; \delta, \quad n \geqslant 0 \tag{1}$$

$$(m_1 \;\&\&\; \ldots \;\&\&\; m_n) \;\|\; \ldots \;\|$$
$$(z_1 \;\&\&\; \ldots \;\&\&\; z_n) \;|->\; \delta, \quad n \geqslant 0 \tag{2}$$

$$(1),(2) => ((a_1 \;\&\&\; \ldots \;\&\&\; a_n) \;\|\; \ldots \;\|$$
$$(m_1 \;\&\&\; \ldots \;\&\&\; m_n) \;\|\; \ldots$$
$$\|\; (z_1 \;\&\&\; \ldots \;\&\&\; z_n)) \;|->\; \delta, \quad n \geqslant 0 \tag{3}$$

3.5.2. User interface in the adapted expert system

Users can interact with the system through the user interface. A user interface generally asks questions from the users to understand their needs. To this extent, the expert system is able to select an assertion set for a specific module or an output variable. For instance, the *Expert-System-Rule* in Listing 2 is related to the case that the user's desire is generating assertion for ∼ *out*0 output.

Table 3
Comparison between the approaches in [16] and the proposed method.

Benchmarks	Features	Approach 1	Approach 2	Proposed approach
SArbiter	#mutants	73	73	73
	#assertions	449	689	10
	%detected mutants	97%	99%	100%
	execution time	2 h 5 m	2 h 22 m	20 s .65 ms
ELBDR	#mutants	76	76	76
	#assertions	37	36	8
	%detected mutants	100%	100%	100%
	execution time	20 m	19 m	13 m

4. Experimental results

The efficiency of the proposed methodology has been studied on the mentioned NoC router Bonfire in Section 3.1, and its two main components, *i.e.*, LBDR, and Arbiter, as well as the crossbar switch of this router [32]. The programming languages that have been used in this work are SystemVerilog, Python, and CLIPS. The benchmarks are developed in SystemVerilog. Assertion miner and *dominance* algorithm are implemented in Python. Finally, CLIPS is the suitable language for the expert system.

Furthermore, with the aid of the adapted expert system, the generated assertions can be provided in two different syntaxes which are SystemVerilog Assertions (SVA), or Property Specification Language (PSL). This can be helpful in analyzing the designs which are programmed in any of these languages.

Table 3 presents the experimental results regarding the *assertion mining* step. Column 'Benchmarks' represents the name of benchmarks, *i.e.* Arbiter and LBDR. Column 'Features' contains four different features *i.e.* '#mutants', '#assertions', '%detected mutants', and 'execution time', which report the number of injected mutants, the number of generated assertions, the number of detected mutant, and the execution time, respectively. Columns 'Approach 1' and 'Approach 2' show the results for two different assertion miners represented in [16]. Finally, 'Proposed methodology' refers to the approach of this work.

As shown here, the proposed methodology has advantages, since it generates a considerably smaller number of assertions *i.e.*, 10 and 8 for Arbiter and LBDR, respectively. In contrast, these numbers for Approach 1 are 449 and 37, and for Approach 2 are 689 and 36. The mutant coverage is slightly increased for the proposed methodology in comparison with the other approaches for Arbiter, and remained steady in LBDR. Moreover, the execution time of the proposed assertion miner has drastically reduced in comparison with Approach 1 and Approach 2. As can be seen here, it reached 20.65s and 13 m for Arbiter and LBDR, respectively. However, these numbers are 2h5 m and 20 m for Approach 1 and, 2h22 m and 19 m for Approach 2.

Regarding mutant analysis, a mutant injector tool according to the details provided in [33] has been implemented. Note that this mutant injector is complete in sense of converting all different types of operators to create mutants. In Table 4, the details about these injected mutants in the benchmarks have been presented. The column 'Mutation operator types' demonstrates different types of operators that have been modified by the mutants. Any of the operators in each row of the column 'List of operators', has been changed to the others. In addition to these mutants, we randomly converted all 0 bits to 1 and vice versa.

As for Arbiter, the number of injected mutants is equal to 73. Note that these mutants are injected according to the details provided in Section 3.4. The number of generated assertions are 449, 689, and 10 for the three approaches, respectively. As for fault coverage, the results in percentage are 97, 99, and 100, respectively. Finally, the execution time for the three approaches is as follows 2h5 m, 2h22 m, and 20s65 ms.

For LBDR, the number of injected mutants is equal to 76. The numbers of generated assertions are 37, 36, and 8 for the three approaches, respectively. Moreover, for fault coverage, the results for all

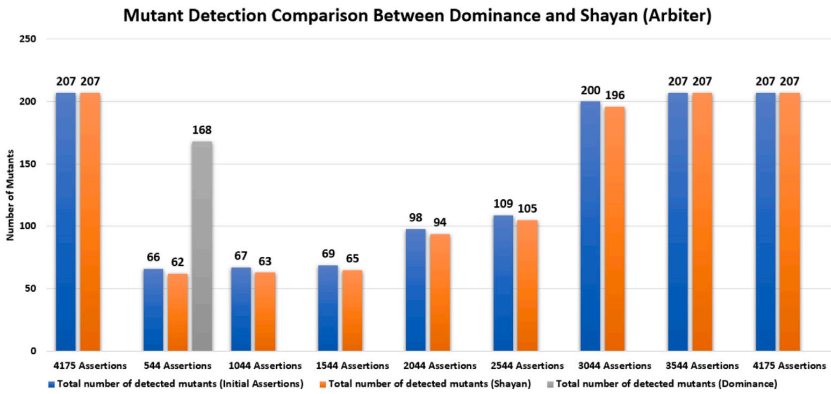


Fig. 5. Comparison between Dominance and Shayan (Arbiter).

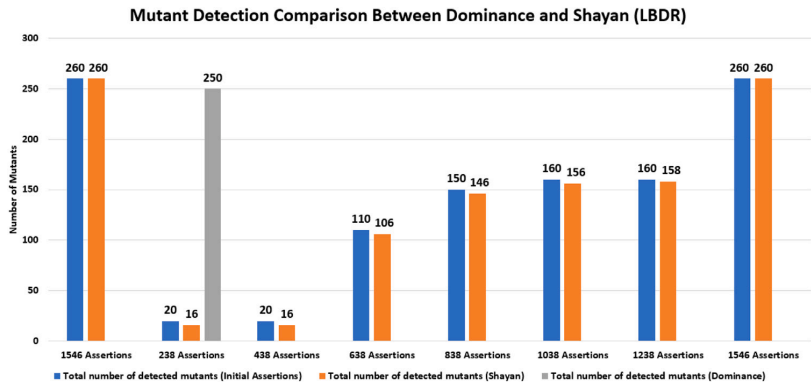


Fig. 6. Comparison between Dominance and Shayan (LBDR).

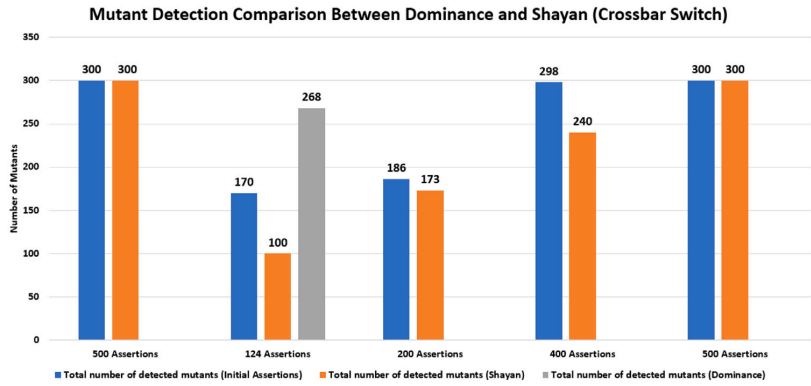


Fig. 7. Comparison between Dominance and Shayan (Crossbar Switch).

three approaches are 100%. Finally, the execution time for the three approaches is as follows 20 m, 19 m, and 13 m.

Regarding the *assertion evaluating* step, we applied the *dominance* algorithm on the assertions generated by the approach in [9] on Arbiter, LBDR and crossbar switch benchmarks, respectively. Moreover, the metrics which are used for this step are taken from the work in [18].

Table 5 demonstrates these results. Column '#Metrics' represents the number of metrics reported for each assertion. In this work, these metrics are *Support, Correlation Coefficient*, and *IS* measure (for detail on how these metrics calculate the quality of assertions, we refer the interested readers to [18]). Column '#Initial assertions' report the number of initial assertions produced by the miner and finally, column

Table 4
Description of injected mutants.

Mutation operator types	List of operators
arithmetic operators	+, -, *, /, %
relational operators	==, !=, >, <, >=, <=
logical operators	&&, ‖
assignment operators	+=, -=, *=, /=, %=, =
unary operators	+, -, ~, !
bitwise operators	≪,≫, &, | , ^
bitwise assignment operators	≪=, ≫=, &=, | =, ^=

Table 5
Result of assertion evaluation after applying dominance algorithm.

Benchmarks	#Metrics	#Initial assertions	#Selected assertions
Arbiter	3	4175	544
LBDR	3	1546	238
Crossbar Switch	3	500	124

'#Selected assertions', demonstrates the number of selected assertions by *dominance*. This number represents the number of high-quality assertions after performing the *assertion evaluating* step. In other words, these assertions are identified as good assertions for the verification process.

As can be seen here, the number of initial assertions for Arbiter and LBDR are 4175 and 1546, respectively. This figure for the crossbar switch is 500 assertions. The number of selected assertions as high-quality assertions that were achieved after applying *dominance* is equal to 544 and 238 for Arbiter and LBDR, respectively. This number is equal to 124 assertions for the crossbar switch. This indicates that according to the proposed *assertion evaluating* phase, by using fewer assertions for verification, we can achieve the same level of accuracy in detecting errors.

To guarantee the efficiency of the proposed *assertion evaluating* step, the results have been evaluated by mutant analysis. Moreover, the proposed approach is compared with an assertion quality evaluator tool called Shayan [17].

In Fig. 5, Fig. 6, and Fig. 7 these results have been presented for Arbiter, LBDR, and crossbar switch, respectively. The blue bar shows the result of mutant analysis, *i.e.*, it represents how many mutants are detected by an assertion set without applying any assertion evaluator. The orange bar represents how many mutants are detected by the high-quality assertions according to Shayan's evaluation. The gray bar indicates how many mutants are detected by the high-quality assertions according to the proposed method in this work.

For Arbiter (Fig. 5), the number of initial assertions is 4175 that they can detect 207 injected mutants (the first set of bars). After applying *dominance* on the initial assertions, 544 assertions out of 4175 have remained that are able to detect 168 injected mutants. On the other hand, the first top 544 assertions of Shayan can detect 62 injected mutants (the second set of bars). The results show that Shayan needs more than 2544 and less than 3044 assertions to detect the same mutants as *dominance* can detect (looking at the 6th and 7th sets of bars). The rest of the bars are demonstrated for more clarification on the accuracy of the *assertion evaluating* phase.

The number of initial assertions for LBDR (Fig. 6) is 1546 that are able to detect at most 260 of the injected mutants (the first set of bars). By applying *dominance* to these initial assertions, we achieved 238 assertions that can detect 250 mutants. On the other hand, the top 238 assertions of Shayan are able to detect only 16 mutants and Shayan needs about 1546 assertions to detect almost the same mutants as *dominance* can detect (looking at the second and the last sets of bars).

Fig. 7 illustrates the results of the comparison between Shayan and *dominance* for the crossbar switch benchmark. The number of initial assertions for this benchmark is 500 which can detect 300 mutants. *Dominance* provides 124 high-quality assertions out of 500 initial assertions that can detect 268 injected mutants, while Shayan can detect 100 mutants and initial assertions detect 170 mutants with 124 assertions.

It should be noted that *dominance* is able to keep those assertions that can detect more mutants and prune the rest. This is the reason that the result of *dominance* has been demonstrated in only one bar (in gray). However, Shayan is not able to prune the assertions. In fact, Shayan is a ranker thus the number of assertions remained unchanged. This is the reason that different bars have been shown for Shayan.

5. Conclusions

This work proposed an efficient methodology for the automated mining of compact and accurate assertion sets. Furthermore, an algorithm has been introduced for evaluating the quality of assertions. In order to mine the assertions, the methodology specified an environment in terms of exhaustive valid simulation traces which served as a complete verification environment for the assertion miner. Consequently, the generated assertions represented a set of valid assertions. Moreover, the assertion evaluator algorithm called *dominance* is able to determine the quality of assertions and provide a unified assertion set. To that end, the experimental results showed that the proposed approach generated significantly more compact assertion sets than the state-of-the-art while achieving 100% fault detection in terms of the injected mutants. The proposed miner is able to generate 10 assertions for SArbiter in less than a minute and 8 assertions for ELBDR in 13 min. Moreover, according to the experimental results, the *dominance* algorithm is capable of selecting the highest quality of assertions that can detect almost the same number of mutants as the initial assertions with much fewer numbers of assertions. In this regard, the *dominance* algorithm selects only 544 assertions out of 4175 assertions for Arbiter and 238 assertions out of 1546 for LBDR. This result for the crossbar switch benchmark is 124 selected assertions out of 500 assertions.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgment

This work was supported in part by the European Union through European Social Fund in the frames of the "Information and Communication Technologies (ICT) programme" ("ITA-IoIT" topic) and by the Estonian Research Council grants PSG837 and PRG1467.

References

[1] M.R.H. Iman, P. Yaghmaie, A software control flow checking technique in multi-core processors, Int. J. Embed. Syst. 13 (2) (2020) 136–147.
[2] M. Boulé, Z. Zilic, Generating Hardware Assertion Checkers: For Hardware Verification, Emulation, Post-Fabrication Debugging and on-Line Monitoring, Springer Publishing Company, 2008, (Incorporated).
[3] M.R. Heidari Iman, J. Raik, G. Jervan, T. Ghasempouri, Immizer: an innovative cost-effective method for minimizing assertion sets, in: 2022 25th Euromicro Conference on Digital System Design (DSD), 2022, pp. 671–678.
[4] X.T. Ngo, J. Danger, S. Guilley, Z. Najm, O. Emery, Hardware property checker for run-time Hardware Trojan detection, in: 2015 European Conference on Circuit Theory and Design, ECCTD, 2015, pp. 1–4.
[5] S. Katz, O. Grumberg, D. Geist, Have I written enough properties? — A method of comparison between specification and implementation, in: Proc. of ACM CHARME, 1999, pp. 280–297.

[6] M. Jenihhin, X. Lai, T. Ghasempouri, J. Raik, Towards multidimensional verification: Where functional meets non-functional, in: 2018 IEEE Nordic Circuits and Systems Conference, NORCAS, 2018, pp. 1–7.
[7] X. Lai, A. Balakrishnan, T. Lange, M. Jenihhin, T. Ghasempouri, J. Raik, D. Alexandrescu, Understanding multidimensional verification: Where functional meets non-functional, Microprocess. Microsyst. 71 (2019) 102867, URL https://www.sciencedirect.com/science/article/pii/S0141933119300250.
[8] J. Malburg, T. Flenker, G. Fey, Property mining using dynamic dependency graphs, in: Asia and South Pacific Design Automation Conference, 2017, pp. 244–250.
[9] G. Martino, G. Fey, Syntax-guided enumeration of temporal properties, in: 2019 Forum for Specification and Design Languages, FDL, 2019, pp. 1–8.
[10] A. Danese, N. Dalla Riva, G. Pravadelli, A-TEAM: Automatic template-based assertion miner, in: Design Automation Conference, 2017 54th ACM/EDAC/IEEE, IEEE, 2017, pp. 1–6.
[11] T. Ghasempouri, A. Danese, G. Pravadelli, N. Bombieri, J. Raik, RTL assertion mining with automated RTL-to-TLM abstraction, in: 2019 Forum for Specification and Design Languages, FDL, 2019, pp. 1–8.
[12] S. Vasudevan, D. Sheridan, S. Patel, D. Tcheng, B. Tuohy, D. Johnson, Goldmine: automatic assertion generation using data mining and static analysis, in: Proc. of ACM/IEEE DATE, 2010, pp. 545–548.
[13] S. Hertz, D. Sheridan, S. Vasudevan, Mining hardware assertions with guidance from static analysis, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 32 (6) (2013) 952–965.
[14] M.W. Anwar, M. Rashid, F. Azam, A. Naeem, M. Kashif, W.H. Butt, A unified model-based framework for the simplified execution of static and dynamic assertion-based verification, IEEE Access 8 (2020) 104407–104431.
[15] T. Ghasempouri, J. Malburg, A. Danese, G. Pravadelli, G. Fey, J. Raik, Engineering of an effective automatic assertion-based verification platform, in: 6th Workshop on Design Automation for Understanding Hardware Designs, DUHDe 2019, 2019, pp. 557–562, URL https://elib.dlr.de/127128/.
[16] T. Ghasempouri, J. Malburg, A. Danese, G. Pravadelli, G. Fey, J. Raik, Engineering of an effective automatic dynamic assertion mining platform, in: 2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration, VLSI-SoC, 2019, pp. 111–116.
[17] T. Ghasempouri, G. Pravadelli, On the estimation of assertion interestingness, in: 2015 IFIP/IEEE International Conference on Very Large Scale Integration, VLSI-SoC, 2015, pp. 325–330.
[18] T. Ghasempouri, S. Payandeh Azad, B. Niazmand, J. Raik, An automatic approach to evaluate assertions' quality based on data-mining metrics, in: 2018 IEEE International Test Conference in Asia, ITC-Asia, 2018, pp. 61–66.
[19] R. Hariharan, T. Ghasempouri, B. Niazmand, J. Raik, From RTL liveness assertions to cost-effective hardware checkers, in: 2018 Conference on Design of Circuits and Integrated Systems, DCIS, IEEE, 2018, pp. 1–6.
[20] D. Pal, S. Offenberger, S. Vasudevan, Assertion ranking using RTL source code analysis, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (8) (2020) 1711–1724.
[21] G. Fey, T. Ghasempouri, S. Jacobs, G. Martino, J. Raik, H. Riener, Design understanding: From logic to specification*, in: 2018 IFIP/IEEE International Conference on Very Large Scale Integration, VLSI-SoC, 2018, pp. 172–175.
[22] M.R. Heidari Iman, J. Raik, M. Jenihhin, G. Jervan, T. Ghasempouri, A methodology for automated mining of compact and accurate assertion sets, in: 2021 IEEE Nordic Circuits and Systems Conference, NorCAS, 2021, pp. 1–7.
[23] C. Deutschbein, C. Sturton, Mining security critical linear temporal logic specifications for processors, in: 2018 19th International Workshop on Microprocessor and SOC Test and Verification, MTV, 2018, pp. 18–23.
[24] M. Shahin, M.R. Heidari Iman, M. Kaushik, R. Sharma, T. Ghasempouri, D. Draheim, Exploring factors in a crossroad dataset using cluster-based association rule mining, Procedia Computer Science 201 (2022) 231–238, https://www.sciencedirect.com/science/article/pii/S1877050922004458, The 13th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 5th International Conference on Emerging Data and Industry 4.0 (EDI40).
[25] E.T. Ogidan, K. Dimililer, Y.K. Ever, Machine learning for expert systems in data analysis, in: 2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies, ISMSIT, 2018, pp. 1–5.
[26] R. Kerr, R. Ebsary, Implementation of an expert system for production scheduling, European J. Oper. Res. 33 (1) (1988) 17–29, URL https://www.sciencedirect.com/science/article/pii/0377221788902500.
[27] P. Saltarelli, B. Niazmand, R. Hariharan, J. Raik, G. Jervan, T. Hollstein, Automated minimization of concurrent online checkers for Network-on-Chips, in: 2015 10th International Symposium on Reconfigurable Communication-Centric Systems-on-Chip, ReCoSoC, 2015.
[28] S. Sharma, S. Kumar, A.K. Mishra, D. Vaithiyanathan, B. Kaur, PVT aware analysis of ISCAS C17 benchmark circuit, in: R.M. Singari, K. Mathiyazhagan, H. Kumar (Eds.), Advances in Manufacturing and Industrial Engineering, Springer Singapore, Singapore, 2021, pp. 1199–1211.
[29] R.K. Brayton, A.L. Sangiovanni-Vincentelli, C.T. McMullen, G.D. Hachtel, Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publishers, USA, 1984.
[30] S. Bouker, R. Saidi, S. Ben Yahia, E. Mephu Nguifo, Mining undominated association rules through interestingness measures, Int. J. Artif. Intell. Tools 23 (04) (2014) 1460011.
[31] S. Bouker, R. Saidi, S.B. Yahia, E.M. Nguifo, Ranking and selecting association rules based on dominance relationship, in: 2012 IEEE 24th International Conference on Tools with Artificial Intelligence. Vol. 1, 2012, pp. 658–665.
[32] Project Bonfire Network-on-Chip, 2017, https://github.com/Project-Bonfire/Bonfire_handshake.
[33] U. Repinski, H. Hantson, M. Jenihhin, J. Raik, R. Ubar, G. Di Guglielmo, G. Pravadelli, F. Fummi, Combining dynamic slicing and mutation operators for ESL correction, in: 2012 17th IEEE European Test Symposium, ETS, 2012, pp. 1–6.

Mohammad Reza Heidari Iman is a Ph.D. candidate in the Department of Computer Systems at Tallinn University of Technology (TalTech), Estonia. He received his master's degree from Ferdowsi University of Mashhad, Iran in 2015. His research interests include verification and assertion-based verification in embedded systems, as well as the application of data mining in the verification of hardware designs.

Jaan Raik is a professor of digital systems verification at the Department of Computer Systems at TalTech and the leader of the Center for Dependable Computing Systems Design (DCSD). Prof. Raik received his M.Sc. and Ph.D. degrees in Computer Engineering from TalTech in 1997 and in 2001, respectively. He is a member of the IEEE Computer Society, HiPEAC and a member of steering/program committees of several conferences. He has co-authored more than 200 scientific publications.

Maksim Jenihhin is an associate professor of Computing Systems Reliability at the Department of Computer Systems of Tallinn University of Technology and the head of the research group "Trustworthy and Efficient Computing Hardware". He received his Ph.D. degree in Computer Engineering from the same university in 2008. His research interests include methodologies and EDA tools for hardware design, verification and debugging as well as nanoelectronics reliability and manufacturing test topics. He is a coordinator for national and European research projects and a member of executive and program committees for IEEE ETS, DATE, DDECS, and a number of other international events.

Gert Jervan is a professor of dependable computer systems at the Department of Computer Systems at TalTech. He is also the Dean of the School of IT at TalTech. He received his Ph.D. degree from Linköping University, Sweden in 2005. His research interests include reliability, security, fault tolerance, dependability, and biomedical engineering. He has been a coordinator of several national and international research projects, including several EU networks of excellence and other EC projects. He is Senior Member of IEEE.

Tara Ghasempouri is a Senior Researcher at the Department of Computer Systems at TalTech. She is interested in finding innovative solutions for the verification process at different levels of abstraction. Her research topic is also focused on security verification in hardware and software domains. She received a Ph.D. degree in Computer Science from the University of Verona, Italy. During her Ph.D. program, she researched on different kinds of verification and specially Assertion-based Verification on the embedded system. She is a member of the IEEE Computer Society and she is program committee for different conferences such as DATE, ETS, VLSI-SOC and etc.

Appendix 3

III

MRH. Iman, J. Raik, G. Jervan, and T. Ghasempouri, "IMMizer: An Innovative Cost-Effective Method for Minimizing Assertion Sets," *2022 25th Euromicro Conference on Digital System Design (DSD)*, Maspalomas, Spain, 2022, pp. 671-678, DOI: https://doi.org/10.1109/DSD57027.2022.00095.

IMMizer: An Innovative Cost-Effective Method for Minimizing Assertion Sets

Mohammad Reza Heidari Iman, Jaan Raik, Gert Jervan and Tara Ghasempouri
Department of Computer Systems, Tallinn University of Technology, Tallinn, Estonia
{mohammadreza.heidari, jaan.raik, gert.jervan, tara.ghasempouri}@taltech.ee

***Abstract*—Assertion-based verification is one of the viable solutions for the verification of computer systems. Assertions can be automatically generated by assertion miners however, these miners typically generate a high number of possibly redundant assertions. In turn, this results in higher costs and overheads in the verification process. Furthermore, these assertions have every so often low readability due to the high number of propositions that they contain. In this paper, an Innovative cost-effective Method for Minimizing assertion sets (IMMizer) has been proposed. IMMizer is performed by identifying *Contradictory Terms*. These terms present the behaviors of the design under verification which are not specified by the initial assertion sets. Subsequently, a new assertion set is extracted based on the identified *Contradictory Terms*. Contrary to data-mining approaches that are unable to minimize the initial assertion set, but can only rank the set according to data-mining measurements, or mutant analysis approaches that require a long execution time, IMMizer is able to minimize the initial assertion set in a very short execution time. Experimental results showed that in the best case, this method has drastically reduced the number of assertions by 93% and the memory overhead imposed on the system by 87%, without any reduction in the detection of injected mutants.**

Keywords: Hardware Verification, Assertion-Based Verification, Assertion Mining, Assertion Minimization

I. Introduction

Digital and embedded systems become more complex with each generation. These systems are employed in different industries and safety-critical applications such as medical and automotive [1]. Due to the application of these systems, their correctness, safety, robustness, and in general, dependability have a significant importance [1–4]. One of the important concepts that can affect the dependability of these systems, is their verification process [5]. The verification process ensures that a system is developed consistently to its specifications and documentation [6].

One of the promising approaches for design verification is Assertion-Based Verification (ABV). Assertions play a key role in ABV. Assertions are Boolean expressions that define the designs' behaviors. They principally are considered as the 'golden rules' to which the implementation is compared and any deviations from these rules can lead to design errors [7].

To this end, several approaches have been carried out to automatically generate assertions [6–12]. However, these approaches generally extract a very high number of assertions that are typically redundant *i.e.* describe the same behavior of the Design Under Verification (DUV). Therefore, it leads to more cost and time in the verification process.

Moreover, the generated assertions are typically not readable by human experts. These assertions mostly contain a lot of propositions that reduce their readability. In addition, each proposition allocates at least one memory bit for itself, consequently imposing overheads (*e.g.* memory overhead) on the system.

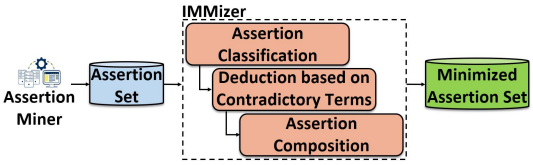


Figure 1: General flow of the IMMizer

Thereby, to resolve the mentioned drawbacks several studies have been conducted to minimize the extracted assertion sets, consequently, selecting the interesting ones for the verification process. These approaches are mainly categorized into two groups, first, data-mining-based [13–19] and second, mutant-analysis-based approaches [20–22]. In the first approach, the assertions are ranked according to several data-mining metrics to select the highest-ranked ones for the verification process. However, in the second approach mutants are injected into the DUV and the assertions which detect more mutants are typically selected for the verification process.

For instance, the work in [13] ranks the assertions based on two main metrics, *i.e.*, Importance and Complexity. The former is higher in the assertions which are describing the output of the design and the latter is related to the number of logic that has been used in an assertion. In [14], the interestingness measurement is according to the number of propositions included in the antecedent of assertions. In some other studies such as the works in [16–19], mined assertions are mainly ranked according to several data-mining metrics. These metrics are such as Support, Correlation Coefficient (CC), and IS. Support is the frequency of occurrences of assertions during the simulation, while CC is the correlation of the occurrence of an assertion to other assertions during the simulation. Furthermore, IS determines assertions that have a low frequency of occurrence but are highly correlated to other assertions.

In spite of the fact that the aim of the aforementioned studies is to generally minimize and reduce the number of assertions, they can only select a few of the best assertions among the others. In fact, they are not capable of minimizing the initially generated assertion set to exclude its redundancy and as a consequence decrease the overheads that have been imposed on the system. In practice, in these approaches, the number of assertions remained unchanged, and the verification engineer selects a subset of an assertion set. However, this kind of assertion selection typically leads

2022 25th Euromicro Conference on Digital System Design (DSD) | 978-1-6654-7404-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/DSD57027.2022.00095

2771-2508/22/$31.00 ©2022 IEEE
DOI 10.1109/DSD57027.2022.00095

to issues such as inaccuracy in the verification process and lower behavior coverage of designs.

To overcome these shortcomings, this paper proposes IMMizer, an Innovative cost-effective Method for Minimizing assertion sets. The main idea of IMMizer is performed by identifying *Contradictory Terms* in the assertion sets. *Contradictory Terms* present the behaviors of the DUV which are not specified by the initially generated assertion set. Subsequently, a new assertion set is extracted based on the identified *Contradictory Terms* (more detail in section III). Fig. 1, demonstrates the overall flow of the proposed IMMizer. As is shown here, the input of the flow is an assertion set that is automatically generated by an assertion miner. The output of the flow is a minimized assertion set where all the redundant assertions have been pruned. IMMizer is composed of three main steps, *i.e.*, Assertion classification, Deduction based on Contradictory Terms, and Assertion composition. The first step finds the most similar assertions (assertions with the same propositions and the same temporal patterns) in order to classify them together. The second step applies to the classified assertion sets to find the *Contradictory Terms*. Finally, the third step composes the identified similar propositions and the *Contradictory Terms* from the previous steps to extract the new minimized assertion set.

Overall, the proposed IMMizer aims at reducing the verification costs and time by pruning the redundant assertions, *i.e.*, assertions that describe the same behavior of the design, and providing minimized and more effective ones to the verification engineers. Therefore, the contributions of the paper are listed as follows:

- significantly reduction in the number of (redundant) assertions in a very short amount of time without any decrements in mutant detection in comparison with the original assertion set;
- reducing the memory overhead imposed on the system as a result of assertion minimization;
- improving the readability of assertions by reducing the number of propositions of assertions;
- detecting the unique and non-redundant assertions for a more effective verification process.

The remainder of the paper is organized as follows: The preliminary concepts have been discussed in section II. Section III presents the proposed method for minimization of assertions. Section IV, discusses the computation time of IMMizer. The experimental results have been elaborated in section V and finally, section VI concludes the paper.

II. Preliminaries

In this section, the definitions and concepts which are deployed in this article are briefly explained to assist the readers in better understating the method.

Definition 1: An *Atomic Proposition* is a logic formula that does not contain logical connectives [10]. A ***Proposition*** is a composition of *atomic propositions* through logical connectives [10]. Examples of *atomic propositions* and *propositions* are presented in Listing 1 and 2, respectively.

Listing 1: Example of atomic propositions

```
a1 = True, b1 = False, c1 = True
```

Listing 2: Example of propositions

```
a1 = True && b1 = False && c1 = True
```

Definition 2: An ***Assertion*** is a composition of propositions through some temporal operators. An assertion is typically divided into two parts, the left side of an implication which is called *antecedent*, and the right part of the implication which is called *consequent* [10].

Listing 3 represents two examples of assertions in SystemVerilog Assertions (SVA) language [23]. Property p1 indicates whenever a && !b occurs then at the same time instant (represented by |->) t1 occurs. Property p2 indicates whenever a && b && !c occurs then after 2 clock cycles (represented by ##2) t2 occurs.

Listing 3: Example of assertions

```
property p1;
   @(posedge clk) a && !b |-> t1;
endproperty

property p2;
   @(posedge clk) a && b && !c |-> ##2 t2;
endproperty
```

Definition 3: We define ***Different Terms*** as a set denoted by *DT* where represents dissimilar *propositions* in the *antecedent* of an assertion in comparison with another assertion.

In Listing 3, *(!b -)* and *(b !c)* in properties p1 and p2, respectively, are considered as elements of *DT*. Since there is neither proposition c nor !c in property p1, it is considered as don't care (-). Therefore, the set *DT* is as follows:

DT = {(!b -), (b !c)} = {(!b c), (!b !c), (b !c)}, where two different values, c and !c are assigned to '-'.

Note that *a* is not considered as an element of *DT* since it exists in both assertions.

Definition 4: ***Sum-Of-Products (SOP)*** is a type of Boolean algebra expression in which different product inputs are being added together [24].

For instance, for the set *DT* in *Definition 3*, *SOP* is as follows:

SOP(DT) = (!b c) + (!b !c) + (b !c) = !b(c + !c) + (b !c) = !b + (b !c)

Definition 5: ***Contradictory Terms*** are equal to the complement of the *SOP* of the set *DT*, denoted by $SOP(DT)^C$.

Complement for the calculated *SOP* in *Definition 4* is as follows:

$SOP(DT)^C = (!b + (b\ !c))^C = b(!b + c) = (b\ c)$

To make the reason of calculating *SOP* and its complement in IMMizer more clear, let's say we have two terms β and γ and the Universal set U equal to the all different combinations for these two terms:

$U = \{(\beta\ \gamma), (\beta\ !\gamma), (!\beta\ \gamma), (!\beta\ !\gamma)\}$

Let's assume the set of different combinations produced by each *DT* is equal to the following set:

$DT = \{(\beta\ \gamma), (\beta\ !\gamma), (!\beta\ !\gamma)\}$

The *SOP* for the set *DT* is as follows:

$SOP(DT) = (\beta\ \gamma) + (\beta\ !\gamma) + (!\beta\ !\gamma) = \beta + (!\beta\ !\gamma)$

To find the *Contradictory Terms*, the complement of *SOP(DT) i.e.*, $(SOP(DT)^C)$ is calculated, which is:

$SOP(DT)^C = (\beta + (!\beta\ !\gamma))^C = !\beta(\beta + \gamma) = (!\beta\ \gamma)$

$SOP(DT)^C$ is equal to the terms that are not in the set *DT* such that:

$SOP(DT)^C \notin DT$ and $SOP(DT)^C \cup DT = U$

In fact, $SOP(DT)^C$ consists of the term that has not been appeared in the set DT and can complete the set U. This is the *Contradictory Term* that IMMizer aims at finding it, *i.e.* the terms that do not exist in the set DT and will complete the set DT to reach the Universal set U so that:

$SOP(DT)^C \cup DT = U$

After achieving the $SOP(DT)^C$ (*Contradictory Term*), IMMizer will finally calculate the $\sim (SOP(DT)^C)$ (more details in section III-C). The reason is that:

$\sim (SOP(DT)^C) = \sim (!\beta\ \gamma) = (\beta + !\gamma)$ (*i.e.* β or $!\gamma$). By comparing the set DT with $(\beta + !\gamma)$, it can be proved that all the three terms in the set DT $((\beta\ \gamma), (\beta\ !\gamma), (!\beta\ !\gamma))$ have been covered by the new term $(\beta + !\gamma)$. In fact the new term $(\beta + !\gamma)$ is equal to all the previous terms in the set DT and can be replaced with them by IMMizer.

A. Structure of generated assertions studied in this work

In this work, IMMizer is applied to different assertions extracted by different assertion miners [8–10]. Generally, these assertions are produced in different formats such as, SVA, Property Specification Language (PSL) [25], as well as Linear Temporal Logic (LTL) [26]. Due to the fact that an advantage of IMMizer is to be applicable to assertions written in different languages, in the following, we present some examples which are studied in this article.

Listing 4 presents an example of assertions generated by the miner introduced in [9]. Listing 5, shows the extracted assertions of the work in [8]. Finally, Listing 3 represents the format of assertions produced by the assertion miner in [10]. Any of these assertions are describing a specific behavior of the DUV. For example, property p1 in Listing 4 means that eventually from 0 to an infinite clock cycle later, the proposition *a==1* will happen, and then it implies that eventually from 0 to an infinite clock cycle later *!(t1==1)* will happen.

Listing 4: Example of generated assertions in [9]

```
property p1;
   (((s_eventually[0:$]( a==1 )))) implies (((s_eventually
       [0:$](!(t1==1)))));
endproperty
```

Listing 5: Example of generated assertions in [8]

```
property p1;
   @(posedge clk)
   ((reset)
     ##1 (!reset && !(c==’b1) && !(d==’b1))[*2]
     ##1 (!reset && !(a==’b1) && !(c==’b1) && !(d==’b1))
     ##1 ((a==’b1) && !reset && !(b==’b1))
     ##1 ((d==’b1) && !reset && !(a==’b1) && !(b==’b1))
     ##2 ((c==’b1) && !reset && !(b==’b1) && !(e==’b1))
     ##1 (!reset && !(b==’b1) && !(e==’b1))[*8]
     ##5 ((b==’b1) && !reset)
     ##1 ((a==’b1) && (b==’b1) && !reset))
     |-> ##1 t1==’b100;
endproperty
```

III. Proposed Methodology

In this section, the proposed method for minimizing the assertion sets has been discussed. As mentioned in Fig. 1, IMMizer consists of the following main steps:

- Assertion Classification
- Deduction based on Contradictory Terms
- Assertion Composition

The first step (section III-A) works by applying three different classifications to the assertions provided by assertion miners. These classifications are performed based on the similarity of consequents, temporal patterns, and antecedents of assertions (*Definition 2*), respectively. These classifications help IMMizer to group assertions more homogeneously therefore the flow can minimize the assertions easier and faster.

The second step (section III-B) consists of two sub-steps which are 1) Proposition Extraction (Section III-B1), and 2) Proposition Subtraction (Section III-B2). This step is the core of IMMizer and the main tasks of minimization are performed here. The method at first performs the 'Proposition Extraction' on the classified assertion sets to analyze propositions (*Definition 1*) in all the antecedents. If there are *Different Terms (DT)* (*Definition 3*) among the antecedents, 'Proposition Subtraction' applies to find the *Contradictory Terms* (*Definition 5*). This is done by calculating the *Sum-Of-Products* (*SOP*) (*Definition 4*).

In the third step (Assertion Composition (section III-C)), a new set of minimized assertions is generated based on the provided ingredients from the previous steps.

The above flow is elaborated in Algorithm 1 for a better demonstration of the overview of the proposed method and has been discussed in more detail in the following subsections.

Algorithm 1 IMMizer

1: **Assertion Classification:**
2: Apply classification based on consequent of assertions;
3: Apply classification based on temporal patterns of assertions;
4: Apply classification based on the antecedent of assertions;
5: **Deduction based on Contradictory Terms:**
6: Find the similar terms in the antecedent of the classified assertions;
7: Find the *DT* in the antecedent of the classified assertions and consider different combinations that each *DT* can produce;
8: Calculate the *Sum-Of-Products (SOP)* for *DT*s;
9: Calculate complement of *SOP* and consider it as *Contradictory Term*;
10: **Assertion Composition:**
11: If the result of Step 9 is equal to 0 then omit all the *DT*s from the minimized assertion;
12: If the result of Step 9 led to new propositions then apply a negation (~) before them and replace these new propositions with the *DT*s in the minimized assertion.

A. Assertion Classification

This step aims at classifying the initial assertion set, in order to prepare the appropriate conditions for the next steps. Classifying assertions leads to placing similar assertions in the same group and thus more effectively discovering the redundant assertions.

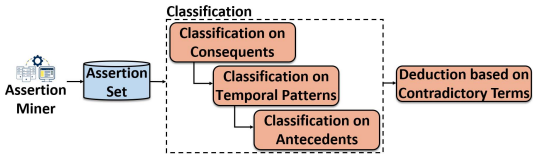


Figure 2: Three different classifications

The first classification is based on the similarities among the consequent of the assertion sets. This means that, in

order to minimize assertions with each other, they have to be related to the same consequent. Otherwise, logically it is not possible to minimize and compact them together as they are describing different behaviors of the DUV.

As an illustrative example, we have considered some of the assertions provided by A-Team [10], an assertion miner for one of our benchmarks (Arbiter), in Listing 6. After performing the first classification, the assertions are classified according to the Listing 7 and in three different classes (Class 1, Class 2, and Class 3). This is due to the fact that the corresponding consequent for each obtained class is similar.

Listing 6: A group of assertions provided by A-Team [10]

```
1) !A && B && !C |-> F
2) A && D |-> G
3) B && C && !D |-> ##2 G
4) !A && !C |-> F
5) !A && C |-> F
6) B && C |-> ##3 F
7) A && B && C ##1 !A && !C |-> ##2 !F
8) A && !C ##1 !A && D && C |-> ##2 !F
9) A && C ##1 !A && !C |-> ##2 !F
```

Listing 7: First classification on consequents

```
Class 1:
1) !A && B && !C |-> F
4) !A && !C |-> F
5) !A && C |-> F
6) B && C |-> ##3 F
Class 2:
7) A && B && C ##1 !A && !C |-> ##2 !F
8) A && !C ##1 !A && D && C |-> ##2 !F
9) A && C ##1 !A && !C |-> ##2 !F
Class 3:
2) A && D |-> G
3) B && C && !D |-> ##2 G
```

Note that, in the above Listings, *A, !A, B, C, !C, D,* and *!D* are in the antecedent part of the assertions, and *F, !F,* and *G* are in their consequents. Moreover, *##2*, and *##3* determine the number of clock cycles where consequent occurs after antecedent.

After doing classification on consequents of assertions, the second classification which aims at categorizing the assertions based on the behavior of temporal patterns is performed. Listing 8 has presented the result of the second classification. The assertions that we analyzed in this research are with different temporal patterns. Thereby, to minimize these assertions, we have to classify them based on the similarities of their temporal patterns.

Listing 8: Second classification on clock cycles

```
Class 1.1:
1) !A && B && !C |-> F
4) !A && !C |-> F
5) !A && C |-> F
Class 1.2:
6) B && C |-> ##3 F
Class 2.1:
7) A && B && C ##1 !A && !C |-> ##2 !F
8) A && !C ##1 !A && D && C |-> ##2 !F
9) A && C ##1 !A && !C |-> ##2 !F
Class 3.1:
2) A && D |-> G
Class 3.2:
3) B && C && !D |-> ##2 G
```

In the last classification, IMMizer works heuristically to find the assertions that their antecedents are with the most similarities. In fact, it finds and classifies those assertions that have more similar propositions in their antecedents. Going back to the illustrative example in Listing 8, in Classes 1.1 and 2.1, the assertions are with the most similar propositions in their antecedents among all the other assertions. Accordingly, assertions in Classes 1.2, 3.1, and 3.2 are unique and without any redundancy. Thus, the final classification's result will be the two dark gray parts which are shown in Listing 8.

After performing the three classifications and categorizing the assertions for minimization, the next step, *i.e. Deduction based on Contradictory Terms*, is applied to the classified assertions to minimize them. In the following, these processes are discussed in more detail.

B. Deduction based on Contradictory Terms

In this section, the process of 'deduction based on Contradictory Terms' has been discussed. The idea is to distinguish the behavior of the DUV which is not illustrated by the assertion set. For this purpose, the assertions which are collected in the same class (in the previous section) are analyzed to determine which behavior of the DUV is not covered by them. These missing behaviors are called *Contradictory Terms*. Consequently, these *Contradictory Terms* are used to create a new assertion set.

This section mainly consists of two steps, *i.e.*, 1) Proposition Extraction and; 2) Proposition Subtraction. The first step looks for the similar terms, as well as the *Different Terms i.e.*, dissimilar propositions among the antecedent of assertions, and the second step instead, aims at finding the *Contradictory Terms* by calculating the *SOP* for dissimilar propositions in the antecedent of assertions. More detail is described in the following.

1) Proposition Extraction

This step determines *DT*s (*Definition 3*), in a classified assertion set. For this purpose, first, the propositions that are the same in the antecedents of the classified assertion sets are identified. Second, dissimilar propositions among the assertions are determined. In other words, *DT*s are extracted. Consequently, with calculating *DT*s, conditions that lead to a 'don't care situation, *i.e.*, '-', are defined.

As illustrative examples, part B.1 in Fig. 3, shows how Proposition Extraction is performed for Class 1.1 in Listing 8. Note that the next steps, *i.e.*, Proposition Subtraction and, Assertion Composition are shown in this figure as step B.2 and step C which are described later.

Looking at Fig. 3, in the antecedent part of the three assertions in the box *'Classified Assertions'*, *!A* has been appeared in all of them, however, *B* and *C* have been appeared with different values. Therefore, in the provided example, we keep *!A* as it is. Subsequently, *B* and *C* have been considered as *DT*.

At this step, all combinations that *DT*s of any of the assertions can produce are computed. Combinations refer to the fact that if a proposition does not occur in the classified assertion set, a don't care situation, '-' is placed. For instance, in the first assertion in the *'Classified Assertions'* box, all the propositions do occur, as a result, no don't care is placed wherefore *B !C* is written (in the box of *Different Terms (DT)*). In the second assertion, instead, there is no

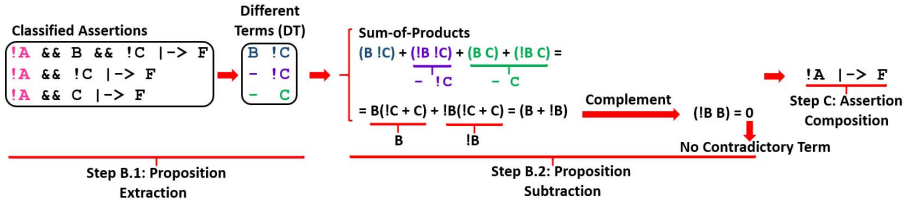


Figure 3: Example 1 for minimizing the assertions

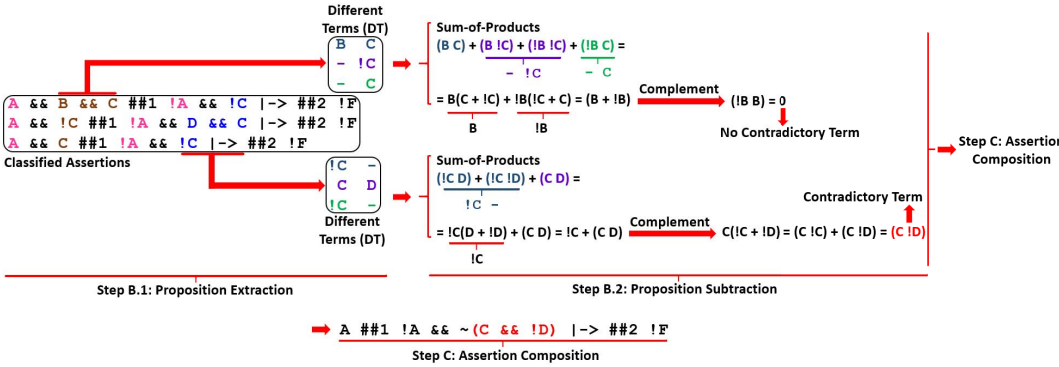


Figure 4: Example 2 for minimizing the assertions

proposition either for *B* or *!B*, subsequently, it is considered as don't care (-). Therefore, for this assertion, we have *- !C* in the box of *Different Terms (DT)* in the illustrated example in Fig. 3. This process has been done for the third assertion, as well, which leads to calculating, *- C*.

In the condition that the classified assertions include a temporal operator (*e.g. ##1*) in the antecedent, the workflow of IMMizer is slightly changed. In fact, IMMizer has to break down the antecedents corresponding with the number of temporal operators which is used in each antecedent of the classified assertion. Accordingly, *DT*s are determined.

Fig. 4, shows an example of a classified assertion set including a temporal operator in the antecedent. As can be seen here, two flows are created, one regarding the part of antecedent before the temporal operator *##1* and one flow related to the part of antecedent after *##1*. In the *'Classified Assertions'* box, the propositions for the first flow are shown as 'brown' and propositions for the second flow are shown as 'blue'. As described previously, *DT* boxes are calculated, correspondingly.

2) Proposition Subtraction

The aim of this step is to find the behavior of DUV which is not described in the classified assertion set, *i.e.* finding the *Contradictory Terms*. This is performed, first, by extracting all the possible combinations which can be produced from the box of *Different Terms* (previous step). Second, by extracting *Contradictory Terms*, based on the *DT*. For this purpose, *Sum-Of-Products (SOP)* for the *DT* set is calculated and its corresponding complement is determined. This result is used later to create a new and minimized assertion set.

Going back to the illustrative example in Fig. 3, as can be seen, *DT = {B !C, - !C, - C}*. At this step, all the possible combinations which can be produced by replacing '-' are calculated. Thus, *DT* will be equal to *{B !C, !B !C, B C, !B C}*. This is due to the fact that for the first assertion *B !C*, for the second assertion, *B !C*, and *!B !C*, and for the third assertion *B C* and *!B C* are produced in *DT* set, respectively. Accordingly, *SOP* of this set can be written as is reported in Step B.2, *i.e.*, (B !C) + (!B !C) + (B C) + (!B C). Obviously, the final result of *SOP* is equal to *(B + !B)*.

At this phase, the *Contradictory Term* which is the complement of *(B + !B)* is extracted *i.e., (!B B) = 0*. In this example, this value is equal to 0, which means that the *DT*s (*B, !B, C,* and *!C*) should be removed and not added to the minimized assertion.

As said, for the classified assertions which include a temporal operator in their antecedents the workflow is slightly different. In such a case, IMMizer considers the before and after of each temporal operator as a separate flow. Accordingly, *SOP* and *Contradictory Terms* are calculated for each flow. Going back to the illustrative example in Fig. 4, there is *##1* in the antecedent of the assertions. Therefore, two boxes for *DT*s have been complied from the assertion set. For the left side of the *##1*, *DT = {(B C), (B !C), (!B !C), (!B C)}*. The calculated *SOP* for this part is equal to *(B + !B)* that its complement is *(!B B) = 0*. Therefore, for the left side of the *##1*, *B, !B, C,* and *!C* will be removed from the final minimized assertion. On the other hand, for the right side of the *##1*, the *DT* box consists of *{(!C -), (C D), (!C*

-)} = {(!C D), (!C !D), (C D)}. The *SOP* and its corresponding complement are *(!C + (C D))* and *(C !D)*, respectively. *(C !D)* means that *(C && !D)* is the new term that will be replaced with the *DT*s and will be added to the right side of the *##1* in the minimized assertion.

C. Assertion Composition

In this step, the final minimized assertion set is produced. For this purpose, the result of $(SOP(DT)^C)$ is considered. If the result is equal to 0, it indicates that the *DT*s should be removed and not added to the final assertion set. Otherwise, if the result of $(SOP(DT)^C)$ is equal to a set of propositions, a negation (i.e., ~) is applied on $(SOP(DT)^C)$, *i.e.*, $\sim(SOP(DT)^C)$. Consequently, $\sim(SOP(DT)^C)$ is added to the final result. In other words, the similar proposition in the classified assertions remains as it is, however, all the produced *DT*s are replaced by $\sim(SOP(DT)^C)$.

Going back to the illustrative example in Fig. 3, the result of $(SOP(DT)^C)$ is equal to *B !B = 0*. Thus, all the *DT*s, i.e, *B !C, - !C, - C* are removed from the assertions in the 'Classified assertion' box. Thereby, the final assertion set is minimized to only one assertion (*!A |-> F*). As can be seen, the final assertion set has a significantly lower number of propositions in comparison with the classified assertions.

Regarding Fig. 4, two *Contradictory Terms* are identified. The first is equal to '0' and the second is equal to *(C !D)*. '0' indicates that the *DT*s should be removed from the final assertion set, however, for the result of the second flow a negation is applied on *(C !D)*, *i.e.*, *~(C !D)*. Therefore, the generated minimized assertion is equal to: *A ##1 !A && ~(C && !D) |-> ##2 !F*.

IV. Discussion on computation time of IMMizer

In this section, we elaborate that the proposed method can perform the algorithm in a reasonable execution time and without any additional overheads on the system.

As mentioned before, in the Deduction step, to find the *Contradictory Terms*, IMMizer needs to consider different combinations produced by each *DT*, and calculate the *SOP* and the complement of *SOP*. This can cause more execution time in cases where there are too many propositions (with different values) in the antecedent of assertions. To resolve this issue and to prevent IMMizer from additional comparisons, equations (1) and (2) have been employed.

$$u = \sum_{i=1}^{n} 2^{\#don't\ cares} + \sum \#combinations\ without\ don't\ care \tag{1}$$

In equation (1), u is the number of all possible combinations that classified assertions can produce. The first part of this equation, *i.e.*: $\sum_{i=1}^{n} 2^{\#don't\ cares}$, is related to those combinations of *Different Terms* (*DT*) that have at least one don't care (-). n is the number of combinations with don't care in *DT* and $n \geqslant 1$. If there is no combination with don't care, this part of the equation will be 0. $\#don't\ cares$ is the number of don't cares in each combination of *DT*. Furthermore, $\#combinations\ without\ don't\ care$ is the number of combinations in *DT* that do not have any don't care.

$$\mathcal{F} = r - (u - s) \tag{2}$$

In equation (2), s is the number of repetitive created combinations produced by different terms (*e.g. B !C* in Fig. 3) and r is the number of all possible combinations that these terms (*e.g. B* and *C* in Fig. 3) can produce. The value of $\mathcal{F}$ determines whether IMMizer can classify assertions together or not.

If $\mathcal{F} <$ *(#grouped assertions)*, IMMizer can classify them. Otherwise, the assertions cannot be classified with each other to perform the minimization on them. The reason is that in this case, the number of generated *Contradictory Terms* is more than or equal to the number of assertions that have been grouped together, and in fact, it generates more or equal assertions than the number of first assertions that had been grouped together. *#grouped assertions* is the number of assertions that IMMizer has grouped together based on the most similarities in the proposition of their antecedent part and aims at minimizing them. In the illustrative example in Fig. 3, u is equal to 5 $(2^1 + 2^1 + 1)$, and s and r are equal to 1 and 4, respectively. Thereby, the value of $\mathcal{F}$ is 0, which is less than 3 (3 is the number of assertions that we have grouped for minimization (*#grouped assertions*)).

The important point in the proposed minimizer is that it first calculates the value of $\mathcal{F}$ and after that, decides to whether minimize or not the grouped assertion set. To calculate the value of $\mathcal{F}$, IMMizer needs to only find the *Different Terms (DT)* and the number of don't cares in each *DT*. Subsequently, at first, there is no need for finding the different combinations produced by each *DT*, calculating the *SOP* and other computations and processes of the IMMizer in the Deduction step. The parameters in $\mathcal{F}$, *i.e.* r, u, and s can be calculated by only finding the *Different Terms (DT)* and the number of don't cares in each *DT*. This can prevent the overheads that might be imposed on the system in cases where there are a lot of propositions with different values in the antecedent part of the assertions and moreover, it reduces the execution time.

V. Experimental Results

To evaluate the efficiency of the method, the proposed minimizer has been applied to the generated assertions of different assertion miners. These miners are related to the works in [8], [9], and [10].

The miners are applied to an NoC framework, (an open-source project named NoC router Bonfire [27]) and two of its important components which are Arbiter and LBDR. Arbiter connects input links to the output links according to the routing algorithm. LBDR instead is the control part of the router, used for calculating the candidate output ports to forward the packets. These two components are critical since they build the control part of the router thus, their verification is essential.

In Table I, the number of initial assertion sets (*#Initial assertions*) provided by the assertion miners, the number of minimized assertion sets (*#Minimized assertions*), as well as the percentage of reduction in the number of assertions (*Reduction in #assertions (%)*) have been reported. Furthermore, the percentage of detected mutants by the initial set which is correspondingly detected by the minimized set (*Mutant detection (%)*), and the execution time of IMMizer (*Execution Time (Sec)*) have been presented. One advantage of IMMizer is its ability in minimizing assertions generated by different approaches. As mentioned in section II, the

Table I: Reduction in the number of assertions after applying IMMizer, the report on what percentage of the detected mutants by the initial set are correspondingly detected by the minimized set, and the execution time of IMMizer for each benchmark.

Assertion Miners	#Initial Assertions	#Minimized Assertions	Reduction in #Assertions (%)	Mutant Detection (%)	Execution Time *(Sec)*
A-Team [10] - Arbiter	318	198	38	100	15
DDG [8] - Arbiter	207	199	4	100	21
Assertion Miner [9] - Arbiter	4175	304	93	100	60
Assertion Miner [9] - LBDR	1546	258	83	100	48

general structure of the assertions that have been provided by the assertion miners are similar to the assertions in Listings 3, 4, and 5. According to Table I, IMMizer has reduced the assertions of A-Team [10] by 38%, it has significantly reduced the generated assertions in [9] by 93% for Arbiter and by 83% for LBDR. This figure for assertions provided by [8] is 4%.

It should be noted that the percentage of reduction shows the amount of redundancy in the assertions generated by each assertion miner. In other words, IMMizer is able to check the redundancy and find the redundant assertions. 4% reduction in assertions of the work in [8] is because of the fact that assertions of this tool are typically unique, therefore the redundancy of assertions generated by this tool is less than other tools.

Furthermore, according to the details provided in [28] and Table II, a complete set of mutants in terms of converting all operators and bits has been used and injected into the RTL designs. In Table II, the details about these injected mutants have been presented. The column *'Mutation operator types'* demonstrates different types of the operators that have been modified by the mutants. Any of the operators in each row of the column *'List of operators'*, has been changed to others. In addition to these mutants, we randomly converted all 0 bits to 1 and vice versa.

According to the mutant analysis results that have been presented in Table I, there is no reduction in the number of detected mutants. This is shown in column (*Mutant detection (%)*) where mutant detection is equal to 100 percent for all the benchmarks. This indicates that all the mutants that are detected by the initial assertion set, are covered by the minimized assertion set, correspondingly. This is an important advantage of IMMizer that can detect the mutants with the same coverage as the initial assertion sets, but with a less number of assertions and with less overhead imposed on the system. Moreover, regarding the execution time for minimizing assertions, according to Table I, IMMizer minimizes assertions of A-Team [10] in 15 seconds. This figure for DDG [8] is 21 seconds and for assertions of [9] is 60 seconds for Arbiter and 48 seconds for LBDR.

Table II: Description of Injected Mutants

Mutation Operator Types	List of Operators
arithmetic operators	+, -, *, /, %
relational operators	==, !=, >, <, >=, <=
logical operators	&&, ||
assignment operators	+=, -=, *=, /=, %=, =
unary operators	+, -, ~,!
bitwise operators	<<,>>,&,|,^
bitwise assignment operators	<<=,>>=,&=,|=,^=

In addition to the ability of IMMizer in reducing the number of generated assertions, in some cases, it can improve the readability of assertions by removing extra propositions from the antecedent of assertions. This can be seen in the illustrative example in Fig. 3 that in the minimized assertion, *B, !B, C,* and *!C* have been removed. According to our experiments, for A-Team [10], in 10% of assertions, the number of propositions has been reduced. This figure did not change for DDG [8], while, in assertion miner of [9], a reduction of 73% and 24% has occurred for LBDR and Arbiter, respectively. These results have been presented in the '*Proposition Reduction (%)*' column of Table III.

Furthermore, a reduction in the number of assertions and propositions leads to a reduction in the allocated memory for these assertions. Table III presents this result. In the columns *'Initial Assertions'* and *'Minimized Assertions'*, the amount of memories that have been allocated to the initial and minimized assertions have been reported. The column *'Memory Reduction (%)'* presents the reduction in the allocated memory for minimized assertion sets. For the assertions related to A-Team [10], about 33% memory reduction has occurred after minimization. This figure for DDG [8], is 13.85%. For the assertion miner in [9] 87.24% memory reduction for LBDR and 87.66% for Arbiter have been reported.

To investigate the efficiency of our method in comparison with other approaches, IMMizer has been compared with a tool called Shayan [17]. It should be noted that Shayan ranks all the assertions according to several data-mining metrics, consequently, it selects the top-ranked assertions for verification engineers. Therefore, to make a better comparison, the same number of top-ranked assertions from Shayan equal to the number of minimized assertions by IMMizer have been considered (*e.g.* for A-Team - Arbiter, in the first row in Table I, the number of minimized assertions is equal to 198, thus the first 198 top-ranked assertions by Shayan are considered for the comparison).

Table IV, reports these results. The number of assertions has been reported in column *'#Assertions'* and the efficiency of the method over Shayan has been presented in the column *'Improvement in Mutant Detection (%)'*. For A-Team [10], the top 198 assertions that have been ranked by Shayan have been compared with 198 minimized assertions by IMMizer. After performing mutant analysis, both could detect the same number of injected mutants. Regarding the DDG [8], IMMizer detected the injected mutants 5% more than Shayan. Moreover, for assertion miner proposed in [9], IMMizer could detect 25% and 45% more mutants in comparison with Shayan for LBDR and Arbiter, respectively. The results prove that while IMMizer can minimize the number of initial assertions, it can also detect more mutants in comparison with other tools like Shayan.

Table III: Reduction in the Allocated Memory and Propositions

Assertion Miners	Initial Assertions	Minimized Assertions	Memory Reduction (%)	Proposition Reduction (%)
A-Team [10] - Arbiter	45.28 KB	30.32 KB	33.03	10
DDG [8] - Arbiter	31.76 KB	27.36 KB	13.85	without change
Assertion Miner [9] - Arbiter	33.24 MB	4.1 MB	87.66	24
Assertion Miner [9] - LBDR	31.36 MB	4 MB	87.24	73

Table IV: Efficiency of IMMizer over Shayan

Assertion Miners	#Assertions	Improvement in Mutant Detection (%)
A-Team [10] - Arbiter	198	equivalent
DDG [8] - Arbiter	199	5
Assertion Miner [9] - Arbiter	258	45
Assertion Miner [9] - LBDR	304	25

VI. Conclusions

This paper proposed IMMizer, a cost-effective method for minimizing assertion sets. IMMizer considers the *Contradictory Terms* among assertions for minimization purpose. The experimental results prove the significant ability of the method in minimizing the assertions extracted by different assertion miners without any reduction in design behavior coverage in a very short execution time. Moreover, experimental results showed that in comparison with other approaches for minimizing assertions like data-mining-based approaches, IMMizer is remarkably more accurate in mutant detection with a less number of assertions. In addition, IMMizer can improve the readability of assertions by removing unnecessary propositions, subsequently, it can reduce the memory overhead imposed on the system.

Acknowledgement

This work in the project "ICT programme" was supported by the European Union through European Social Fund.

References

[1] J. C. Knight, "Safety critical systems: Challenges and directions," in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 547–550.

[2] M. R. H. Iman and P. Yaghmaie, "A software control flow checking technique in multi-core processors," *International Journal of Embedded Systems*, vol. 13, no. 2, pp. 136–147, 2020.

[3] M. Jenihhin, X. Lai, T. Ghasempouri, and J. Raik, "Towards multidimensional verification: Where functional meets non-functional," in *2018 IEEE Nordic Circuits and Systems Conference (NORCAS)*, 2018, pp. 1–7.

[4] X. Lai, A. Balakrishnan, T. Lange, M. Jenihhin, T. Ghasempouri, J. Raik, and D. Alexandrescu, "Understanding multidimensional verification: Where functional meets non-functional," *Microprocessors and Microsystems*, vol. 71, p. 102867, 2019.

[5] I. Tuzov, D. de Andrés, and J.-C. Ruiz, "Davos: Eda toolkit for dependability assessment, verification, optimisation and selection of hardware models," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018, pp. 322–329.

[6] A. Danese, F. Filini, T. Ghasempouri, and G. Pravadelli, "Automatic generation and qualification of assertions on control signals: A time window-based approach," in *VLSI-SoC: Design for Reliability, Security, and Low Power*. Springer International Publishing, 2016, pp. 193–221.

[7] M. Boulé and Z. Zilic, *Generating Hardware Assertion Checkers: For Hardware Verification, Emulation, Post-Fabrication Debugging and On-Line Monitoring*. Springer Publishing Company, Incorporated, 2008.

[8] J. Malburg, T. Flenker, and G. Fey, "Property mining using dynamic dependency graphs," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 244–250.

[9] G. Martino and G. Fey, "Syntax-guided enumeration of temporal properties," in *2019 Forum for Specification and Design Languages (FDL)*, 2019, pp. 1–8.

[10] A. Danese, N. D. Riva, and G. Pravadelli, "A-team: Automatic template-based assertion miner," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[11] S. Vasudevan, D. Sheridan, S. Patel, D. Tcheng, B. Tuohy, and D. Johnson, "Goldmine: Automatic assertion generation using data mining and static analysis," in *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, 2010, pp. 626–629.

[12] M. R. Heidari Iman, J. Raik, M. Jenihhin, G. Jervan, and T. Ghasempouri, "A methodology for automated mining of compact and accurate assertion sets," in *2021 IEEE Nordic Circuits and Systems Conference (NorCAS)*, 2021, pp. 1–7.

[13] D. Pal, S. Offenberger, and S. Vasudevan, "Assertion ranking using rtl source code analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 8, pp. 1711–1724, 2020.

[14] S. Hertz, D. Sheridan, and S. Vasudevan, "Mining hardware assertion with guidance from static analysis," *IEEE Trans. on CAD*, vol. 32, 2013.

[15] M. Bertasi, G. Di Guglielmo, and G. Pravadelli, "Automatic generation of compact formal properties for effective error detection," in *Proc. of ACM/IEEE CODES+ISSS*, 2013, pp. 1–10.

[16] G. Fey, T. Ghasempouri, S. Jacobs, G. Martino, J. Raik, and H. Riener, "Design understanding: From logic to specification*," in *2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2018, pp. 172–175.

[17] T. Ghasempouri and G. Pravadelli, "On the estimation of assertion interestingness," in *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2015, pp. 325–330.

[18] T. Ghasempouri, S. Payandeh Azad, B. Niazmand, and J. Raik, "An automatic approach to evaluate assertions' quality based on data-mining metrics," in *2018 IEEE International Test Conference in Asia (ITC-Asia)*, 2018, pp. 61–66.

[19] R. Hariharan, T. Ghasempouri, B. Niazmand, and J. Raik, "From rtl liveness assertions to cost-effective hardware checkers," in *2018 Conference on Design of Circuits and Integrated Systems (DCIS)*. IEEE, 2018, pp. 1–6.

[20] J. S. Bradbury, J. R. Cordy, and J. Dingel, "Exman: A generic and customizable framework for experimental mutation analysis," in *Second Workshop on Mutation Analysis*, 2006, pp. 4–4.

[21] Y. Zhang and A. Mesbah, "Assertions are strongly correlated with test suite effectiveness," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 214–224.

[22] T. Xie, N. Tillmann, J. de Halleux, and W. Schulte, "Mutation analysis of parameterized unit tests," in *2009 International Conference on Software Testing, Verification, and Validation Workshops*, 2009, pp. 177–181.

[23] C. B. Spear, *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*, 2nd ed. Springer Publishing Company, Incorporated, 2010.

[24] S. L. Harris and D. M. Harris, "2 - combinational logic design," in *Digital Design and Computer Architecture*, S. L. Harris and D. M. Harris, Eds. Boston: Morgan Kaufmann, 2016, pp. 54–106.

[25] T. Ghasempouri, A. Danese, G. Pravadelli, N. Bombieri, and J. Raik, "Rtl assertion mining with automated rtl-to-tlm abstraction," in *2019 Forum for Specification and Design Languages (FDL)*, 2019, pp. 1–8.

[26] C. Deutschbein and C. Sturton, "Mining security critical linear temporal logic specifications for processors," in *2018 19th International Workshop on Microprocessor and SOC Test and Verification (MTV)*, 2018, pp. 18–23.

[27] "Project Bonfire Network-on-Chip," https://github.com/Project-Bonfire/Bonfire_handshake, 2017.

[28] U. Repinski, H. Hantson, M. Jenihhin, J. Raik, R. Ubar, G. Di Guglielmo, G. Pravadelli *et al.*, "Combining dynamic slicing and mutation operators for esl correction," in *2012 17th IEEE European Test Symposium (ETS)*, 2012, pp. 1–6.

Appendix 4

IV

A. Roberts and MRH. Iman, M. Bellone, T. Ghasempouri, J. Raik, O. Maennel, M. Hamad, and S. Steinhorst, "ADAssure: Debugging Methodology for Autonomous Driving Control Algorithms," *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Valencia, Spain, 2024, pp. 1-6.

2024 Design, Automation & Test in Europe Conference (DATE 2024)
Special Initiative "Autonomous Systems Design"

ADAssure: Debugging Methodology for Autonomous Driving Control Algorithms

Andrew Roberts[◇*], Mohammad Reza Heidari Iman[◇†], Mauro Bellone[*], Tara Ghasempouri[†], Jaan Raik[†], Olaf Maennel[‡], Mohammad Hamad[§], Sebastian Steinhorst[§]
[*] FinEst Centre for Smart Cities, Tallinn University of Technology
[†] Department of Computer Systems, Tallinn University of Technology
[‡]School of Computer and Mathematical Sciences, The University of Adelaide
[§]Department of Computer Engineering, Technical University of Munich

***Abstract*—Autonomous driving (AD) system designers need methods to efficiently debug vulnerabilities found in control algorithms. Existing methods lack alignment to the requirements of AD control designers to provide an analysis of the parameters of the AD system and how they are affected by cyber-attacks. We introduce ADAssure, a methodology for debugging AD control system algorithms that incorporates automated mechanisms which support generation of assertions to guide the AD system designer to identify vulnerabilities in the system. Our evaluation of ADAssure on a real-world AD vehicular system using diverse cyber-attacks developed a set of assertions that identified weaknesses in the OpenPlanner 2.5 AD planning algorithm and its constituent planning functions. Working with an AD control system designer and safety validation engineer, the results of ADAssure identified remediation of the AD control system, which can support the implementation of a redundant observer for data integrity checking and improvements to the planning algorithm. The adoption of ADAssure improves autonomous system design by providing a systematic approach to enhance safety and reliability through the identification and mitigation of vulnerabilities from corner cases.**

***Index Terms*—Security, Autonomous Driving**

I. Introduction

Autonomous driving (AD) vehicles are increasingly being utilised for transportation on public roads. Waymo and Cruise offer AD ride-hailing services in San Francisco, Apollo Baidu in China, and numerous such services are operating in Europe. Central to the wider-adoption of AD vehicles on public roads is the security and safety of their control algorithms that enable self-driving technology. AD control algorithms comprise a complex code-base of interconnected modules that perform tasks and sub-tasks that enable a vehicle to sense, perceive, localise, and navigate in a driving environment. With the increase in diversity of AD use-cases from valet parking to public transportation in public traffic, the code base of AD control algorithms will reputedly grow from 100-200 million to billions of lines of code [1].

Within this complex environment, debugging the code for logical errors arising from unexpected control behaviour is a fundamental challenge [2]. AD system designers need to pinpoint where in the control software weaknesses are, in order to focus debugging efforts in an efficient manner. Existing studies attempt to rectify unexpected AD control behaviour at run-time through smoothing trajectories utilising neural networks [3] [4] [5]. The applicability of these studies in real-world AD programs are limited due to the highly dynamic environment of autonomous driving and the probabilistic nature of the algorithms for planning.

◇ These authors contributed equally to this work.

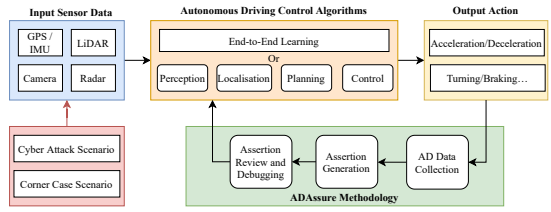


Fig. 1: Comprehensive ADAssure methodology overview that illustrates each step of the process, from data collection to assertion creation, review of assertions, and debugging.

Furthermore, in these studies, the analysis lacks the expertise from the algorithm designer and safety engineer to inform on the nature of the behaviour of vehicle dynamics, whether noise identified as irregular could be considered for a control engineer within normal constraints, whether AD behaviour could be considered a legitimate safety response to an unexpected event and whether the parameters for which the run-time solution is designed are appropriate for differing class of vehicles with different dynamic profiles. We consider the design phase to offer the most promising area of initial investigation to improve the robustness of control algorithms, which can be translated to real-world AD systems.

In this work, we propose ADAssure, a methodology for debugging control algorithms during the design-time phase of AD control software development (Fig. 1). ADAssure is built upon the idea that the data of vehicle dynamics and sensing of AD systems can be analysed for anomalous control behaviour, which can then be transformed into assertions on the AD control. We use association rules that enable us to mine datasets of varying scales and fingerprint the pattern of anomalous activity. These rules can be used to guide AD system designers to focus on the debugging of the control algorithms. To evaluate ADAssure, we focus on a control system algorithm used in a real-world AD vehicular system providing ride-hailing services. To summarise, the paper makes the following contributions:

- We propose ADAssure, a methodology designed for debugging AD control algorithms during the autonomous system development's design phase (see § II).
- To demonstrate our methodology's feasibility, we applied it to diverse datasets, revealing three new vulnerabilities in `Openplanner 2.5` AD, used in real-world vehicles. (see § IV).
- We provide to the community our artifacts to enable reproducibility and assist with developing efforts to im-

979-8-3503-4859-0/DATE24/© 2024 EDAA

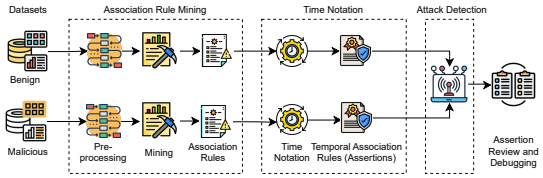


Fig. 2: Phases for Assertion Generation

prove AD control system design. These artifacts include simulation datasets and real-world AD system data (*ADAssure Datasets*).

II. ADAssure: Methodology

The development of ADAssure has three main motivations. First, it aims to provide AD system designers with a methodology to identify and fix vulnerabilities that align with the design of AD algorithms. Second, given the ever-changing nature of the autonomous vehicle system, it strives to establish a structured methodology that allows for consistent, flexible, and repeatable testing. Third, it aims to support unit testing, allowing testing of individual components of the autonomous system in isolation from other dynamic factors affecting autonomous control.

The foundations of the ADAssure methodology are based on the analysis of the vehicle dynamics and sensing data to guide the creation of assertions of the vulnerability of the AD control algorithms. The analysis consists of a sensitivity analysis of vehicle dynamics data (e.g., velocity, yaw, and steering angle), sensor data (e.g., lateral and longitudinal movement), and visualisation of the trajectory of the AD system. This helps identify key parameters to build assertions of the AD control algorithms. The AD control system designers can use the assertions to identify and locate the vulnerabilities of the control model and develop mechanisms to test and fix the errors. The ADAssure methodology comprises three main phases: AD Data Collection, Association Rule Generation, and Assertion Review and Debugging. Next, we will explore each phase in more depth.

A. Autonomous Driving Data Collection

This phase consists of generating data from the real-world system or simulation environment. The benefit of a simulation environment is that driving scenarios can be automated or designed to test a specific condition, such as a cyber-attack or a corner case. The data output is structured according to established metrics. These can be vehicle dynamics parameters (yaw angle, velocity, etc.), sensing data (position co-variance, point-cloud, etc.), and safety parameters (distance-to-collision, etc.). The AD data is outputted in a format that can be interpreted by analytical tools, in our use-case, .csv format.

B. Association Rule Generation Phase

The goal of this phase is to process the data generated from the previous phase and produce a set of association rules that can be translated into assertions in the Assertion Review and Debugging phase. This phase is comprised of three primary steps (as shown in Fig. 2): a) Association Rule Mining, b) Time Notation, and c) Attack Detection. The association rule mining is applied to both benign and malicious datasets, resulting in two distinct sets of association rules.

Algorithm 1: Association rule mining & time notation

1 **Input:** $\mathcal{N}$, $\mathcal{D}$
2 **Output:** $next[\mathcal{N}] = antecedent \rightarrow next[\mathcal{N}]consequent$, $before[\mathcal{N}] = antecedent \rightarrow before[\mathcal{N}]consequent$
/* Initialization and Preprocessing */
3 $\mathcal{R}$ = $antecedent \rightarrow consequent$
4 **forall** $f \in \mathcal{D}$ **do**
5 $\quad \mathcal{D}' = \textbf{MoveUp}(f(\mathcal{N}))$
/* Mining */
6 $\mathcal{R} \leftarrow \textbf{apriori}(\mathcal{D}')$
/* Time Notation */
7 **if** $(\mathcal{R}.antecedent == (t \in \mathcal{D}'))$ *and* $(\mathcal{R}.consequent == (f \in \mathcal{D}'))$ **then**
8 $\quad next[\mathcal{N}] \leftarrow \textbf{label}(\mathcal{R})$
9 **if** $(\mathcal{R}.antecedent == (f \in \mathcal{D}'))$ *and* $(\mathcal{R}.consequent == (t \in \mathcal{D}'))$ **then**
10 $\quad before[\mathcal{N}] \leftarrow \textbf{label}(\mathcal{R})$

These rules are then processed through the Time Notation step to incorporate temporal information, yielding temporal association rules (assertions) in the form of $next[\mathcal{N}]$ and $before[\mathcal{N}]$ patterns. We define $next[\mathcal{N}]$ type of rule in the general form of $\mathcal{X} \rightarrow next[\mathcal{N}]\mathcal{Y}$. This rule indicates that when $\mathcal{X}$ occurs, after $\mathcal{N}$ time instants, $\mathcal{Y}$ will occur. $\mathcal{N}$ is a positive integer value. Moreover, we define $before[\mathcal{N}]$ rule in the general form of $\mathcal{X} \rightarrow before[\mathcal{N}]\mathcal{Y}$. This rule demonstrates that whenever $\mathcal{X}$ happens, $\mathcal{Y}$ should have occurred $\mathcal{N}$ time instants before that. The "Attack Detection" step compares these temporal association rules, ultimately detecting attacks and anomalies within the datasets. Subsequent sections provide a more in-depth discussion of each step.

a) Association Rule Mining: This step primarily serves two objectives: pre-processing the datasets and subsequently mining association rules from the preprocessed data. To mine the association rules, apriori algorithm [6] was adopted and enhanced to mine temporal rules capable of detecting attacks at various time instances during autonomous vehicle (AV) operation. Algorithm 1 presents the details of the Association Rule Mining and Time Notation steps. In this algorithm, $\mathcal{D}$ denotes the dataset and $\mathcal{D}'$ is the preprocessed dataset, while f and t represent the dataset's features and target values. To prepare the dataset for mining the $next[\mathcal{N}]$ and $before[\mathcal{N}]$ temporal patterns, all the features of the dataset are moved $\mathcal{N}$ records above its original position (Line 5). However, the target of the dataset remains as it is. Afterwards, the apriori algorithm is applied to the preprocessed dataset to mine a set of association rules. The output of this phase is a set of association rules in the general form of $antecedent \rightarrow consequent$ that are ready to be forwarded to the Time Notation step.

b) Time Notation: In this step, the method integrates the concept of time into the association rules generated in the association rule mining step, leading to a set of temporal association rules. The method determines to which temporal pattern ($next[\mathcal{N}]$ or $before[\mathcal{N}]$) each extracted rule belongs and subsequently assigns the corresponding time label to the rule. If the antecedent value matches a target value in the dataset, and the consequent value has already been moved to another record in the dataset, the rule is labelled as a $next$ temporal association rule (Line 8). Otherwise, if the antecedent of a rule mined in the association rule mining step matches a dataset feature that has already been moved

to another record and the consequent of the rule matches the target value of the dataset, we label this rule as a $before$ temporal association rule (Line 10). The mined rules are in the forms of $antecedent \rightarrow next[\mathcal{N}]consequent$, and $antecedent \rightarrow before[\mathcal{N}]consequent$, serving as assertions for debugging the AD system.

c) Attack Detection: This step aims to identify rules indicating attacks on the AV. We assume that the sets of mined rules from the benign and malicious datasets should be similar under normal conditions, without any AV attacks. Any deviation between these rule sets signifies an anomaly in the autonomous vehicle. Per this assumption, the temporal association rules (assertions) mined during the time notation phase are classified into two sets. The first category comprises rules exclusively mined from the malicious dataset, lacking counterparts in the benign dataset. Any rule extracted solely from the malicious dataset, without a corresponding counterpart in the benign dataset, signifies an attack. These rules reveal abnormal behaviour in the malicious dataset, contrasting with different behaviour observed in the corresponding time instance of the benign dataset. Consequently, we classify these as attacks. The second category comprises similar rules mined from both benign and malicious datasets, but with different *minimum support* (`min_supp`) and *minimum confidence* (`min_conf`) values. The variations in these values indicate that, while the mined rules are similar, abnormal behaviours and anomalies exist between the datasets. The apriori algorithm employs these two metrics (i.e., `min_supp` and `min_conf`). The `min_supp` value is the threshold and a minimum value that is chosen by the expert to decide whether a rule occurs frequently in the dataset or not [7], [8]. The `min_conf` is the minimum value that is chosen by the expert and is an indication of how often a rule has been found to be true [6], [9]. Increasing the `min_supp` value results in fewer association rules that describe more general behaviour of the autonomous vehicle, while decreasing the `min_supp` value leads to rules covering rare behaviours (corner cases). Similarly, raising the `min_conf` value produces fewer but more valid rules. Valid rules refer to association rules that will not be violated with different attack scenarios like corner cases. These values in the ADAssure facilitate an effective attack detection process. The second category of rules aids the ADAssure in effectively identifying corner cases and the attacks that rarely occur on the AV. These rare attacks exhibit behaviour very similar to normal vehicle operation but are malicious and can lead to AV failure.

C. Assertion Review and Debugging

Within this phase, the association rules generated from the association rule mining are reviewed in conjunction with an analysis of the control behaviour and individual data parameters to develop assertions. Trajectory maps of the AD system and graphs, which demonstrate the sensitivity of the data parameters during benign and cyber-attack scenarios, are compared to the anomalous behavioural patterns detected by the association rule mining tool. Using expertise from the algorithm designer and safety validation engineer assists in understanding which parameters can uniquely demonstrate a vulnerability of an algorithm within the system. From developing an assertion on the system's vulnerability, the debugging effort focuses on a control flow analysis. As the assertion assists in pinpointing the specific module, the static analysis can focus on the control flow of the substituent functions within the module. As an example of the importance of this pinpointing, a local-planning module could have 15 diverse algorithms, and within these, each could have multiple different methods or functions. As the code of AD algorithms are differential equations, debugging can suggest optimisations that enable mitigation mechanisms against the identified vulnerabilities.

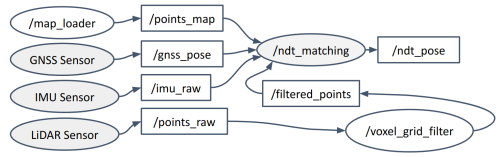


Fig. 3: Localisation Algorithm Flow within AD System.

III. Autonomous Driving Control Algorithm

To evaluate the methodology, we focus on an AD control algorithm used in a real-world AD ride-hailing service. Within the AD pipeline, there are four key modules: localisation, perception, planning, and control. Within our study, we focus on the localisation and planning modules.

A. Localisation Module

This module provides accurate information regarding the position and orientation of the vehicle. Using a Normal Distributions Transform (NDT) matching search algorithm, it identifies the best matching position based on sensor perception. It uses input from the Inertial measurement unit (IMU) and the point cloud generated by the LiDAR. Then, it attempts to match the points from our current scan to a grid of probability functions derived from the map. NDT matching algorithms can also benefit from the GNSS sensor, which provides initial rough estimates of localization on geo-referenced maps, thereby avoiding any sudden errors in localization calculations that may result in failures. Fig. 3 displays the flow of the localisation algorithm within the AD system.

B. Planning Module

For the AD system to plan a mission, firstly, a global planner generates a global reference path using a vector (road network) map. The function of the global planner is to stipulate a route between the starting position and goal position of the mission on the road map. The local-planner generates smooth and obstacle-free trajectories in the operational local domain following the global route. The local-planner consists of several modules (see Fig. 4); trajectory generation, trajectory evaluation, intention and trajectory estimator, object-tracker and behavior selection (decision making) [10]. The trajectory generation module generates alternative tracks parallel to the main path defined by the global planner. These tracks are named roll-outs. The trajectory evaluation module assesses all possible roll-outs and the data input from sensed-data of the AV and makes a cost estimation. The behaviour selector will lead the AV to motion on a roll-out based on the least-cost.

IV. Experimentation and Results

To evaluate the impact of corner cases on AD system behaviour using the ADAssure methodology, we use datasets of corner cases from simulation and real-world driving from

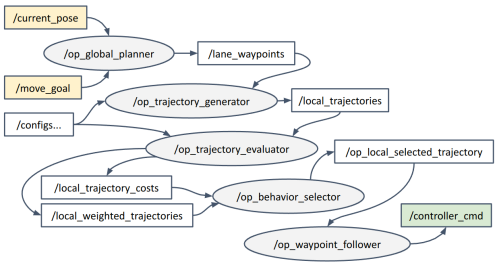


Fig. 4: Abstract Local Planning Algorithm Flow within AD System.

the target AD system. The 1^{st} corner case scenario dataset is of three diverse cyber-security attacks on the AD system conducted in a simulation environment. As our focus is the planning and localisation algorithms, we used a low-fidelity simulation provided by `Autoware.AI` and the `OpenPlanner 2.5` planning algorithm. The 2nd corner case scenario dataset is of a Global Positioning System (GPS) spoofing event that occurred on the AD system during its operation on the roads of a capital city.

A. AD Control System Datasets

a) Cyber-security Corner Case Dataset: Within this dataset, three attacks were conducted on the target AD vehicular system, which is attempting an overtaking manoeuvre. The three attacks are classified as: 1) *Lateral Position Offset Attack* 2) *Longitudinal Position Offset Attack* 3) *Message Time-Delay*. In the lateral and longitudinal position offset attack, an attacker injects malicious data input into the lateral or longitudinal pose whilst the AD vehicular system is in the process of the overtaking manoeuvre (Fig. 5). This attack could be conducted through GPS spoofing or interception and manipulation of the localisation sensor data. The attacker introduces a delay into the `current_pose` (lateral and longitudinal) sensor messages reaching the AD control pipeline for the message time-delay. The malicious data is injected at around the $21\,\mathrm{m}$ mark of the AV journey (travelled distanced) to the $67\,\mathrm{m}$. Each attack was conducted 300 times, accommodating a variation of different attack parameters. The lateral and longitudinal attacks introduced a deviation ranging from $0.16\,\%$ to $1.0\,\%$, which equates to around $20\,\mathrm{cm}$ to $1\,\mathrm{m}$. The message time-delay introduced delays of $0.3\,\%$, $0.6\,\%$, $1.0\,\%$ second, as a message is transmitted every $20\,\mathrm{ms}$, this range represents a delay of 15 to 50 messages. In total, the dataset comprises over 1500 scenario runs of attacks and benign safety cases.

b) GPS Spoofing Real-World AV Dataset: The AD ride-hailing service transmits its sensor data via a logging node to an edge server, which stores the AD System data in a database.

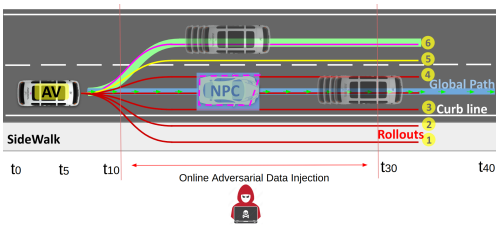


Fig. 5: The threat model used for conducting the attack cases.

TABLE I: AD System Data.

AD Data Type	Description
AV_X	Longitudinal Position of the AD System as to the HD Map
AV_Y	Lateral Position of the AD System as to the HD Map
AV_Steer	Steering Angle of the AD System
AV_Vel	Velocity of the AD System
AV_Yaw	Orientation of the AD System based on its centre of gravity
Roll-out_Num	Current Lane according to the lane selector of the AD Control Algorithm
DTC	Distance to collision of the AD vehicular system to the overtaking vehicle.
Position Co-variance	GPS position co-variance
Altitude	Altitude derived from the GPS

During its operations near the port area of the city, the AD vehicle encountered a loss of localisation from a GPS spoofing event which also affected other GPS-enabled platforms. This GPS spoofing continued intermittently throughout the preceding months. The dataset used in this study is from the logging system of AD ride-hailing service.

c) AD System Data: The simulation and real-world datasets were structured to output data as shown in Table I.

B. Experimental Results

To evaluate the ADAssure methodology, we chose six attack types and their corresponding safety (benign) scenarios. These attack types included each of the aforementioned attacks with differing levels of noise (lateral and longitudinal position offset, delay message).

a) Automated Analysis: Utilising the ADAssure methodology on the three types of attacks yields three distinct set of assertions corresponding to each attack type. The results of the assertion generation phase are presented in Table II. The threshold for minimum support (`min_supp`) is set at 0.01, while the minimum confidence (`min_conf`) threshold is 1 Notably, the method exhibits a swift execution time. Within the 3 attacks of the cybersecurity corner case dataset, the assertions identify two patterns of anomalous AD behaviour. Firstly, extreme steering angles of $20°$ and $-20°$ and sudden lane transition. Secondly, multiple lane-transitions combined with the extreme steering angle and sudden changes in vehicular velocity. This behaviour can be seen to be the effect of cyber activity on the smoothness of the initiation of the overtaking manoeuvre which results in turbulent movements and in some cases, a collision event. The assertions generated from the GNSS spoofing dataset identified the changes to the altitude and position co-variance. These were consistent with dramatic change in the values of the GPS coordinates and the resultant change in altitude.

b) Assertion Review and Debugging: The patterns identified in the association rules enables us to extrapolate that the Yaw angle and angular velocity are good reference point to show the effect of cyber-attacks. During the injection of the position offset attacks, the vehicle's orientation demonstrates dramatic action; in some circumstances, the vehicle can be

TABLE II: ADAssure Assertion Generation phase results.

Dataset		Assertion			Execution Time
Name	#Records	Total	$\#Next[\mathcal{N}]$	$\#Before[\mathcal{N}]$	
Longitude	412	5	3	2	1 ns
Latitude	356	7	7	0	1 ns
Delay	417	5	3	2	1 ns
GNSS	16	5	4	1	1 ns

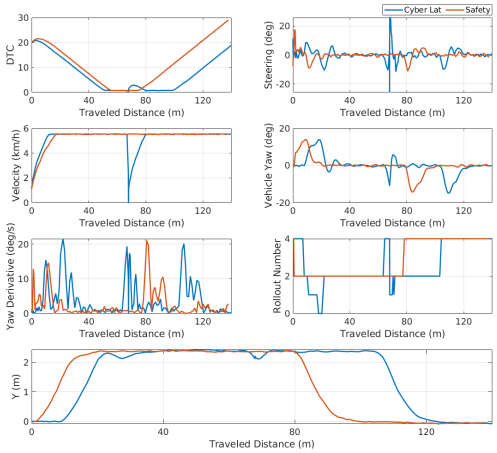


Fig. 6: Lateral position offset attack vehicle parameters.

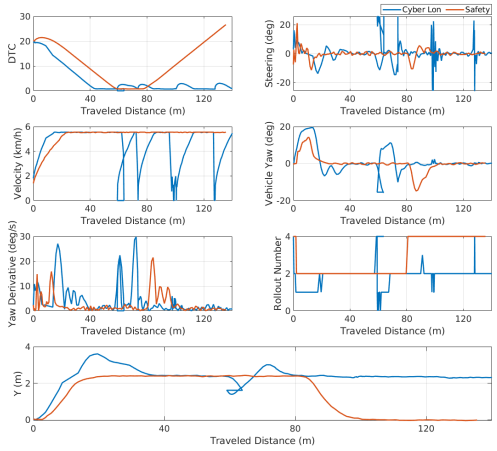


Fig. 7: Longitudinal position offset attack vehicle parameters.

seen to be essentially spinning. As displayed in Fig. 6, the Lateral Position Offset Attack displays the Yaw (angle) of the vehicle making sharp changes, of 15 deg/sec from 15 meters mark of the AV journey. This vehicle dynamic behaviour is a characteristic also seen in both the longitudinal position offset (Fig. 7) and delay message attack (Fig. 8). The results for the velocity parameter demonstrate that it only indicates immediate collision of the vehicle, and it does not support early identification of anomalous vehicle behaviour. Assertion 1 contends that the AD system should not allow movements that challenge the physical limitations of the steering model.

> **Assertion 1:** To determine the vulnerability of the yaw angle and momentum, we can derive the assertion: *AV.displacement_of_yaw_angle* $>$ *max_yaw_angle_threshold && time* $<$ *time_threshold.*

The roll-out transition, steer, and distance-to-collision parameters demonstrate identifiable change during a cyber-attack. The manipulation of the lateral and longitudinal position alters the vehicle position on the map and, therefore, has the effect of inducing greater transitions between roll-outs, which is the effective position of the vehicle on the road. The frequency of transition impacts the smoothness of the steering angle. From the distance-to-collision parameter, it is noted that the effect of the attack is most prominent during the overtaking maneuver and mostly during the cut-in process, when the vehicle cuts-in front of the passing vehicle (NPC). Assertion 2 contends that when the vehicle transitions across multiple roll-outs and displays $180°$ steering and closes to less than $0.5\,\mathrm{m}$ to the passing vehicle, this represents affected behaviour from the cyber attack.

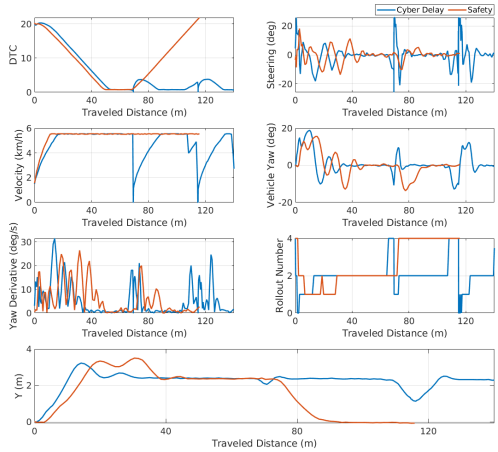


Fig. 8: Delay message attack vehicle parameters.

> **Assertion 2:** To identify vehicle dynamic changes from cyber-attack: *AV.x* $-$ *NPC.x* $<$ *distance_threshold && AV.lane_transition* $>$ *max_transition_number && AV.steer_angle* $\notin$ *[min, max]_steer_angle*

Assertion 3 contends with activity seen in the longitudinal position offset (Fig. 7) where the AV collides with the passing vehicle and then accelerates to the previous set-point.

> **Assertion 3:** To identify collisions we can derive the assertion: $|AV.v_k - AV.v_{k+1}| > threshold.$

Assertion 3 could also be used to detect anomalies in GPS data. The GNSS spoofing attack demonstrates a significant deviation in the altitude and position co-variance parameters. Assuming that velocity data comes from two sources, a wheel sensor measurement and calculated by deriving the position from GPS data, the two results should be close to each other. In the case of a GNSS spoofing attack, the deviation in the position co-variance would generate a spike in the velocity (calculated by deriving the position in GPS data), and thus violating assertion 3.

For our specific AD system, the threshold for assertion 1 is $15°$ yaw angle displacement within $1\,\mathrm{s}$ duration. Assertion 2 threshold is identified as a distance between AV and passing

vehicle as less than $0.5\,\mathrm{m}$, lane transition greater than 1 roll-out and steering angle that is outside the bounds of 20 and $-20°$. It is important to note that these values are valid for a low-speed AV ride-hailing service and for designers of different classes of vehicles, it is required to calculate values consistent with their specific application.

Solvable bugs come from several points in the controller; a simple one is wrong or imprecise saturation values of the control signal, which generates a high acceleration or a high steering angle in the vehicle. This is clearly visible in Fig. 7 where a signal overshoot causes the vehicle to change lane multiple times. Another example, clearly visible in Fig. 6,7& 8 is the lack of a fallback plan. There is a clear indication of a collision as the vehicle speed suddenly drops to $0\,\mathrm{m\,s^{-1}}$ and then quickly accelerates to the reference point, this is a violation of Assertion 3. A robust controller should have a fallback plan for such a case which indicates a bug in the functional design of the controller. In such a case, the vehicle should be aware of the fact that the global trajectory cannot be followed anymore and switch to emergency mode.

The main reason for searching for unexpected behaviours is to debug the controller, with reference to the experimental results, a violation of Assertion 1 can be associated to a bug in the `/ndt_pose` module (see Fig. 3), while a violation of Assertion 2 can be back-propagated to the module `/op_trajectory_evaluator`. A violation of assertion three can be backpropagated to the modules of `/op_trajectory_generator` and `/op_behaviour_selector` (see Fig. 3). To pinpoint the violation of assertion 3 to a specific function, we abstracted from the local_planner algorithm and its substituent lane_rule algorithm, the `getClosestWaypointNumber` method, which selects the next waypoint to follow in the global trajectory and returned an exception to be handled as a different driving behaviour (e.g., there was a crash, emergency mode activated).

In the case of GNSS attack, the NDT localisation algorithm doesn't detect the deviation in position co-variance, and this is due to the normal vector pointing in the same direction. Debugging focuses on optimisation of the NDT localisation using visual odometry for holding the local position at short-distances until the source of the disturbance has been resolved.

V. Related Work

Recent publications on anomaly detection in vehicular AD control systems propose the usage of vehicle dynamics as a key detection indicator for cyber-attacks [11] [12] [13]. Studies such as Guo et al. [14] emphasise the effect cyber-attacks have on the trajectory of the AD system and the noise of individual sensors. Mitigation mechanisms focus on two diverse approaches 1) implementation of an observer of AD vehicle state estimation which can inform an emergency action (sensor switching etc.) [14] 2) implementation of trajectory smoothing algorithm to correct unplanned vehicle behaviour [12] [13]. However, these solutions for detection and mitigation are developed based on assumptions of driving environment and algorithm configuration and this limits the scope of their applicability.

VI. Conclusion

Cyber-attacks present new challenges to the design of AD algorithms. Designers need methods to debug vulnerabilities to improve robustness. In this paper, we introduced ADAssure, a methodology for debugging AD algorithms during the design phase. The methodology consisted of three phases; 1) AD Data collection 2) Assertion Rule Generation 3) Assertion Review and Debugging. The concept of the methodology was to develop association rules from mining AD data which can be transformed into assertions on the vulnerability of the system.

Our evaluation of ADAssure on diverse cyber-security datasets from simulation and real-world revealed that the ADAssure method could identify three assertions on the vulnerability of the `OpenPlanner 2.5` AD planning algorithm. These assertions were able to guide an algorithm designer and safety engineer to pinpoint the specific modules in the planning algorithm for debugging.

Acknowledgment

This work has been supported by the European Commission through the European Union's Horizon 2020 Research and Innovation Programme, under grant agreement No 101021727.

References

[1] Bosch, "Facts and figures about electronics and software in vehicles," *Automotive World*, July, 2021.

[2] W. Zeng, M. Wu, P. Chen, Z. Cao, and S. Xie, "Review of shared online hailing and autonomous taxi services," *Transportmetrica B: Transport Dynamics*, vol. 11, no. 1, pp. 486–509, 2023.

[3] K. K.-C. Chang, X. Liu, C.-W. Lin, C. Huang, and Q. Zhu, "A safety-guaranteed framework for neural-network-based planners in connected vehicles under communication disturbance," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023.

[4] R. Jiao, H. Liang, T. Sato, J. Shen, Q. A. Chen, and Q. Zhu, "End-to-end uncertainty-based mitigation of adversarial attacks to automated lane centering," in *2021 IEEE Intelligent Vehicles Symposium (IV)*, 2021.

[5] X. Liu, R. Jiao, B. Zheng, D. Liang, and Q. Zhu, "Safety-driven interactive planning for neural network-based lane changing," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '23. Association for Computing Machinery, 2023.

[6] J. Han, M. Kamber, and J. Pei, "6 - mining frequent patterns, associations, and correlations: Basic concepts and methods," in *Data Mining (Third Edition)*, ser. The Morgan Kaufmann Series in Data Management Systems. Boston: Morgan Kaufmann, 2012, pp. 243–278.

[7] M. Zaki, "Scalable algorithms for association mining," *IEEE Transactions on Knowledge and Data Engineering*, 2000.

[8] M. R. Heidari Iman, J. Raik, M. Jenihhin, G. Jervan, and T. Ghasempouri, "An automated method for mining high-quality assertion sets," *Microprocessors and Microsystems*, vol. 97, p. 104773, 2023.

[9] M. Shahin, M. R. Heidari Iman, M. Kaushik, R. Sharma, T. Ghasempouri, and D. Draheim, "Exploring factors in a crossroad dataset using cluster-based association rule mining," *Procedia Computer Science*, 2022.

[10] H. Darweesh, E. Takeuchi, and K. Takeda, "Openplanner 2.0: The portable open source planner for autonomous driving applications," in *2021 IEEE Intelligent Vehicles Symposium Workshops*, 2021.

[11] Z. Ju, H. Zhang, X. Li, X. Chen, J. Han, and M. Yang, "A survey on attack detection and resilience for connected and automated vehicles: From vehicle dynamics and control perspective," *IEEE Transactions on Intelligent Vehicles*, vol. 7, no. 4, pp. 815–837, 2022.

[12] Y. Ma, J. Sharp, R. Wang, E. Fernandes, and X. Zhu, "Sequential attacks on kalman filter-based forward collision warning systems," in *AAAI Conference on Artificial Intelligence*, 2020.

[13] J. Shen, Y. Luo, Z. Wan, and Q. A. Chen, "Lateral-direction localization attack in high-level autonomous driving: Domain-specific defense opportunity via lane detection," 2023.

[14] J. Guo, L. Li, J. Wang, and K. Li, "Cyber-physical system-based path tracking control of autonomous vehicles under cyber-attacks," *IEEE Transactions on Industrial Informatics*, 2023.

Appendix 5

V

MRH. Iman, S. Ahmadi-Pour, R. Drechsler, and T. Ghasempouri, "Processor Vulnerability Detection with the Aid of Assertions: RISC-V Case Study," *TechRxiv, to be submitted*, 2024, pp. 1-8, DOI: https://doi.org/10.36227/techrxiv.172101134.45466090/v1.

Processor Vulnerability Detection with the Aid of Assertions: RISC-V Case Study

Mohammad Reza Heidari Iman[1], Sallar Ahmadi-Pour[2], Rolf Drechsler[2,3], and Tara Ghasempouri[1]
[1]Department of Computer Systems, Tallinn University of Technology, Tallinn, Estonia
[2]Institute of Computer Science, University of Bremen, Bremen, Germany
[3]Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
{mohammadreza.heidari, tara.ghasempouri}@taltech.ee
{sallar, drechsler}@uni-bremen.de

***Abstract*—As RISC-V processors become more widely distributed, security issues arise in them. To this end, security verification techniques for processors should be incorporated into the design process to ensure the processor specifications, security requirements, and its actual implementation are consistent. Among the verification techniques, assertion-based verification has emerged as one of the most promising techniques. Although assertions are widely used for functional verification, there is limited effort in applying assertions for security verification. Thus, in this work, a novel security assertion-based technique is introduced for verifying the invulnerability of the processors against attacks. The method works in three main steps, first, collecting the security properties related to RISC-V, second, systematically identifying characteristics of instruction set architecture for each security property. Third, with the guide of input from the previous step, a security-based assertion miner is proposed to automatically extract a set of security assertions for the RISC-V. The experiments show that the proposed method can automatically generate security assertions in a very short amount of time and detect all the inserted Hardware Trojans in the processor, thereby accurately verifying the security of the processor.**

***Keywords*—Automatic Security Verification, RISC-V Security Verification, Security Assertion Mining, Data Mining**

I. Introduction

Processors are ubiquitous in our daily lives and are embedded in nearly every electronic device. In the world of processor design, RISC-V is regarded as one of the most promising technologies. RISC-V's Instruction Set Architecture (ISA) is increasingly adopted in open-source and commercial processor designs due to its flexibility in supporting extensions and customization, as well as its open instruction set [1]. While the openness of the RISC-V ISA is advantageous for fostering a large community to examine and extend the ISA continually, it also poses a disadvantage, as attackers can more easily target the architecture [2]. In recent years, several notable processor vulnerabilities, such as the Spectre and Meltdown attacks, have been identified [3]. These attacks exploit security vulnerabilities in processors to leak confidential data.

In this context, an innovative method is required to verify the alignment between the specifications and security requirements of RISC-V, the corresponding security-related ISA, and the practical implementation of the RISC-V extension.

Numerous mature works on functional verification in the design process exist in the literature [4–11], and some of them can be applied to security verification [6, 12, 13]. The fundamental distinction between functional and security verification lies in their objectives: the former addresses functional problems in alignment with functional specifications, while the latter identifies security issues by considering security requirements and threat models [14].

Meanwhile, semi-formal verification techniques like assertion-based verification (ABV) [15] have garnered interest for their scalability. In the realm of ABV, assertions represent logical formulas that describe the design's behavior. They enhance both the design's observability and error localization capabilities [16, 17].

There is limited literature on utilizing assertions for the security verification of processors [18–24]. In [19], security assertions for processor vulnerabilities are manually crafted to validate security requirements against the processor design. However, this manual approach demands significant time and expertise, incurring high costs. The method presented in [20] translates security assertion sets from one design to another, but it relies on manual assertion definition and is only evaluated for the OR1200 processor. The other study presented in [22] proposes a method for manually identifying security-critical properties for use in the security verification of the OR1200 processor. Another study analyzing the security properties of the OR1200 processor is the paper presented in [23]. It employs model checking and formal approaches to detect processor errors that can lead to security vulnerabilities [23]. Furthermore, [24] introduces a method named SPECS, which serves as a lightweight mechanism to protect software from security-critical bugs in the OR1200 processor. The work in [21] introduces a tool named Isadora for generating security assertions based on information flow tracking techniques, which is more suitable for only tracking the information flows of the processors and hardware designs. It produces overly general security assertions that merely delineate signal paths between block diagrams. The work in [18] formally verifies information flow security for ARM processor kernel and user modes. Nevertheless, its formal nature introduces scalability issues, making it less applicable to large designs.

Moreover, the work presented in [6] introduces an assertion miner called HARM designed explicitly for the functional verification of hardware designs. However, it purports the ability to identify Hardware Trojans (HTs) within these designs.

All in all, the majority of current approaches neither explicitly focus on the security verification of processors nor include an automated method to generate security assertions to check the consistency between processor's security requirements and the actual implementation [25].

Thus, to fill in the gap, in this paper, the proposed method first, collects the most important security properties of RISC-V processors by studying RISC-V specifications [26, 27] and processor security requirements in the literature [19, 20, 22–24] (details are described in Section IV-A); second, proposes a systematic approach to extract ISA characteristics of each security property (details are presented in section IV-B); third, introduces an automatic assertion miner which takes these ISA characteristics as well as simulation trace of the processor as inputs to automatically extract a set of security assertions (details are described in section IV-C).

To the best of the author's knowledge, this is the first automated security-based assertion miner that can extract security assertions with the guidelines of ISA characteristics of security requirements, directly from the simulation trace of the processor. This extracted security assertion set is employed in the security verification process to uncover vulnerabilities, such as hardware Trojans, embedded within the design. Therefore, the contributions of this paper are listed as follows:

- A systematic method for translating security properties of the processor to the instruction set architecture. The idea is that the security properties that describe the general behavior of a 32-bit processor (*e.g.*, read correctly from memory) will be translated to the proposer 32-bit instruction set;
- An automatic security-based assertion miner to generate security assertions with a high Trojan detection rate;
- The proposed security-based assertion miner can automatically analyze the ISA characteristics (32-bit instruction set) and the simulation traces of the processor to generate security assertions in a short execution time;
- The proposed method is expandable to any processor family and is not limited to RISC-V.

The paper is organized as follows: The preliminaries are presented in section II. The related background and concepts are described in section III and the proposed method is elaborated in section IV. Section V presents the experimental results and finally, section VI concludes the paper.

II. Preliminaries

In this section, we briefly explain the definitions used in this paper.

Definition 1: A ***simulation trace*** consists of the values of the variables of hardware designs that have been stored as records of data for different time instants (clock cycles) during the execution of the designs [6].

Definition 2: A ***security property*** in this study is defined as a critical security aspect of the processor that neglecting consideration of it can lead to security vulnerabilities in the processor. These security properties are usually presented in the specification of the processor [26, 27].

Definition 3: An ***atomic proposition*** is a logical formula that does not contain logical connectives [8]. Examples of *atomic propositions* are such as a1 = True, and b1 = False.

Definition 4: A ***proposition*** is a composition of *atomic propositions* through logical connectives [8]. An example for *proposition* is a1 = True && b1 = False && c1 = True.

Definition 5: A ***security assertion*** in this study checks the consistency between the defined behaviors in the *security properties* (Definition 2) and the actual implementation when it faces an attack. A *security assertion* is a composition of *propositions* through temporal operators that must hold or must become true during the execution of the design [8]. Typically, a security assertion is divided into two parts: the left side, named *antecedent*, and the right side, called *consequent* [8].
The general structure of a security assertion in Property Specification Language (PSL) is like $always(antecedent \rightarrow consequent)$, which implies that the consequent will hold whenever the antecedent occurs [28].

Definition 6: ***Temporal pattern next[$\mathcal{N}$]***: *Next[$\mathcal{N}$]* temporal pattern in PSL is in the form of: $always(antecedent \rightarrow next[\mathcal{N}]\ consequent)$ [28]. This temporal pattern means that when antecedent occurs, after $\mathcal{N}$ time instant (clock cycle), consequent will occur [28]. $\mathcal{N}$ is an integer value and $\mathcal{N} > 0$.

Definition 7: ***Frequent itemsets*** refer to a set of variables in simulation trace that occur with a frequency, indicating significant relations/associations between the variables.

Definition 8: An ***Association Rule (AR)*** is defined as an implication of the form $\mathcal{X} \rightarrow \mathcal{Y}$ where $\mathcal{X}, \mathcal{Y} \subseteq \mathcal{I}$, with $\mathcal{X} \cap \mathcal{Y} = \emptyset$, and $\mathcal{I}$ is a set of items [29–31]. $\mathcal{X}$ and $\mathcal{Y}$ are called *frequent itemsets*.

Definition 9: ***Support*** is a metric in association rule mining that indicates how frequently an itemset appears in the dataset [31]. This value is between 0 and 1. For the rule $\mathcal{X} \rightarrow \mathcal{Y}$, the value of support is calculated with the following formula [31]:

$$Supp(\mathcal{X} \rightarrow \mathcal{Y}) = P(\mathcal{X} \cup \mathcal{Y}) \tag{1}$$

In (1), $P(\mathcal{X} \cup \mathcal{Y})$ is the probability where $\mathcal{X} \cup \mathcal{Y}$ indicates that a record contains both $\mathcal{X}$ and $\mathcal{Y}$, that is the union of itemsets $\mathcal{X}$ and $\mathcal{Y}$.

Definition 10: The **min_supp** value is the threshold and a minimum value for *support* to decide whether an itemset

is frequent (*i.e.*, occurs frequently in the simulation trace) or not [31]. If the frequency of the itemset is more than this threshold, the itemset is considered a frequent itemset [31]. A higher value of min_supp leads to generating commonly occurring (general) ARs, while a lower value of min_supp leads to generating rarely occurring ARs (corner cases) [31].

Definition 11: ***Confidence*** is an indication of how often the rule has been found to be true [32]. For the rule $\mathcal{X} \rightarrow \mathcal{Y}$, this value is calculated with the following formula [31, 32]:

$$Conf(\mathcal{X} \rightarrow \mathcal{Y}) = P(\mathcal{Y}|\mathcal{X}) \tag{2}$$

It evaluates the degree of certainty of the detected association rule. This is taken to be the conditional probability $P(\mathcal{Y}|\mathcal{X})$, that is the probability that a record containing $\mathcal{X}$ also contains $\mathcal{Y}$. This value is between 0 and 1.

Definition 12: The **min_conf** is the minimum value for *confidence* [31]. The higher value of min_conf leads to fewer but more accurate and valid association rules [31].

III. Background

In this section, we briefly explain the related concepts and background used in this paper.

A. RISC-V Instruction Set Architecture

RISC-V provides a flexible instruction set for various general and application-specific scenarios. Around a set of basic mandatory instructions for arithmetic, control flow and memory instructions, called the base instruction set (*e.g.*, RV32I for the 32 bit base instruction set) [26, 33], various instruction set extensions can be appended (*e.g.*, multiply/divide, floating point numbers, vector operations).

In this work, we utilize the RV32I base instruction set, but the proposed method is independent of the specific ISA or utilized instruction set extensions. The RV32I base instruction set defines a 32 bit architecture around 32 general-purpose registers `x0` to `x32` (with `x0` being constant 0). More information on the RISC-V instruction set, and its various extensions can be found in Volume 1 [26] and further details on the privileged architecture details, especially CSRS, can be found in Volume 2 [27] of the RISC-V Specification, respectively.

B. Threat model: MicroRV32 Platform

Amongst the many available open-source implementations, we choose the MicroRV32 platform [34], implemented in the open-source Hardware Description Language SpinalHDL. Through the modern SpinalHDL language, it is possible to prototype modifications in the data path quickly, while keeping control over the generated Verilog or VHDL description. The platform is synthesizable for FPGAs and ASICs, while providing a lightweight and robust microarchitecture. MicroRV32 features a configurable multi-cycle processor compliant to RV32IMC, meaning it is capable of the aforementioned RISC-V 32-bit base instruction set (I), the multiply/divide extension (M) and the compressed instruction extension (C). Within the platform, a set of peripherals enables the interaction with the outside environment, similar to other microcontroller units.

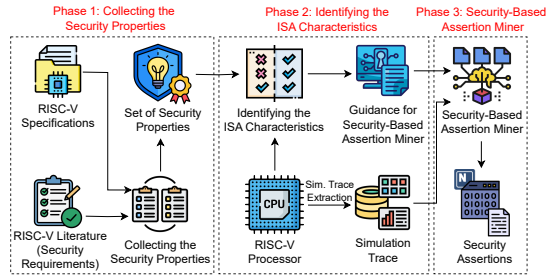


Figure 1: Overview of the proposed method

In this work, our threat model environment consists of a processor based on MicroRV32 embedded in a SoC, with various peripherals and a memory hierarchy.

C. Attack Model

To evaluate the proposed method, we have implemented Hardware Trojans (HTs) based on the following details. The attacker targets the RISC-V RTL code, aiming to disrupt normal operations of IPs and cause damages to the IP design house, *e.g.*, financial losses for any reason. Specifically, the attacker intends to add three Hardware Trojans to the processor: two of them alter the control unit's functionality, while one focuses on the memory and illegal access to it. The attacker possesses knowledge of the design modules and implementation. HTs typically consist of a trigger and a payload [35]. The trigger is the condition activating the Trojan, while the payload executes the malicious function [35]. Triggers can be of different types like Always-On, Conditional, or Time-Based, with payloads causing diverse corruptions like Data and Control Flow Manipulation, DoS, etc [35]. Our HTs use conditional triggers, activating under specific rare conditions, and their payload manipulates program data and control flow. Trojan 1 triggers a specific input combination in the control flow. Its payload alters the execution flow and causes incorrect computation in specific part of control unit. Trojan 2 activates through a specific sequence of control signals, initiating illegal memory access with a payload involving unauthorized memory access. Trojan 3 is triggered by an improbable combination of input conditions, leading to the alteration of the opcode signal and manipulation of update registers. Its payload executes incorrect instructions, disrupting the normal program flow, potentially compromising system integrity, and enabling the attacker to control the instruction sequence. In this work, the HTs have been implemented so that they will be activated in very rare conditions, making their detection difficult.

IV. Methodology

Fig. 1 provides an overview of our proposed method, which is structured in three key phases. These phases are 1- Collecting the Security Properties, 2- Identifying the ISA Characteristics, and 3- Security-Based Assertion Miner.

Table I: Collected Security Properties

Security Properties	Security Property Type
1: Calculation of memory address and memory data is correct	Memory Access
2: Jumps update the program counter correctly	Control Flow
3: Jumps update the link register correctly	Update Register
4: Addition with register value and immediate value results in correct result in the correct target register	Update Register
5: Load immediate value into the upper 20 bits in the correct target register	Update Register
6: Adds the immediate as upper 20 bits to the program counter and puts it into the correct target register	Update Register

In the first phase, leveraging RISC-V specifications [26, 27] and already introduced security properties from literature [19, 20, 22–24], a set of RISC-V security properties (Definition 2) is collected. This set, along with the RISC-V implementation, becomes the input for the second phase. In the second phase, the security properties of the processor are translated to the instruction set architecture. The main aim of this phase is to systematically identify corresponding instruction sets and the associated signals related to each security property.

Based on this identification, a guideline is formulated for the next phase. This guide assists the security-based assertion miner in the third phase. Operating with the simulation trace generated from the RISC-V design and the guidance report, the security-based assertion miner automatically mines a set of security assertions (Definition 5). These assertions encompass the behaviors outlined in the security properties identified in the initial phase. Further details for each phase are elaborated in the subsequent subsections.

A. Collecting the Security Properties

After reviewing the RISC-V specifications [26, 27] and its security requirements from literature [19, 20, 22–24], a set of security properties (Definition 2) have been collected for the purpose of this work. These collected security properties are outlined in Table I. However, it is noteworthy that the proposed method can seamlessly extend to encompass other security properties. This versatility enables the method to cater not only to the collected set in our case study benchmark (RISC-V) but also to the diverse security properties of other processors.

In this study, we have collected security properties from the most important categories pertinent to processor security, namely Memory Access, Control flow, and updating Registers. In papers [19, 20, 22–24] which represent the latest studies on processor security properties, these categories have been reported as critical areas requiring scrutiny in the security verification process of processors. Consequently, as depicted in Table I, security properties corresponding to each of these categories have been delineated. More details for these security properties are as follows:

- Security property 1: The first security property of Table I describes the correctness and relation between the current load or store instruction and the resulting memory address and data on the memory interface. For example, if an HT corrupts the address maliciously to redirect the data, the mismatch between the intended address and the actual address should become visible.
- Security property 2: The second security property ensures, that the jump instructions redirect the control flow correctly, *i.e.,* the change in program counter reflects the provided register and immediate values accordingly. In this case, an HT could redirect the control flow to malicious code before returning to the original application's code.
- Security property 3: As jump instructions are used to call functions and return to the code calling a function, the storage of this return address can similarly be tampered with. Thus, the third security property ensures, that the return address saved on jump instructions is correct.
- Security property 4: As control flow and memory access instructions prepare addresses and values through arithmetic and logic operations, their correctness is vital for security. To avoid information leakage through HT, the fourth security property assures that the `ADDI` (add immediate) instruction saves the correct result only into the correct target register. As an example, an HT could enable writing to the target register and a temporary register, in order to leak the information in a different subroutine.
- Security property 5: Similarly, the fifth security property ensures the correctness for the `LUI` (load upper immediate) instruction, that the accordingly extended immediate value is stored into the correct target register.
- Security property 6: Lastly, the sixth security property addresses the `AUIPC` (add upper immediate and program counter) instruction. The security property ensures the correct arithmetic and storage in the correct target register. As this instruction is utilized to prepare target addresses, an HT can potentially redirect the control flow to a different part of the code.

These security properties serve as the inputs for the next phase, *i.e.,* Identifying the ISA Characteristics.

B. Identifying the ISA Characteristics

In order to identify which parts of the ISA specification contribute to a security property, we will utilize an example that covers security properties 2 and 3. Consider the RISC-V instruction `JAL rd, offset`, *Jump And Link*. This

instruction manipulates the control flow of the application, jumping to a new location in the program by setting the *Program Counter* (PC) to an `offset` encoded as an immediate value (*i.e.*, `PC = offset`). As a second part, the instruction will store the value of program counter of the instruction after the `JAL` instruction (`rd = PC + 4`). We can consider possible violation of security if, for example, a HW Trojan introduces are change of the offset under specific system conditions. This could lead to a possible attack in which an unwanted execution of different code occurs. Therefore, for the `JAL` instruction, we can denote two possible high-level properties:

(a) Jumps update the PC correctly according to the `offset` passed in the instruction immediate.
(b) Jumps update the register `rd` with the correct PC (*i.e.*, `rd = PC+4`).

Consequently, the security-based assertion miner would need to find potential security assertions covering the signals mentioned in the high-level properties. In a more general sense, these steps can be abstracted as follows:

1) Identifying instructions affected by threat model (*e.g.*, control flow integrity affects the instructions for branch, jump, and address manipulation).
2) Identifying each part of functional behavior contributing to instructions (*e.g.*, jump instruction changes PC and a register based on PC value).
3) Formulating high-level property reflecting parts of the functional behavior (*e.g.*, resulting addresses for store or load operations have to be consistent with the initial instruction and register values).
4) Utilizing the high-level property to identify the affected signals in the processor and its interfaces. This includes the relation between instruction bits and the observable behavior on results and interfaces.

In this phase, alongside the guidance report created based on the identified ISA characteristics, a simulation trace is generated from the RISC-V design. These inputs are utilized by the security-based assertion miner in the next phase.

C. Security-Based Assertion Miner

In this phase, we elaborate on the details of the proposed security-based assertion miner. As illustrated in Fig. 2, the proposed miner is comprised of four primary steps: 1) Signal Identification and Pre-processing of Simulation Trace, 2) Association Rule Mining, 3) Time Notation, and 4) Assertion Conversion.

During the Signal Identification and Pre-processing step, after identifying the RISC-V signals that are associated with the identified security properties, the exhaustive simulation trace of the RISC-V undergoes pre-processing to prepare the data. Subsequently, in the Association Rule Mining step, we apply our algorithm to the pre-processed simulation trace to mine all association rules (Definition 8) derived from the simulation trace.

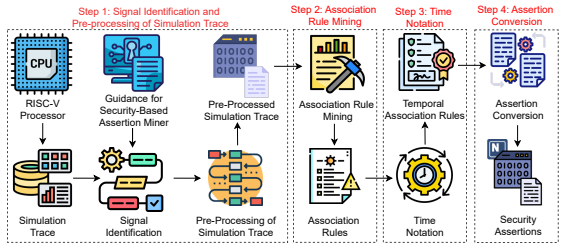


Figure 2: Proposed Security-Based Assertion Miner

Algorithm 1: Security-Based Assertion Miner

```
1  Input: 𝒩, 𝒮𝒯, min_supp
2  Output: next[𝒩] = antecedent → next[𝒩]consequent
   /* Initialization                                   */
3  ℛ = antecedent → consequent
4  L_1 = {frequent 1−itemsets ∈ 𝒮𝒯'}
5  K = 2
   /* Signal Identification & Preprocessing            */
6  𝒮𝒯' ← prune_nonrelevant_security_signals(𝒮𝒯)
7  forall f ∈ 𝒮𝒯' do
8      𝒮𝒯' = MoveUp(f(𝒩))
   /* Association Rule Mining                          */
9  while L_{k−1} != ∅ do
10     C_k = generate_candidate_itemsets(L_{k−1})
11     L_k = prune_infrequent_itemsets(C_k, min_supp)
12     k = k + 1
13 foreach frequent itemset L_i ∈ L do
14     foreach subset S of L_i do
15         if (S != ∅) && (S != L_i) then
16             confidence = support(L_i) / support(S)
17             if confidence >= min_conf then
18                 ℛ ← association_rule(S => L_i)
19 return ℛ
   /* Time Notation                                    */
20 if (ℛ.antecedent == (t ∈ 𝒮𝒯')) and (ℛ.consequent == (f ∈ 𝒮𝒯')) then
21     next[𝒩] ← label(ℛ)
22 else
23     Discard(ℛ)
```

Afterward, the association rules obtained from the second step are passed to the third step, *i.e.*, Time Notation. Here, the extracted association rules are integrated with the concept of time to generate appropriate time-integrated rules (temporal association rules) in the form of $next[\mathcal{N}]$ pattern (Definition 6).

These rules serve as the input for the Assertion Conversion step. Consequently, the Assertion Conversion step transforms the rules from the previous step into assertions, rendering them ready for utilization in the security verification process of the RISC-V design.

Algorithm 1 presents the detailed process of the first three steps of the security-based assertion miner. In this algorithm, $\mathcal{ST}$ denotes the simulation trace and $\mathcal{ST}'$ is the pre-processed simulation trace, while f and t represent the simulation trace's outputs and input values.

In the following subsections, we discuss each phase of the security-based assertion miner in more detail.

1) Signal Identification and Pre-processing of Simulation Trace

Lines 6 to 8 of the Algorithm 1 are related to the Signal Identification and Pre-processing step of the security-based assertion miner.

In this phase of the security-based assertion miner, at first, the RISC-V simulation trace and guidance report generated from the second phase of the method (section IV-B) are processed for signal identification. In signal identification, the assertion miner prunes all the signals of the simulation trace that are not relevant to the identified security properties. Once signals associated with the specified security properties are identified, the simulation trace undergoes pre-processing.

To pre-process the simulation trace, all the identified output of the simulation trace is moved $\mathcal{N}$ records above its original position (line 8 of the Algorithm 1). However, the identified inputs of the simulation trace remain as they are. Traditional association rule mining algorithms (*e.g.*, Apriori [31]) cannot typically mine the rules in the form of *next[*$\mathcal{N}$*]*. Because of this reason, and also since the corresponding output of input variables in a sequential hardware design may occur in the simulation trace $\mathcal{N}$ time instants later, this pre-processing needs to be performed. This ensures the correct alignment of outputs with their corresponding inputs, allowing for accurate temporal analysis and also mining patterns for different $\mathcal{N}$ time instants (clock cycles).

Fig. 3 illustrates an example of pre-processing for *next[2]* clock cycles. The simulation trace in Fig. 3.1 is pre-processed by moving the output parts 2 time instants above their original positions, resulting in the modified simulation trace shown in Fig. 3.2. The figure uses T to represent the true value, and F to show the false value. In this example, the simulation trace consists of 5 records, divided into two categories: input variables and output variables. Each variable is assigned its corresponding value at each time instant. For example, the first row indicates that v2 is equal to 01 at time t0 and 11 at time t1. The last two records in Fig. 3.2 are marked as not available (NA) due to the absence of data after time instant t4 to be moved in front of these two records.

Time	Inputs			Outputs	
	v1	v2	v3	v4	v5
t0	001	01	F	11	T
t1	001	11	F	10	T
t2	101	00	T	01	T
t3	110	10	F	10	F
t4	000	01	T	11	F

(1)

moving 2 time instants

Preprocessed

Time	Inputs			Outputs	
	v1	v2	v3	v4	v5
t0	001	01	F	01	T
t1	001	11	F	10	F
t2	101	00	T	11	F
t3	110	10	F	NA	NA
t4	000	01	T	NA	NA

(2)

Figure 3: (1) Simulation trace (2) Pre-processed simulation trace

2) Association Rule Mining

After pre-processing the simulation trace in the previous step, the resulting pre-processed simulation trace is subsequently fed into lines 9 to 19 of Algorithm 1 to mine association rules. Applying these lines of the algorithm to the pre-processed simulation trace provides us with a set of association rules in the form of $antecedent \rightarrow consequent$.

In lines 9 to 12 of Algorithm 1, frequent itemsets (Definition 7) of various sizes (1-itemsets, 2-itemsets, etc.) are generated iteratively until the list of the frequent itemsets is empty. Specifically, the algorithm mines frequent itemsets whose support values (Definition 9) exceed the min_supp value (Definition 10), while pruning the others. In line 10 of the algorithm, C_k is the candidate itemsets of size k that are generated by combining frequent (k-1)-itemsets and L_k in line 11 of the algorithm is the set of frequent k-itemsets. In this algorithm, 1-itemsets consist of individual variables of simulation trace, 2-itemsets are pairs of variables, etc.

After mining the frequent itemsets and adding them to the L_k list, in lines 13 to 19 of the algorithm, the association rules are extracted from the list of frequent itemsets. To clarify these lines of the algorithm, let's consider an example where the list of frequent itemsets is equal to 4-itemsets of {A, B, C, D}. To generate association rules from this frequent itemset, we consider all non-empty subsets of it. These subsets are 1-itemsets of {A}, {B}, {C}, {D}, 2-itemsets of {A, B}, {A, C}, {A, D}, {B, C}, {B, D}, {C, D}, as well as the 3-itemsets of {A, B, C}, {A, B, D}, {A, C, D}, {B, C, D}, and 4-itemset of {A, B, C, D}. Afterward, for each non-empty subset $\mathcal{Y}$, an association rule of the form $\mathcal{Y}$ => $\mathcal{X} - \mathcal{Y}$ is generated. By doing so, the algorithm considers all possible combinations of items in the frequent itemset to identify significant associations between different sets of items. For example, if $\mathcal{X}$ = {A, B, C, D} and $\mathcal{Y}$ = {A, B, C}, then the rule {A, B, C} => {D} is generated. The antecedent of the rule ($\mathcal{Y}$) is the subset {A, B, C}, and the consequent of the rule ($\mathcal{X} - \mathcal{Y}$) is the set difference between the frequent itemset {A, B, C, D} and the antecedent {A, B, C}, which is {D}. This process is repeated for each non-empty subset of the frequent itemsets list, yielding a set of association rules. Finally, we evaluate each association rule for its confidence, pruning those that are below the min_conf threshold (Definition 12).

Increasing the min_supp value results in fewer assertions that describe more general design behavior, while decreasing the min_supp value leads to assertions covering rare design behavior (corner cases). These corner cases are important as attackers can consider them for performing any corruption in the design. Similarly, raising the min_conf value produces fewer but more valid assertions. Valid assertions refer to assertions that will not be violated during the simulation with different attack scenarios. The utilization of these values in the security-based assertion miner facilitates an effective vulnerability detection process. In this paper, min_supp and min_conf are set to 0.01 and 1, respectively, as we aim to discover corner cases while achieving high vulnerability detection (details are presented in Section V).

At this point, with the completion of the association rule mining, these rules serve as the fundamental components of the Time Notation step.

Table II: Detailed Experimental Results on Six Different Security Properties

Total Number of Security Assertions	Number of Generated Security Assertions for each Security Property					
	Security Property 1	Security Property 2	Security Property 3	Security Property 4	Security Property 5	Security Property 6
4036	870	1490	1522	2026	1898	1898

3) Time Notation

In the previous step, the method provides us a set of rules in the general form of $antecedent \rightarrow consequent$. In this step, the method integrates the concept of time into the association rules generated in the association rule mining step, leading to a set of temporal association rules in the form of $antecedent \rightarrow next[\mathcal{N}]consequent$. Lines 20 to 23 in Algorithm 1 describe the details of the Time Notation step. If the antecedent value matches an input in the simulation trace, and the consequent value has already been moved to another record in the simulation trace, the rule is labeled as a *next* temporal association rule. Otherwise, other mined rules are discarded.

4) Assertion Conversion

In this step, the mined temporal association rules are transformed into temporal security assertions (Definition 5) in SystemVerilog Assertions (SVA) language [36] using the labels assigned in the Time Notation step. While the security-based assertion miner provides assertions in the SVA language, this section elucidates the general structure and format of temporal mined rules in PSL [28] for enhanced comprehension. Consequently, the output of the Time Notation step for temporal association rules labeled as *next[*$\mathcal{N}$*]* is transformed into the PSL format of "*always(antecedent* $\rightarrow$ *next[*$\mathcal{N}$*]consequent)*".

V. Experimental Results

For our experimental evaluation, we utilized the security-based assertion miner to generate security assertions in the form of SVA. As shown earlier, in Fig. 2, the security-based assertion miner requires a simulation trace generated from a processor. In our case study, we utilized the processor embedded in the open-source MicroRV32 platform [34]. For generating the simulation trace, we executed a software application on the processor that is centered around checksum calculations for embedded systems. It features different types of control flow, loops, arithmetic operations, and interaction with the memory as well as the available peripherals. Hence, we generated a simulation trace with 10000 records that activates various parts of the microarchitecture to provide a diverse data set for security assertion mining. The generated assertions are then evaluated by including Hardware Trojans in the processor's microarchitecture together with the generated security properties, to verify them. Notably, in our experiments the values for the minimum support (Definition 10) and minimum confidence (Definition 12) have been set to 0.01 and 1, respectively. These values allow us to guide the security-based assertion miner in mining fewer yet more valid assertions, effectively considering the corner cases of designs. Moreover, $\mathcal{N}$ has been set to 2 for the $next[\mathcal{N}]$ pattern, but it can be adjusted to other values.

Table III: Experimental Results on Detected Trojans

Trojans	#Security Assertions Detecting Trojans	Trojan Detection
Trojan 1	16	✓
Trojan 2	64	✓
Trojan 3	176	✓

✓: Trojan has been detected.

Table II exhibits the detailed experimental results about the assertions that are associated with any of the security properties that we presented in section IV-A. The proposed security-based assertion miner generated a total of 4036 security assertions. It should be noted that while the total number of generated assertions is 4036, some of them overlap so that they contain the signals of the design that are related to several security properties. According to the experimental results in Table II, 870 and 1490 security assertions are related to security properties 1 and 2, which means that these numbers of assertions can cover the behaviors that have been described for these two security properties. The results indicate that for security properties 3 and 4, 1522 and 2026 security assertions have been mined, respectively. This figure for both the security properties 5 and 6 is 1898 assertions.

Table III presents the experimental results on the Trojan detection. The column '#Security Assertions Detecting Trojans' presents the number of security assertions that could detect any of the Trojans. For Trojan 1, 16 assertions detected it and 64 and 176 security assertions detected Trojans 2 and 3, respectively.

Table IV presents a comparative analysis between our proposed security-based assertion miner and HARM, the latest assertion miner in the literature [6]. HARM is explicitly presented as a tool that can be employed in the context of security verification for Trojan detection, making it a relevant tool for our comparison. While HARM generated an extensive set of 16073 assertions, contributing to prolonged security verification process, our method yielded a more compact and accurate set of 4036 assertions. Among all the mined assertions by HARM, a total of 397 assertions could detect the Trojans, while we accomplished Trojan detection with 256 assertions. The ratio of security assertions that detect Trojans to the total number of assertions demonstrates that our mined security assertions are more effective in Trojan detection. Our assertions exhibited a superior effectiveness with a ratio of 6.3%, in contrast to HARM's 2.4%. Considering the fact that there are only three Trojans, and as mentioned in section III-C, the probability of their activation is very rare, 6.3% shows promise. These findings underscore the efficiency of our method in producing a smaller yet more potent and accurate set of security assertions. Furthermore, our security-based assertion miner demonstrated significantly shorter execution time, completing the assertion mining process in approximately

Table IV: Comparison of the proposed method with HARM

Assertion Miner	Total number of Security Assertions	#Security Assertions Detecting Trojans	Ratio of Security Assertions Detecting Trojans to the Total Number of Assertions (%)	Execution Time
Proposed Method	4036	256	6.3	5min30sec
HARM [6]	16073	397	2.4	74min31sec

5 minutes, compared to HARM's duration of over an hour.

VI. Conclusion

In this paper, we introduced a method for generating security assertions tailored to a RISC-V processor. The method involves systematically analyzing design specifications and requirements to pinpoint critical security properties, neglecting which could result in design vulnerabilities. By identifying the pertinent security properties and their associated design signals, our proposed security-based assertion miner is applied to the processor's simulation trace. The miner automatically outputs security assertions. Our experiments demonstrate that these assertions proficiently encapsulate the behavior described by identified security properties, effectively detecting injected Hardware Trojans within the design.

VII. Acknowledgement

This work was supported in part by the Estonian Research Council grants PSG837 and the German Federal Ministry of Education and Research (BMBF) within the project ECXL under contract no. 01IW22002, the project SASPIT under contract no. 16KIS1852K and VE-HEP under grant no. 16KIS1342.

References

[1] A. Harris, T. Verma *et al.*, "Morpheus ii: A risc-v security extension for protecting vulnerable software and hardware," in *HOST*, 2021, pp. 226–238.

[2] A. L. D. Antón, J. Müller *et al.*, "Fault attacks on access control in processors: Threat, formal analysis and microarchitectural mitigation," *IEEE Access*, vol. 11, pp. 52 695–52 711, 2023.

[3] E. M. Koruyeh, K. N. Khasawneh *et al.*, "Spectre returns! speculation attacks using the return stack buffer," in *12th USENIX Workshop on Offensive Technologies (WOOT 18)*. USENIX Association, 2018.

[4] R. Hariharan, T. Ghasempouri *et al.*, "From rtl liveness assertions to cost-effective hardware checkers," in *2018 Conference on DCIS*, 2018, pp. 1–6.

[5] D. Pal, V. Dodeja *et al.*, "Goldmine: A tool for enhancing verification productivity."

[6] S. Germiniani and G. Pravadelli, "Harm: A hint-based assertion miner," *IEEE TCAD*, vol. 41, no. 11, pp. 4277–4288, 2022.

[7] J. Malburg, T. Flenker *et al.*, "Property mining using dynamic dependency graphs," in *ASP-DAC*, Jan 2017, pp. 244–250.

[8] A. Danese, N. D. Riva *et al.*, "A-team: Automatic template-based assertion miner," in *54th DAC conf.*, 2017, pp. 1–6.

[9] M. R. Heidari Iman, J. Raik *et al.*, "A methodology for automated mining of compact and accurate assertion sets," in *NorCAS*, 2021, pp. 1–7.

[10] S. Germiniani and G. Pravadelli, "Exploiting clustering and decision-tree algorithms to mine ltl assertions containing non-boolean expressions," in *VLSI-SoC*, 2022.

[11] M. R. Heidari Iman, J. Raik *et al.*, "An automated method for mining high-quality assertion sets," *MICPRO*, vol. 97, p. 104773, 2023.

[12] H. Witharana, A. Jayasena *et al.*, "Automated generation of security assertions for rtl models," *J. Emerg. Technol. Comput. Syst.*, vol. 19, no. 1, jan 2023.

[13] F. Erata, S. Deng *et al.*, "Survey of approaches and techniques for security verification of computer systems," *J. Emerg. Technol. Comput. Syst.*, vol. 19, no. 1, jan 2023.

[14] T. Ghasempouri, J. Raik *et al.*, "Survey on architectural attacks: A unified classification and attack model," *ACM Comput. Surv.*, vol. 56, no. 2, sep 2023.

[15] T. Ghasempouri, J. Malburg *et al.*, "Engineering of an effective automatic dynamic assertion mining platform," in *2019 IFIP/IEEE 27th VLSI-SoC*, 2019, pp. 111–116.

[16] M. Eslami, T. Ghasempouri *et al.*, "Reusing verification assertions as security checkers for hardware trojan detection," in *2022 23rd ISQED Symposium*, 2022, pp. 1–6.

[17] H. Witharana, Y. Lyu *et al.*, "A survey on assertion-based hardware verification," *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–33, 2022.

[18] N. Dong, R. Guanciale *et al.*, "Formal verification of correctness and information flow security for an in-order pipelined processor," in *FMCAD*, 2023, pp. 247–256.

[19] C. S. Chuah, C. Appold *et al.*, "Formal verification of security properties on risc-v processors," in *21st MEMOCODE*, 2023, pp. 159–168.

[20] R. Zhang and C. Sturton, "Transys: Leveraging common security properties across hardware designs," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1713–1727.

[21] C. Deutschbein, A. Meza *et al.*, "Isadora: Automated information flow property generation for hardware designs," in *Proc. of the 5th Workshop on Attacks and Solutions in Hardware Security*, ser. ASHES '21. Association for Computing Machinery, 2021, p. 5–15.

[22] R. Zhang, N. Stanley *et al.*, "Identifying security critical properties for the dynamic verification of a processor," *SIGARCH Comput. Archit. News*, vol. 45, no. 1, p. 541–554, apr 2017.

[23] B. Kumar, A. K. Jaiswal *et al.*, "Analyzing hardware security properties of processors through model checking," in *2020 33rd International Conference on VLSI Design (VLSID)*, 2020, pp. 107–112.

[24] M. Hicks, C. Sturton *et al.*, "Specs: A lightweight runtime mechanism for protecting software from security-critical processor bugs," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 517–529.

[25] F. Farahmandi, M. S. Rahman *et al.*, *CAD for Hardware/Software Security Verification*. Cham: Springer International Publishing, 2023, pp. 187–210.

[26] A. Waterman, K. Asanovic *et al.*, "The risc-v instruction set manual volume i: Unprivileged isa," *Document Version*, vol. 20191213, 2019.

[27] ——, "The risc-v instruction set manual, volume ii: Privileged architecture," *RISC-V Foundation*, 2019.

[28] "Standard for property specification language (psl), ieee standard," p. 1–184, 2012.

[29] C. Antunes and A. L. Oliveira, "Temporal data mining: an overview," 2001.

[30] S. Bilqisth and K. Mustofa, "Determination of temporal association rules pattern using apriori algorithm," *IJCCS Journal*, vol. 14, p. 159, 04 2020.

[31] J. Han, M. Kamber *et al.*, "6 - mining frequent patterns, associations, and correlations: Basic concepts and methods," ser. The Morgan Kaufmann Series in Data Management Systems, 2012, pp. 243–278.

[32] M. R. H. Iman, P. Chikul *et al.*, "Anomalous file system activity detection through temporal association rule mining," in *9th ICISSP*, 2023, pp. 733–740.

[33] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design RISC-V Edition*, ser. The Morgan Kaufmann Series in Computer Architecture and Design, 2017.

[34] S. Ahmadi-Pour, V. Herdt *et al.*, "The microrv32 framework: An accessible and configurable open source risc-v cross-level platform for education and research," *Journal of Systems Architecture*, vol. 133, p. 102757, 2022.

[35] S. Bhasin and F. Regazzoni, "A survey on hardware trojan detection techniques," in *ISCAS*, 2015, pp. 2021–2024.

[36] C. B. Spear, *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*, 2nd ed. Springer, 2010.

Appendix 6

VI

MRH. Iman, J. Raik, M. Jenihhin, G. Jervan, and T. Ghasempouri, "A Methodology for Automated Mining of Compact and Accurate Assertion Sets," *2021 IEEE Nordic Circuits and Systems Conference (NorCAS)*, Oslo, Norway, 2021, pp. 1-7, DOI: https://doi.org/10.1109/NorCAS53631.2021.9599865.

A Methodology for Automated Mining of Compact and Accurate Assertion Sets

Mohammad Reza Heidari Iman, Jaan Raik, Maksim Jenihhin, Gert Jervan and Tara Ghasempouri
Department of Computer Systems, Tallinn University of Technology, Tallinn, Estonia
{mohammadreza.heidari, jaan.raik, maksim.jenihhin, gert.jervan, tara.ghasempouri}@taltech.ee

***Abstract*—Assertion-based verification has become a viable solution for functional verification. Manual definition of assertions is costly and needs human expertise. On the other hand, the automatic approaches are still struggling with generating high-quality assertion sets in terms of accuracy, and readability. To overcome the mentioned shortcomings, this paper exceeds the state-of-the-art by introducing a new fully automated approach to generate a set of complete and accurate assertions. Moreover, the mined assertions are readable thereby, easy to understand by the verification engineer. Furthermore, the method is equipped with a feature that communicates with users to select the assertions related to a specific block based on the user's need for further debugging and design analysis. Experimental results present the effectiveness of the proposed approach by showing that it generates significantly more compact assertion sets than the state-of-the-art while achieving 100% detection of the injected mutants.**

I. Introduction

Digital systems become more complex with each generation. Therefore, verifying that their behavior is correct has become a very challenging task. To this end, functional verification aims at guaranteeing that the design of a system satisfies its specification before manufacturing, by detecting and removing design errors [1]. Among all the solutions for ensuring the robustness of the systems, Assertion-Based Verification (ABV) has emerged as one of the most popular solutions for checking the design functionality [1].

An assertion is a Boolean expression that defines the behavior of designs [1]. Traditionally, assertions were defined manually [2, 3]. However, the manual definition of assertions needs human expertise, it is costly and error-prone [2, 3]. In general, to write an assertion set, a verification engineer needs creativity and a deep understanding of the design's functionality [4, 5]. Therefore, some studies have been carried out to automatically mine the assertion sets [6–14].

Several works on assertion mining for digital designs have been proposed. The work in [6] is an automatic assertion miner which generates assertions using a dynamic dependency graph. They have extracted relations between signals of the design using simulation traces. [7] is another approach that uses a syntax-guided enumeration assertion miner. [8] and [9] are techniques that extract assertions using several templates in the form of Finite State Machines (FSMs). The GoldMine tool is presented in [15] and [16]. The tool automatically generates assertions for a given Register-Transfer Level (RTL) design. This tool uses simulation traces, formal verification, and static code analysis. Finally, [12–14] combine a dynamic dependency graph and FSM to achieve the strength of both techniques for assertion extraction.

978-1-6654-0712-0/21/$31.00 ©2021 IEEE

However, it is notable that in most assertion mining approaches, the number of generated assertions is prohibitively high, which can lead to a redundant or inconsistent assertion set, consequently resulting in exhaustive, non-compact assertion sets. To that end, some studies have been performed to select a set of good quality assertions from a large number of generated assertions [17–19].

In the context of assertions' selection from a large number of generated assertions, there are some works in the literature to rank the assertions and select the interesting ones to provide this set to the verification engineer. For instance, in [20], the assertions are ranked based on two main metrics, Importance, and Complexity. The degree of Importance is higher in the assertions which are describing the output of the design and Complexity is related to the number of logic operators used in an assertion. In [21], the estimation is based on the number of propositions included in the antecedent of the assertion. In [17–19, 22], mined assertions are mainly ranked according to several data mining metrics such as Support (*i.e.* their frequency of occurrences in the simulation trace); Correlation Coefficient (*i.e.* their correlation to other assertions); IS measure (*i.e.* assertions which have a low frequency of occurrence but highly correlated to other assertions), etc.

This paper exceeds the state-of-the-art, first, by introducing an automated methodology to extract a set of verification assertions from simulation trace thus overcoming the shortcomings of manual definition. Second, the methodology specifies an environment in terms of exhaustive valid simulation traces which serves as a complete verification environment for the assertion miner. Consequently, the generated assertions represent a set of valid assertions that lead to a significant reduction in the number of generated assertions in comparison with the state-of-the-art. Third, the assertion miner is enhanced by an algorithm that provides a compact and accurate assertion set with 100% fault coverage when mutant analysis has been performed.

In addition, in the context of assertion selection, the flow is equipped with an external feature that is able to communicate with the users to understand their needs during the verification process. This feature allows selecting a specific set of assertions related to an IP core or individual variable of the design, thus offering the ability of assertion selection for further design analysis to the users.

Thus, the contributions of this work are listed as follows:

- we propose a methodology for automated mining of compact assertion sets which is based on a complete verification environment of exhaustive valid simulation traces,
- we propose an assertion miner to automatically gen-

2021 IEEE Nordic Circuits and Systems Conference (NorCAS) | 978-1-6654-0712-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/NORCAS53631.2021.9599865

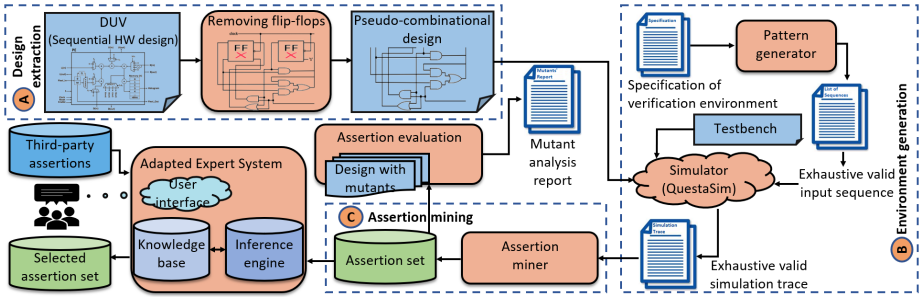


Figure 1: Flow of the proposed methodology

erate a human-readable assertion set from the simulation traces of hardware designs,

- the methodology yields more accurate assertion sets than existing approaches, achieving 100% mutant coverage for the considered examples,
- an added external feature to the methodology is that it can communicate with users to select an assertion set based on the user's need for further design analysis and debugging.

The remainder of the paper is organized as follows. The proposed methodology is discussed in Section II. Section III presents the experimental results and a comparison of the proposed method with the other related works and finally, Section IV concludes the paper.

II. Proposed Methodology

In this section, we propose the methodology for automated mining of compact and accurate assertion sets. Fig. 1 demonstrates in detail the three main steps which lead to generating the assertion set, *i.e.*, A) Design extraction, B) Environment generation, and C) Assertion mining. Moreover, the figure shows an external feature that uses an adapted Expert System. Furthermore, an assertion evaluation phase based on mutant analysis is applied for assessing the quality of the generated assertions. The flow of each step is described in the following:

A) Design extraction: This step deals with converting sequential designs to pseudo-combinational ones by removing flip-flops in order to enable a complete analysis of the assertion space within a single clock cycle.

B) Environment generation: In this step, with analyzing specifications of design under verification, a pattern generator is led to extract an exhaustive valid input sequence. Thereby, it prevents the flow to deal with the exhaustive input sequences, as well as with any invalid input combinations. This step is an essential part of the flow since generating an assertion set from the exhaustive valid input sequences provides a complete verification environment for the assertion mining step.

C) Assertion mining: This step is in charge of mining the assertion set. An algorithm is presented which mines the assertion set directly from the output of the previous step.

Moreover, an adapted Expert System has been used to communicate with users to provide a selected assertion set based on the user's requests. In the following, the description of these components is detailed.

A. Design extraction

In order to prepare the suitable inputs for the proposed assertion miner, the pseudo-combinational designs are derived out from the original sequential designs. This is performed by removing the flip-flops and converting them to pseudo primary inputs and pseudo primary outputs [23]. Step A in Fig. 1 represents this flow.

In the following, in the illustrative case study, application of this conversion is presented. As a case study, we have considered an open-source 5-port NoC router Bonfire [23]. A high-level overview of this router is illustrated in Fig. 2.1. As shown in this figure, the router consists of an input Buffer in form of First-In-First-Out (FIFO), LBDR, Arbiter, Crossbar, and an output Buffer.

The router has 5 input/output ports, four of which (North – N, East – E, South – S, West – W) are connected to each cardinal direction. The last port (Local (L)) is connected to the local processing element. Besides, in the router, packets are sent in form of flits, and each flit is composed of header flit, and tail flit, as well as body flit(s).

LBDR and Arbiter (demonstrated in a gray box in Fig. 2.1) are the control parts of the router and the rest are considered as the datapath. The control part controls all the main procedures of the router. Generally, the control part of any design is considered as the critical and the hard-to verify elements since the program flow of the design is maintained by it. For this reason, LBDR and Arbiter are specifically targeted for the proposed workflow.

Fig. 2.2 and 2.3 demonstrate these components after conversion to pseudo-combinational version. As shown here, this is performed by removing the flip-flops and converting their outputs and inputs to pseudo primary inputs and pseudo primary outputs, respectively. This allows deriving of assertions corresponding to a form A -> next C, where A is the antecedent and C is the Consequent of the assertion.

In the next step, it is described how for the above components, a set of exhaustive valid simulation traces are generated to derive the complete verification environment for the assertion miner.

B. Environment generation

In this step, a Pattern generator is developed to extract a set of exhaustive valid input sequences. Therefore, the specifications of the verification environment of LBDR and Arbiter, more specifically ELBDR and SArbiter since they

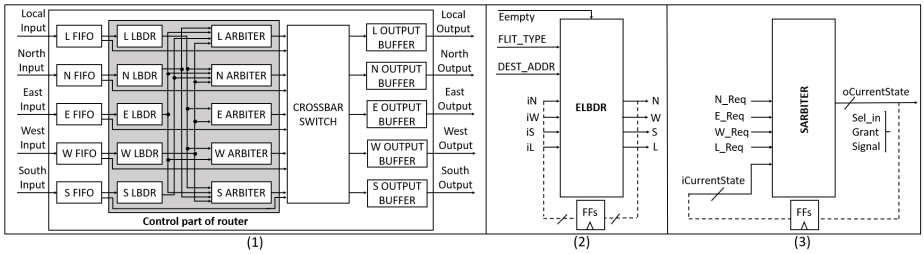


Figure 2: (1) High-level overview of the router (2) ELBDR pseudo-combinational design (3) SArbiter pseudo-combinational design

have one of the most connected signals are studied. These specifications guide the Pattern generator to extract the set of exhaustive valid input sequences. These sequences are then fed to the simulator to produce a set of exhaustive valid simulation traces. Step B in Fig. 1 demonstrates this flow.

As shown in Fig. 2.2 and Fig. 2.3, ELBDR's output port signals are the N, W, S, and L signals, while for SArbiter, request and grant signals exist for the N, E, W, and L. In ELBDR, 3 bits are considered for the flit_id (FLIT_TYPE), and 4 bits for the destination address (DEST_ADDR). Moreover, an Eempty bit that is coming from the East input buffer is considered for ELBDR. The other inputs of ELBDR (iN, iW, iS, and iL) are one bit. For SArbiter, there is one bit input for any of the N_Req, E_Req, W_Req, L_Req, as well as 5 bits for any of the iCurrentState and oCurrentState.

Based on the specifications, two different conditions should be considered for ELBDR to lead the Pattern generator for generating the exhaustive valid input sequences:

- if Eempty is equal to 1 (East input is empty):
  - the input ports, (*i.e.* iN, iW, iS and iL) can have any values,
  - flit_id (FLIT_TYPE in Fig. 2.2) can have any value,
  - destination address (DEST_ADDR) can have any value
- if Eempty is equal to 0 (East input is not empty):
  - iN, iW, iS and iL must be one-hot (*i.e.* at the same time only one of the inputs could be 1),
  - flit_id (FLIT_TYPE) is one-hot (*e.g.* Header = 001, Body = 010, Tail = 100),
  - the outputs of ELBDR will always be one-hot (assuming that the routing algorithm is XY and we can only go from East to North, South, West, or Local. And they cannot be active at the same time),

Initially, the number of sequences for the exhaustive simulation trace was equivalent to 2^{12} = 4096 for ELBDR due to the fact that the number of input bits is equal to 12. After feeding the Pattern generator based on the specifications, the number of sequences for the simulation trace is reduced to 2144. As a result, the invalid conditions from the exhaustive input sequences are removed and filtered out.

Similarly for SArbiter, the specifications are studied to guide the Pattern generator. The specification is as follows:

- only one input can be granted at the same time. Hence, the grant vector (iCurrentState) state is defined as one-hot,
- N_Req, E_Req, W_Req, and L_Req, can have any values, (because SArbiter can get requests from different inputs except the South port),

Initially, the number of sequences for the exhaustive simulation trace was equivalent to 2^9 = 512 for SArbiter due to the fact that the number of input bits is equivalent to 9. After guiding the Pattern generator according to the specification, the number of sequences for the simulation trace is reduced to 80.

It is notable that this section adapted the idea of producing exhaustive valid input sequences from the work in [23]. However, the proposed methodology markedly improved the work in [23], as it developed an automatic tool to generate the valid exhaustive sequences based on the specifications. On the contrary, the work in [23], manually applies a set of filters on the exhaustive input sequence to gain the valid one. Thereby, the proposed method considerably reduced the time and cost in terms of memory requirement in comparison with [23].

The generation of exhaustive valid simulation trace is significantly pivotal since it prevents the assertion miner from being in non-realistic conditions and only the functionally of feasible values are retained. Thereby, the miner generates a complete assertion set that is based on valid scenarios. In the following the assertion miner's algorithm is detailed.

C. Assertion mining

As demonstrated in step C of Fig. 1, the assertion miner generates a set of assertions from the output of the previous step, *i.e.* exhaustive valid simulation trace. This step is one of the fundamental components of the proposed methodology since it is in charge of discovering the relations between the primary inputs and primary outputs of the design. The assertions are generated according to the proposed Algorithm 1. As it can be seen, the algorithm is divided into three phases.

Phase 1 starts by reading the sequences of simulation trace to discover the similarities among them. Note that N in the algorithm indicates the number of sequences in the simulation trace file. Besides, input() and output() methods show the input and output parts of a sequence. $length()$ method indicates the length of a sequence. In this phase, all the sequences are compared. For every comparison, each bit in a sequence is compared with its corresponding bit in the next sequence. Lines 5 to 8 of Algorithm 1 describe how this comparison is performed. If both bits are equal,

the value of the bit is kept. Otherwise, it is considered as a 'don't care' status and the result is stored in the variable 'result1'. Note that the comparisons in which all the input or output bits are determined as a 'don't care' are not analyzed in the next phase. Lines 11 and 12 show the input and output parts of a sequence.

Phase 2 aims at finding similarities among the outcomes of Phase 1. At first, the input parts of all the results of Phase 1 are compared (Looking at Fig. 3 to distinguish the input part and output part of a sequence). If the bits in the input part are equivalent, the output part of the sequences is checked.

To achieve this, similar to Phase 1, each bit is compared with its corresponding bit in the other sequence. If they are equal, the value of the bit is stored in the variable 'result2'. Otherwise, 'don't care' status is considered. On the contrary, according to lines 29 and 30 of Algorithm 1, if a sequence is not equal to any other sequences, it will go to the next phase of the algorithm.

Note that in both Phase 1 and 2, there might be some results where their input/output parts all are don't care. In such cases, those results should be discarded. This is described in lines 13, 14, 35, and 36 of the algorithm.

Finally, Phase 3 decides which logic operator should be inserted among the variables of Result2. The mechanism is as follows: A logical AND is inserted between the bits of the input part of a Result2. Moreover, it checks if a bit in the output part is equal to its corresponding bit in another generated Result2. In this condition, an OR will be added between those cases.

Fig. 3 demonstrates several examples from C17 design for each phase of algorithm 1. As shown in this figure, Phase 1 starts by reading and comparing all the sequences of the simulation trace. For a better illustration, five different examples are shown for this phase. Each pair of sequences is shown by A and B.

After applying the first phase of Algorithm 1, the EX1, EX2, EX3, EX4 and EX5 are calculated which are equal to 0-01- 0-, 0-01- 0-, 00-1 01, 00-1 01 and, 11— –, respectively. In these examples, the first part of bits (*e.g.,* 0-01- in EX1) indicates the input bits, and the second part (*e.g.,* 0- in EX1), indicates the output bits. Moreover, each bit represents a variable name of the simulation trace. For instance, in 0-01-0-, the first bit represents the i0 variable, the second one represents the i1 variable, and finally 0- represent out0 and out1 variables, respectively.

In these examples, EX1 and EX2 create Case 1, and EX3 and EX4 create Case 2. Subsequently, Case 1 and Case 2 go to Phase 2 since each pair has the same inputs. On the contrary, EX5 will be discarded since its output is equal to '-', *i.e.,* 'don't care'.

As said and as shown in Fig. 3, Case 1 and Case 2 are entered to Phase 2. Since the input of both cases is equivalent, their outputs should be analyzed according to the algorithm. As a result, EX6 and EX7 are calculated.

Finally, in Phase 3 the mined assertions have been generated. Going back to the illustrative example in Fig. 3, Phase 3 combines EX6 and EX7 on the basis of finding similar output bits. Since the first bit in the output part of both EX6 and EX7 are equal to 0, their input bits have ORed with each other *i.e.,* $0-01- \parallel 00-1-$. To this end,

Algorithm 1 Assertion Miner

```
 1: index = 0                                                          ▷ Phase 1
 2: for i in 1 to N-1 do
 3:   for j in i+1 to N do
 4:     for k in 1 to length(seq) do
 5:       if (seq[i,k] == seq[j,k]) then
 6:         Result1[index,k] = seq[i,k]
 7:       else
 8:         Result1[index,k] = 'don't care'
 9:     index = index + 1
10: for i in 1 to length(Result1) do
11:   in = input(Result1[i])
12:   out = output(Result1[i])
13:   if (in.allBits[i] == 'don't care') or (out.allBits[i] == 'don't care')
    then
14:     discard(Result1[i])
15: index = 0                                                          ▷ Phase 2
16: for i in 1 to length(Result1)-1 do
17:   cnt = 0
18:   for j in i+1 to length(Result1) do
19:     if (input(Result1[i]) == input(Result1[j])) then
20:       input(Result2[index]) = input(Result1[i])
21:       for k in 1 to length(output(Result1[i])) do
22:         if (output(Result1[i])[k] == output(Result1[j])[k] then
23:           output(Result2[index])[k] = output(Result1[i][k])
24:         else
25:           output(Result2[index])[k] = 'don't care'
26:       index = index + 1
27:     else
28:       cnt = cnt + 1
29:       if (cnt == length(Result1)-i)) then
30:         Result2[index] = Result1[i]
31:         index = index + 1
32: for i in 1 to length(Result2) do
33:   in = input(Result2[i])
34:   out = output(Result2[i])
35:   if (in.allBits[i] == 'don't care') or (out.allBits[i] == 'don't care')
    then
36:     discard(Result2[i])
37: temp = ' '                                                         ▷ Phase 3
38: index = 0
39: for i in 1 to length(Result2)-1 do
40:   temp = AND(temp, input(Result2[i]))
41:   for j in i+1 to length(Result2) do
42:     for k in 1 to length(output(Result2[i])) do
43:       if (output(Result2[i])[k] == output(Result2[j])[k] then
44:         temp = OR(temp, AND(input(Result2[j])))
45:   Result3[index] = temp
46:   index = index + 1
```

with looking at the name of the variables in the simulation trace it is clear that the extracted assertion is equal to $(\sim i0 \&\& \sim i2 \&\& i3) \| (\sim i0 \&\& \sim i1 \&\& i3) -> \sim out0$.

D. Assertion evaluation

This phase is performed to assure the accuracy and validity of the mined assertion sets. According to Fig. 1, the input of this phase is the generated assertion set which is provided by the assertion miner and the output of this phase is a detailed report on the effectiveness of the assertions in detecting the injected mutants.

We implemented the mutant injector tool according to the details provided in [24]. Note that this mutant injector

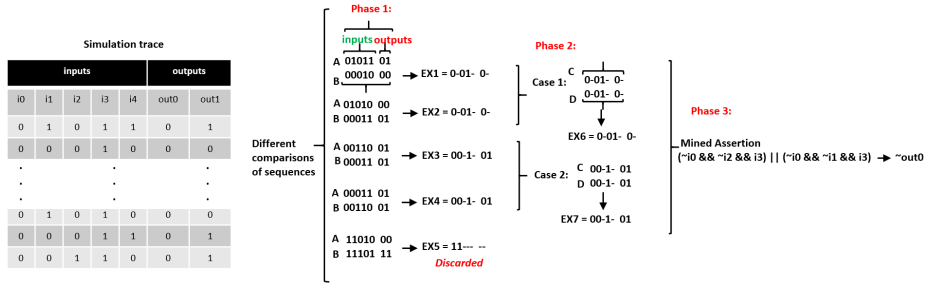


Figure 3: Three phases of the assertion miner's algorithm

Table I: Description of injected mutants.

Mutation operator types	List of operators
arithmetic operators	+, -, *, /, %
relational operators	==, !=, >, <, >=, <=
logical operators	&&, ||
assignment operators	+=, -=, *=, /=, %=, =
unary operators	+, -, ~,!
bitwise operators	<<, >>, &, |, ^
bitwise assignment operators	<<=, >>=, &=, |=, ^=

is complete in sense of converting all different types of operators to create mutants. In Table I, the details about these injected mutants in the benchmarks have been presented. The column 'Mutation operator types' demonstrates different types of the operators that have been modified by the mutants. Any of the operators in each row of the column 'List of operators', has been changed to the others. In addition to these mutants, we randomly converted all 0 bits to 1 and vise versa. As mentioned above, this section is mainly for proving the accuracy of the mined assertions. Thus, the reports of this phase is discussed in the Experimental result *i.e.*, Section III.

E. Adapted Expert System

In this phase, a feature is presented that provides the methodology with the ability to communicate with the users. In this way, verification engineers can request a specific set of assertions related to a specific module or variable under analysis that meets their needs and interests.

In other words, this feature assists the user by discarding unnecessary assertions and provide a compact assertion set associated with the module under analysis. Correspondingly, this leads to an easier design's behavior understanding and a faster bug localization.

The proposed feature adapts an Expert System [25] for the purpose of this work. In general, any Expert System has two main components which are a Knowledge base and an Inference engine [26]. The former consists of Facts and the latter is based on a set of Rules. With the help of the Facts stored in the knowledge base and Rules in the inference engine, the Expert System provides the selective set of assertions based on the users' needs.

The details about the Facts and Rules are provided in the following. As the proposed Expert System has been implemented in CLIPS language, (which is a specific tool and language for implementing Expert Systems) the descriptions and the Listings in the following subsections have been written in CLIPS syntax.

1) Facts and Rules in the adapted Expert System:

Explanation 1: ***FACT***s, are a formal form of describing knowledge and information required to solve a specific problem. Generally, these Facts are gathered from the knowledge of the experts in the domain related to that specific problem. For instance, in the real world *headache* and *fever* are two symptoms of flu that are reported by the experts in the domain *i.e.*, medical doctors. Thereby, to prepare the knowledge for an Expert System, *headache* and *fever* are stored as two Facts for detection of flu. We have adapted these concepts for the proposed flow.

Listing 1: Example of a Fact

```
(deffacts minedFacts
   (f1 !i[0]) (f2 !i[2]) (f3 i[3]) (f4 !i[1]))
```

In the proposed adapted Expert System, these Facts mainly extract from the left part of the implication of each mined assertion. As an illustrative example, we can consider the generated assertion in Phase 3 of Fig. 3.

As can been seen, the left part of the implication for this assertion is equivalent to $(\sim i0 \&\& \sim i2 \&\& i3) \| (\sim i0 \&\& \sim i1 \&\& i3)$. Correspondingly, the Facts related to this assertion has been presented in Listing 1.

In this listing, each Fact is written in CLIPS syntax and represented by i[n] which demonstrates the name of the variable i, and its corresponding bit "n". For instance, f1 means that the value for input variable i[0] is equal to 0.

Similarly, for the other Facts, f2, f3, and f4 the variables and their values are i[2] = 0, i[3] = 1 and i[1] = 0, respectively.

Explanation 2: ***RULE,*** the structure of a Rule is similar to the $IF-THEN$ statements. If the first part of a Rule (IF), matches with one or more stored Facts in the knowledge base of the Expert System then the second part of the Rule ($THEN$) will be fired and the Rule will produce the result. For example, in the real world, IF the symptoms in a person are *headache* and *fever*, $THEN$ we can realize that the person has gotten flu. Accordingly, this concept has been adapted for the proposed flow.

Listing 2 demonstrates the Rule related to the assertion in Phase 3 of Fig. 3 in CLIPS syntax. This Listing represents that IF (i[0] = 0 && i[2] = 0 && i[3] = 1) || (i[0] = 0 && i[1] = 0 && i[3] = 1) $THEN$ $out0$ is always 0. Thus, if the users are interested in selecting the assertions related to $out0$, this rule causes the selection of the mentioned assertion.

Listing 2: Example of a Rule

```
(or(and(i[0]=0)(i[2]=0)(i[3]=1))(and(i[0]=0)(i
    [1]=0)(i[3]=1)) => ~out0)
```

The above described the basic components of an Expert System and the process it is adapted for purpose of this work. In the following, we demonstrate further usage of this feature. According to Fig. 1, the adapted Expert System can be fed by the assertions from the proposed assertion miner, as well as assertions provided by third parties for a specific design. This can enhance the methodology by two advantages.

Firstly, in cases that the generated assertions by an assertion miner cannot completely cover all the functionality of the design, the adapted Expert System can learn from what assertions have been stored by the third parties in its knowledge base and generate a new set of assertions to increase the coverage. Secondly, the adapted Expert System can reduce the redundancy of the assertion sets after the selection phase. Formula (1), formula (2), and formula (3) represent the reduction mechanism. As can be seen here, m_1, m_2, ..., m_n is repeated in both formula (1) and formula(2) which is omitted in formula (3). With this straightforward mechanism, this feature can prevent this kind of redundancy.

$$(a_1 \;\&\&\; \ldots \;\&\&\; a_n) \;||\; \ldots \;||\; (m_1 \;\&\&\; \ldots \;\&\&\; m_n) \;|->\; \delta, \;\; n \geqslant 0 \quad (1)$$

$$(m_1 \;\&\&\; \ldots \;\&\&\; m_n) \;||\; \ldots \;||\; (z_1 \;\&\&\; \ldots \;\&\&\; z_n) \;|->\; \delta, \;\; n \geqslant 0 \quad (2)$$

$$(1),(2) => ((a_1 \;\&\&\; \ldots \;\&\&\; a_n) \;||\; \ldots \;||\; (m_1 \;\&\&\; \ldots \;\&\&\; m_n)) \;||\; \ldots \;||\; (z_1 \;\&\&\; \ldots \;\&\&\; z_n)) \;|->\; \delta, \;\; n \geqslant 0 \quad (3)$$

2) User interface in the adapted Expert System:

Users can interact with the system through the user interface. A user interface generally asks questions from the users to understand their needs. To this extent, the expert system is able to select an assertion set for a specific module or an output variable. For instance, the Rule in Listing 2 is related to the case that the user's desire is generating assertion for $\sim out0$ output.

III. Experimental Results

The efficiency of the proposed methodology has been studied on the mentioned NoC router Bonfire in Section II-A, and its two main components, *i.e.,* ELBDR, and SArbiter [27]. The programming languages that have been used in this work are SystemVerilog, Python, and CLIPS. The benchmarks are developed in SystemVerilog. Assertion miner is implemented in Python. Finally, CLIPS is the suitable language for the Expert System.

Furthermore, with the aid of the adapted Expert System, the generated assertions can be provided in two different syntaxes which are SystemVerilog Assertions (SVA), or Property Specification Language (PSL). This can be helpful in analyzing the designs which are programmed in any of these languages.

Table II presents the experimental results. Column 'Benchmarks' represents the name of benchmarks, i.e. SArbiter and ELBDR. Column 'Metrics' contains four different metrics i.e. '#mutants', '#assertions', '%detected mutants', and 'execution time', which report the number of injected mutants, the number of generated assertions, the number of detected mutant, and the execution time, respectively. Columns 'Approach 1' and 'Approach 2' show the results for two different assertion miners represented in [14]. Finally, 'Proposed methodology' refers to the approach of this work.

Table II: Comparison between the approaches in [14] and the proposed method.

Benchmarks	Metrics	Approach 1	Approach 2	Proposed approach
SArbiter	#mutants	73	73	73
	#assertions	449	689	10
	%detected mutants	97%	99%	100%
	execution time	2h5m	2h22m	20s.65ms
ELBDR	#mutants	76	76	76
	#assertions	37	36	8
	%detected mutants	100%	100%	100%
	execution time	20m	19m	13m

As for SArbiter, the number of the injected mutants is equal to 73. Note that, these mutants are injected according to the details provided in Subsection II-D. The number of generated assertions are 449, 689, and 10 for the three approaches, respectively. As for fault coverage, the results in percentage are 97, 99, and 100, respectively. Finally, the execution time for the three approaches is as follows 2h5m, 2h22m, and 20s65ms.

For ELBDR, the number of the injected mutants is equal to 76. The number of generated assertions is 37, 36, and 8 for the three approaches, respectively. Moreover, for fault coverage, the results for all the three approaches are 100%. Finally, the execution time for the three approaches is as follows 20m, 19m, and 13m.

As shown above, the proposed methodology has advantages, since it generates a considerably smaller number of assertions *i.e.,* 10 and 8 for SArbiter and ELBDR, respectively. In contrast, these numbers for Approach 1 are 449 and 37, and for Approach 2 are 689 and 36. The mutant coverage is slightly increased for the proposed methodology in comparison with the other approaches for SArbiter, and remained steady in ELBDR. Moreover, the execution time of the proposed assertion miner has drastically reduced in comparison with Approach 1 and Approach 2. As can be seen here, it reached 20.65s and 13m for SArbiter and ELBDR, respectively. However, these numbers are 2h5m and 20m for Approach 1 and, 2h22m and 19m for Approach 2.

IV. Conclusions

This work proposed an efficient methodology for automated mining of compact and accurate assertion sets. The methodology specified an environment in terms of exhaustive valid simulation traces which served as a complete verification environment for the assertion miner. Consequently, the generated assertions represented a set of valid assertions. The assertion miner was further enhanced by an algorithm that provides a compact and accurate assertion set. To that end, the experimental results showed that the proposed approach generated significantly more compact assertion sets than the state-of-the-art while achieving 100% fault detection in terms of the injected mutants.

Acknowledgement

This work in the project "ICT programme" was supported by the European Union through European Social Fund.

REFERENCES

[1] M. Boulé and Z. Zilic, *Generating Hardware Assertion Checkers: For Hardware Verification, Emulation, Post-Fabrication Debugging and On-Line Monitoring*. Springer Publishing Company, Incorporated, 2008.

[2] X. T. Ngo, J. Danger, S. Guilley, Z. Najm, and O. Emery, "Hardware property checker for run-time hardware trojan detection," in *2015 European Conference on Circuit Theory and Design (ECCTD)*, 2015, pp. 1–4.

[3] S. Katz, O. Grumberg, and D. Geist, "Have I written enough properties? — A method of comparison between specification and implementation," in *Proc. of ACM CHARME*, 1999, pp. 280–297.

[4] M. Jenihhin, X. Lai, T. Ghasempouri, and J. Raik, "Towards multidimensional verification: Where functional meets non-functional," in *2018 IEEE Nordic Circuits and Systems Conference (NORCAS):*, Oct 2018, pp. 1–7.

[5] X. Lai, A. Balakrishnan, T. Lange, M. Jenihhin, T. Ghasempouri, J. Raik, and D. Alexandrescu, "Understanding multidimensional verification: Where functional meets non-functional," *Microprocessors and Microsystems*, vol. 71, p. 102867, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933119300250

[6] J. Malburg, T. Flenker, and G. Fey, "Property mining using dynamic dependency graphs," in *Asia and South Pacific Design Automation Conference*, Jan 2017, pp. 244–250.

[7] G. Martino and G. Fey, "Syntax-guided enumeration of temporal properties," in *2019 Forum for Specification and Design Languages (FDL)*, 2019, pp. 1–8.

[8] A. Danese, N. Dalla Riva, and G. Pravadelli, "A-team: Automatic template-based assertion miner," in *Design Automation Conference, 2017 54th ACM/EDAC/IEEE*. IEEE, 2017, pp. 1–6.

[9] T. Ghasempouri, A. Danese, G. Pravadelli, N. Bombieri, and J. Raik, "Rtl assertion mining with automated rtl-to-tlm abstraction," in *2019 Forum for Specification and Design Languages (FDL)*, 2019, pp. 1–8.

[10] S. Vasudevan, D. Sheridan, S. Patel, D. Tcheng, B. Tuohy, and D.Johnson, "Goldmine: automatic assertion generation using data mining and static analysis," in *Proc. of ACM/IEEE DATE*, 2010, pp. 545–548.

[11] S. Hertz, D. Sheridan, and S. Vasudevan, "Mining hardware assertions with guidance from static analysis," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 32, pp. 952–965, 2013.

[12] M. W. Anwar, M. Rashid, F. Azam, A. Naeem, M. Kashif, and W. H. Butt, "A unified model-based framework for the simplified execution of static and dynamic assertion-based verification," *IEEE Access*, vol. 8, pp. 104 407–104 431, 2020.

[13] T. Ghasempouri, J. Malburg, A. Danese, G. Pravadelli, G. Fey, and J. Raik, "Engineering of an effective automatic assertion-based verification platform," in *6th Workshop on Design Automation for Understanding Hardware Designs (DUHDe 2019)*, March 2019, pp. 557–562. [Online]. Available: https://elib.dlr.de/127128/

[14] T. Ghasempouri, J. Malburg, A. Danese, G. Pravadelli, G. Fey, and J. Raik, "Engineering of an effective automatic dynamic assertion mining platform," in *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*, 2019, pp. 111–116.

[15] S. Vasudevan, D. Sheridan, S. Patel, D. Tcheng, B. Tuohy, and D. Johnson, "Goldmine: automatic assertion generation using data mining and static analysis," in *Proc. of ACM/IEEE DATE*, 2010, pp. 626–629.

[16] S. Hertz, D. Sheridan, and S. Vasudevan, "Mining hardware assertions with guidance from static analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, pp. 952–965, 2013.

[17] T. Ghasempouri and G. Pravadelli, "On the estimation of assertion interestingness," in *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2015, pp. 325–330.

[18] T. Ghasempouri, S. Payandeh Azad, B. Niazmand, and J. Raik, "An automatic approach to evaluate assertions' quality based on data-mining metrics," in *2018 IEEE International Test Conference in Asia (ITC-Asia)*, 2018, pp. 61–66.

[19] R. Hariharan, T. Ghasempouri, B. Niazmand, and J. Raik, "From rtl liveness assertions to cost-effective hardware checkers," in *2018 Conference on Design of Circuits and Integrated Systems (DCIS)*. IEEE, 2018, pp. 1–6.

[20] D. Pal, S. Offenberger, and S. Vasudevan, "Assertion ranking using rtl source code analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, pp. 1711–1724, 2020.

[21] S. Hertz, D. Sheridan, and S. Vasudevan, "Mining hardware assertion with guidance from static analysis," *IEEE Trans. on CAD*, vol. 32, 2013.

[22] G. Fey, T. Ghasempouri, S. Jacobs, G. Martino, J. Raik, and H. Riener, "Design understanding: From logic to specification*," in *2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2018, pp. 172–175.

[23] P. Saltarelli, B. Niazmand, R. Hariharan, J. Raik, G. Jervan, and T. Hollstein, "Automated minimization of concurrent online checkers for network-on-chips," in *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2015.

[24] U. Repinski, H. Hantson, M. Jenihhin, J. Raik, R. Ubar, G. Di Guglielmo, G. Pravadelli, and F. Fummi, "Combining dynamic slicing and mutation operators for esl correction," in *2012 17th IEEE European Test Symposium (ETS)*, 2012, pp. 1–6.

[25] E. T. Ogidan, K. Dimililer, and Y. K. Ever, "Machine learning for expert systems in data analysis," in *2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, 2018, pp. 1–5.

[26] R. Kerr and R. Ebsary, "Implementation of an expert system for production scheduling," *European Journal of Operational Research*, vol. 33, pp. 17–29, 1988. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377221788902500

[27] "Project Bonfire Network-on-Chip," https://github.com/Project-Bonfire/Bonfire_handshake, 2017.

Appendix 7

VII

H. Rostami, M. Hosseini, A. Azarpeyvand, MRH. Iman, and T. Ghasempouri, "Automatic High Functional Coverage Stimuli Generation for Assertion-based Verification," *The 30th IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS'24)*, Rennes, Brittany, France, 2024.

Automatic High Functional Coverage Stimuli Generation for Assertion-based Verification

Hossein Rostami[1], Mostafa Hosseini[1], Ali Azarpeyvand[1,2], Mohammad Reza Heidari Iman[2], and Tara Ghasempouri[2]

[1]University of Zanjan, Zanjan, Iran
[2]Tallinn University of Technology, Tallinn, Estonia
[1]{rostami.hossein, mostafa.hosseini, azarpeyvand@znu.ac.ir}
[2]{ali.azarpeyvand, mohammadreza.heidari, tara.ghasempouri@taltech.ee}

***Abstract*—Assertion-based verification is a promising method that uses predefined rules, known as assertions, to check the functionality of hardware designs. The manual assertion definition is time-consuming and requires expert knowledge. Automatic assertion mining is gaining acceptance as a trustworthy method for assertion definition. Some automatic assertion miners extract assertions from simulation traces of the design, but the quality of mined assertions depends on the coverage of the stimuli used to generate the traces. Existing stimuli generation methods are either random or exhaustive. A random approach can only cover some design behavior, resulting in incomplete assertions. On the other hand, an exhaustive approach can cover all the design behavior but produces lengthy simulation traces that cause a high overhead for the miner. We propose a novel approach for stimuli generation based on constraint random verification. A set of user-defined metrics then examines the generated stimuli to measure how much of the design specification has been exercised by the verification environment. Our approach uses a coverage model that defines, collects, and analyzes the design's functionalities and identifies the gaps in the verification. The assertions generated by the proposed method have been compared with a well-known assertion miner, GoldMine. The result showed that our method detects 20.63% more faults in the design than GoldMine in a shorter time. Moreover, it produces assertions that are about 79% more effective.**

***Index Terms*—Assertion-based verification, constraint random verification, Simulation trace, Stimuli**

I. Introduction

Verification methods are employed to confirm that a design precisely aligns with its defined requirements and specifications. By applying these methods, developers can identify discrepancies between the design's intended behavior and its actual behavior under various conditions, thereby reducing the risk of failures and enhancing the quality of the final product. Due to digital systems' growing complexity and the importance of time to market, hardware verification has become the biggest bottleneck in creating new electronic products [1].

Assertion-based verification (ABV) [2] has emerged as one of the most effective techniques for validating "pre-Silicon" designs [1] among all the available methods. Assertions allow for the explicit specification of desired behaviors and properties, providing a direct means to test design intents and ensuring correct behavior under all conditions. They enable continuous monitoring of a design's internal workings, not just its inputs and outputs, allowing for the immediate detection and easier debugging of issues and enhancing observability. Moreover, assertions automate and enhance the verification process, acting as dynamic checklists for constant evaluation against expected behaviors, thus significantly reducing manual effort.

Traditionally, the definition of assertions is a fully manual effort that is time-consuming and error-prone, and the quality of them depends on human expertise [3]. To address the challenges associated with manually defining assertions, various strategies, approaches, and tools have been developed to automate the extraction of assertions directly from the Design Under Verification (DUV). This automation serves as a supplementary method, enhancing the traditional manual process by leveraging algorithms and software that can analyze the DUV and automatically generate assertions based on its behavior [4].

Two methods are commonly used for automatically extracting assertions from the DUV implementation [5]. One approach involves the static analysis of the DUV's source code [6], which focuses on automatically generating formal properties for a given design. In contrast, the second approach analyzes simulation traces generated by simulating the DUV using stimuli [7]. While static methods face scalability issues, dynamic assertion mining is more computationally efficient. Independently from the adopted techniques, mined assertions can be compared with design intents to discover unexpected behaviors implemented in the DUV, to confirm that relevant behaviors are actually implemented, and for documentation purposes [4].

Dynamic assertion mining is a popular method for extracting assertions from a DUV because it is more scalable than other methods. However, the effectiveness of this method depends on the quality and diversity of the stimuli used to generate simulation traces. It is essential to use appropriate stimuli to

979-8-3503-7055-3/24/$31.00 ©2024 IEEE

ensure a comprehensive DUV exploration [8].

In recent verification methodologies, constrained random verification (CRV) has emerged as a preferred approach for generating numerous stimuli while adhering to specific constraints [9-10]. These constraints serve the purpose of ensuring that the generated input stimuli follow particular design requirements, encompassing considerations such as data ranges, timing constraints, and interface protocols. Through developing many stimuli, CRV contributes to thoroughly examining the design, containing many potential scenarios [11].

The underlying concept of CRV is grounded in the notion that stimuli can adopt arbitrary values. Through randomization, these elements can accept different values in each verification iteration. Constraints play a pivotal role in this mechanism, controlling the values of these randomly generated elements.

During the generation of stimuli using CRV and subsequent simulation, it is essential to monitor metrics related to the DUV to ensure that the design is being examined as expected. Functional coverage, a metric manually defined by engineers focusing on corner cases, is a valuable measure for evaluating the thoroughness of stimuli in verification processes [12]. Prioritization of corner cases is crucial for verification. These cases represent rare or extreme conditions that standard tests may not cover. Functional coverage addresses these critical situations, ensuring a more comprehensive evaluation.

This paper introduces a flexible method for rapidly generating high-functional coverage stimuli to provide dynamic assertion miners with appropriate simulation traces. This accelerates the automatic ABV process. Furthermore, the paper suggests a way to evaluate the assertions using these stimuli. The main contributions of this research are outlined below:

- A novel approach is introduced to generate high-functional coverage stimuli. This method stands out for its swiftness in producing stimuli that thoroughly exercise DUV's functionalities.
- The proposed method for generating simulation traces is scalable since it uses the CRV technique and introduces a set of metrics for determining functional coverage.
- The proposed method improves assertion-based verification by generating high-functional coverage stimuli, consequently extracting high-quality assertions, thus reducing the time and cost of the verification process.

To the best of our knowledge, the method proposed in this paper for rapidly generating high-functional coverage stimuli represents a first in the field of automatic assertion-based verification techniques. This innovative approach is expected to significantly contribute to the efficiency and effectiveness of the verification process, marking a notable advancement in the domain.

The remainder of the paper is organized as follows: Section 2 reviews previous related works. Section 3 introduces some preliminary definitions. Section 4 provides an overview of the proposed method. Section 5 reports experimental results. Finally, section 6 concludes the paper.

II. RELATED WORK

Different methodologies for assertion mining have been developed. In [13], GoldMine is designed to generate assertions directly from Register-Transfer Level (RTL) designs. It accomplishes this by employing static analysis of the DUV's source code and data mining techniques applied to the behavioral data of the design. However, when utilizing formal verification techniques like model checking within this tool, there's a notable challenge known as "state explosion" that becomes prominent with complex designs.

The research in [14] presents an automatic assertion mining tool that leverages a dynamic dependency graph to generate assertions. They could extract relationships between signals in the design by analyzing simulation traces. They utilize use cases to create simulation traces but have yet to suggest any approach to create these use cases. Additionally, they employ a model checker to verify the accuracy of their mined assertions, which necessitates access to the source code of the DUVs.

M.R. Heidari Iman et al. [15] proposed an assertion miner and evaluator. The assertion miner can generate a set of readable and compact assertions. The assertion evaluator employs a data-mining algorithm called *dominance* to assess the quality of the assertions. They use exhaustive, valid simulation traces to extract assertions dynamically. However, this method for generating simulation traces is impractical when dealing with complex DUVs and suffers from scalability issues. Furthermore, no approach for verifying mined assertions is suggested, assuming that the mined assertions are always factual by using exhaustive valid simulation traces.

The researchers in [16]–[18] start their automatic mining flow from a set of given simulation traces. As previously mentioned, the quality of mined assertions heavily relies on the quality of simulation traces. However, the authors did not propose any approaches for generating appropriate simulation traces.

Although several approaches to assertion mining and validation have been proposed, a dependable and rapid approach for generating stimuli to provide dynamic assertion miners with high-quality and valid simulation traces has yet to be developed.

III. PRELIMINARIES

In this section, we'll briefly explain the definitions and concepts used in this article to help the readers better understand the approach.

Definition 1: An **assertion** is a logic formula that describes the behavior of the design through temporal operators that must hold or must become true during the execution of the design [16].

Definition 2: A **seed** is a numerical value that initializes the generation of a sequence of pseudo-random numbers. To obtain different sequence values from a random number-generating algorithm, we must start with different seed values [20]. Initiating the process with the same seed yields an identical sequence of numbers each time.

Definition 3: **Bins** serve as fundamental elements for organizing and categorizing the various states or values that

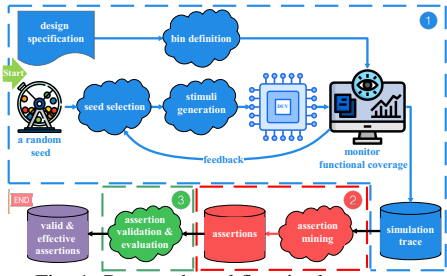


Fig. 1: Proposed workflow in three steps.

variables within a DUV might assume. These bins enable the systematic analysis of both static conditions and dynamic sequences, facilitating a comprehensive understanding of the DUV's behavior.

Definition 4: **Functional coverage** mapping each functional feature of a design to a user-defined coverpoint, in this work named as bin. A coverpoint essentially encapsulates a specific aspect of the design that requires evaluation in terms of functional coverage. Within each coverpoint is a collection of bins, each associated with either the sampled values or the transitions between values of the targeted variable.

IV. METHODOLOGY

The proposed method consists of three steps, which are illustrated in Fig. 1.

The verification process begins by creating a simulation trace, which involves providing stimuli to the DUV to initiate the simulation. The results of this simulation are then recorded in the simulation trace. Following this, an assertion miner automatically utilizes the simulation trace to generate assertions (definition 1). These newly created assertions undergo validation through simulation, and any assertions that fail the validation process are discarded and finally, we evaluate the valid assertions.

The subsequent section provides a thorough examination for a more in-depth exploration of these steps.

A. Generating Simulation Trace

The automatic generation of assertions from simulation traces begins with the creation of a simulation trace covering all expected functionalities of the DUV. At this step, bins (definition 3) are defined. They are derived from the design specifications. This foundational step is crucial as it bridges the gap between the theoretical aspects of the design and the practical components that require validation. Bins serve as pivotal markers or checkpoints, directing the process of generating stimuli.

Functional coverage (definition 4) is a measure used to evaluate the extent to which a design's intended functionalities have been tested during the verification process. This metric stands in contrast to other measures like code and toggle coverage, which respectively gauge the execution of code blocks and the changes in signal states, but may not directly reflect the testing of specific design functionalities. Functional coverage is thus regarded as a critical metric for ensuring that all specified functionalities of a design have been thoroughly tested [21].

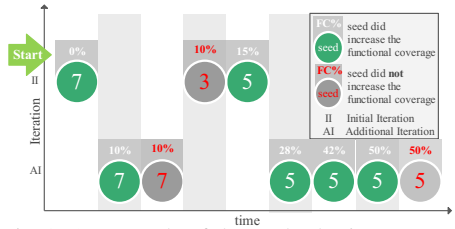


Fig. 2: An example of the seed selection process.

To achieve comprehensive functional coverage, a deliberate process of bin definition is undertaken. This process starts with the identification of the key functional requirements of the DUV, which can encompass various operational modes, states, and precise value ranges that the design is expected to handle. For each distinct requirement, corresponding bins are created to monitor these functionalities. For instance, the occurrence of a particular output signal being activated can be defined as a bin; if this specific output is triggered during the verification phase, the corresponding bin is marked as covered.

Following the definition of bins, the subsequent step is to choose a seed (definition 2). This seed is used to kick-start the CRV method. The CRV method then employs this seed to randomly generate input signal values. However, these values aren't completely random; they are confined within the bounds of predefined constraints.

The seed selection process, as shown in Fig. 2, begins with randomly selecting a seed, such as '7'. Stimuli are generated using the CRV method based on this seed, and an initial set of stimuli is created for simulation to measure functional coverage. In this example, seed '7' increases the coverage to 10%. If additional stimuli generated by seed '7' do not improve coverage beyond 10%, a new seed, like '3', is selected, and the process repeats. This methodology ensures adaptability. When a selected seed cannot produce a sequence of random numbers conducive to increased functional coverage, the system switches to a different seed. The objective is to replace an under-performing seed with a more effective one, ultimately attaining the desired functional coverage. This iterative approach persists until the specified functional coverage target is achieved, enhancing the process's efficiency to reach the desired functional coverage. A similar approach is suggested in [22]. By adjusting the testbench seed dynamically, the verification flow is dynamic, and its structure changes based on the feedback obtained from functional coverage assessments.

In conventional CRV environments, achieving comprehensive functional coverage typically involves conducting multiple independent verifications. Subsequently, the union of the results from these verifications is used to assess the sufficiency of functional coverage. In contrast, our approach autonomously identifies a complete stimulus solution to fulfill the functional coverage objectives within a single run. The current tools and infrastructures employed in functional verification tasks, such as the Universal Verification Methodology (UVM) [23] or the SystemVerilog [24] hardware verification language, require enhanced functionality to apply advanced verification mechanisms effectively. To meet this need, a new solution has been developed: the COCOTB platform [25]. This platform,

Algorithm 1: Generating Simulation Traces

```
SET max-limit-iteration, initial-iteration, additional-iteration;
FC_desired ← 100;
FC_previous, FC_current, N_iterations, cnt_total, cnt_inner ← 0;
while cnt_total < max-limit-iteration do
    while FC_current < FC_desired do
        N_iterations ← initial-iteration;
        seed ← random value;
        while cnt_inner < N_iterations do
            generate stimuli (seed);
            simulate (design, stimuli);
            FC_current ← assess functional coverage;
            if cnt_inner == N_iterations then
                if FC_current > FC_previous then
                    FC_previous ← FC_current;
                    N_iterations ← N_iterations + additional-iteration;
                end if
            end if
            cnt_inner ← cnt_inner+1
        end while
    end while
    cnt_total ← cnt_total+1
end while
```

which is both free and open-source, allows for the creation of testbench environments using Python.

The process of generating simulation traces, outlined in Algorithm 1, involves a systematic approach to stimuli generation.

This algorithm initiates by setting several key parameters: max-limit-iteration which defines the maximum number of iterations allowed for the entire simulation process, initial-iteration which specifies the number of iterations for each internal cycle at the outset, additional-iteration for adding extra iterations in response to progress, and $FC_{desired}$ as the target functional coverage percentage. Additionally, it initializes various counters and trackers such as $FC_{previous}$, $FC_{current}$, $N_{iterations}$, cnt_{total}, cnt_{inner} to zero. The simulation operates within an outer loop that continues until the total iterations exceed the set limit, managing the overall simulation cycles. Nested within is an inner loop that runs until the current functional coverage meets or surpasses the desired level. For each cycle in this loop, a random seed is generated to diversify simulation inputs, and the number of iterations ($N_{iterations}$) is initially set to the initial-iteration value. During the internal iteration loop, the algorithm generates stimuli from the current seed and simulates the design with these stimuli, assessing functional coverage after each iteration. If by the end of these iterations the current functional coverage improves over the previous, the algorithm updates $FC_{previous}$ to $FC_{current}$ and increments $N_{iterations}$ by additional-iteration, fostering further exploration. Loop management involves cnt_{inner}, which tracks the iterations within the current cycle and increments until it aligns with $N_{iterations}$. If no improvement in functional coverage is noted by the end of $N_{iterations}$, the loop restarts with the same number of iterations but potentially different stimuli, influenced by a new random seed. The process outlined in the algorithm concludes when the outer loop's conditions are not met, which can occur when either the maximum number of iterations is reached or the desired functional coverage is achieved. If the desired functional coverage is not reached within the specified maximum iterations, further analysis is necessary to determine which specific functionalities were not covered and to understand why these gaps in coverage occurred.

B. Assertion mining

In the next step of our process, we begin mining assertions, employing a specialized tool as recommended in [15] to produce SystemVerilog Assertions. This tool is particularly effective in our analysis, offering significant benefits for our purposes.

This assertion mining tool scrutinizes the simulation trace created in the earlier step, searching for patterns and relationships among the signals of the DUV. Leveraging these observations, it crafts precise SystemVerilog Assertions that closely mirror the functional behaviors observed in the design.

C. Assertion validation and evaluation

In the final step of our method, we validate and evaluate the assertions acquired from the earlier mining process. The absence of exhaustive stimuli during assertion mining introduces uncertainty regarding whether the mined assertions accurately reflect the intended behavior or conditions. To address this issue we have introduced a specific validation procedure.

We avoid using formal methods for validating assertions because they need an internal view of the DUV, often unavailable with third-party intellectual property (IP). Moreover, formal methods can struggle with complex designs due to state explosion. In contrast, our method can effectively handle complex designs without requiring direct access to the design code. we employ a new set of stimuli to validate the assertions under different conditions. This approach significantly increases the likelihood of the validity of the mined assertions in diverse testing scenarios due to the randomness involved in creating the simulation trace. In this phase, we simulate the DUV using the new stimuli and concurrently check the mined assertions to detect potential counterexamples. If an assertion fails, we discard it. The validation process significantly boosts confidence in the validity of the retained assertions, regardless of the initial simulation trace used for their discovery.

In addition to the validation phase, we use mutation testing to evaluate the assertions. The starting point of this stage is the set of valid assertions and the output of this phase results in a comprehensive report detailing how effectively these assertions identify the introduced mutants. In this paper, RTL benchmarks were mutated by injecting mutations listed in Table. I. The column 'Mutation operator type' displays the many operators affected by the mutation [15]. In each row of the 'List of operators' column, any operators have been replaced with the others in the same row. In addition to these mutations, all 0 bits are randomly converted to 1 and vice versa.

TABLE I: List of Operators for Mutation [15]

Mutation operator types	List of operators
arithmetic operators	+, −, ×, /, %
relational operators	==, !=, >, <, >=, <=
logical operators	&&, ||
assignment operators	+=, − =, ×=, /=, %=, =
unary operators	+, −, ∼, !
bitwise operators	≪, ≫, &, —, ∧
bitwise assignment operators	≪=, ≫=, & =, | =, ∧ =

In our approach to mining assertions, we deliberately avoid relying on a golden model as a reference because such models

may not always be accessible. Instead, we first mine a broad set of assertions directly from the DUV and subsequently engage in a rigorous filtering process. Through this filtering, we ensure that only valid and effective assertions—those that accurately represent the intended functionality and specifications of the DUV—are retained. Following the selection of these assertions, a thorough comparison against the design specifications can be done. This crucial step verifies that the properties and behaviors identified and extracted by the automatic assertion miner align perfectly with what is desired and correct for the DUV.

V. EXPERIMENTAL RESULTS

In this section, we present the results of our approach when applied to two primary components of an NoC router [26], including an Arbiter and a Logic Based Distributed Routing (LBDR) and an Advance Bus Protocol (ABP) used in the [27]. All experiments were conducted on a virtual system with access to 4 cores of an 11th Gen Intel® Core™ @ 2.60GHz processor and 8GB of RAM, A 64-bit Ubuntu.v20.04 operating system, the COCOTB.v1.7.0 testbench environment, and the Questasim.v2021.1 simulator.

Table. II shows the number of bins determined for each benchmark. This table also reports the total time required to generate the number of stimuli for each benchmark to reach the desired functional coverage using the first step of the proposed approach.

TABLE II: Stimuli and Time Required for Different Functional Coverage Levels in Various Benchmarks

Benchmarks	**#Bins**	**Functional converge**			
		80%		**100%**	
		#Stimuli	**Time(s)**	**#Stimuli**	**Time(s)**
LBDR	4147	7230	5.92	36000	26.48
Arbiter	159	241	0.23	1336	1.13
APB	542	779	0.45	3726	2.28

As LBDR has the highest number of bins, the process of generating sufficient stimuli to cover them takes a longer duration. The reliability of functional coverage improves with an increased total number of bins. To attain 100% functional coverage, up from 80% across all benchmarks, the number of stimuli had to be quadrupled. This is due to the diminishing likelihood of covering the remaining bins with random stimuli as functional coverage rises.

A. Comparative Performance Analysis: GoldMine vs. Our Approach

We compare our methodology with the GoldMine tool, employing mutation testing to enable a meaningful and rigorous evaluation of our approach against GoldMine.

GoldMine features a random stimuli generator, where the user must specify the quantity of stimuli to be produced. However, the tool does not provide instructions on determining this quantity. To decide on the number of stimuli, we adopt a strategy where we allow GoldMine to generate an equivalent amount of stimuli as required by our method to achieve 100% functional coverage. This means we use the number of stimuli our method needs to fully cover all functional aspects as a benchmark for GoldMine.

Following the generation of stimuli, we move to the assertion validation phase. This phase is crucial as it sifts through the assertions derived from both our method and GoldMine, ensuring only valid assertions are taken forward. The next step involves the mutation testing phase, where the validated assertions from both methodologies are tested against the same set of mutants across each benchmark. This process allows us to directly compare the efficiency and effectiveness of our methodology against GoldMine in detecting and handling various mutations within the design. Table. III, providing clear insights into the comparative performance and effectiveness of our approach relative to GoldMine.

TABLE III: Comparison between the GoldMine and the proposed method.

	Proposed Methodology			**GoldMine**		
	LBDR	**Arbiter**	**APB**	**LBDR**	**Arbiter**	**APB**
#Stimuli	36000	1366	3726	36000	1366	3726
mining time (s)	**28**	**3.25**	**3.18**	6884	207	389
#mined assertions	591	638	124	1507	189	6
#failed assertions	475	257	55	314	53	0
#valid assertions	116	381	69	1193	136	6
#effective assertions	5	14	8	14	11	3
%detected mutants	62.1	**75.5**	**75**	78.9	59.3	12.5
%mutant detection effectiveness (per assertion)	**12.42**	5.39	**9.37**	5.63	5.39	4.16

Our experiment reveals that our method outperforms GoldMine significantly across three benchmarks. In the LBDR benchmark, our method completed the task in 28 seconds compared to GoldMine's 6884 seconds. Similar trends were observed in Arbiter and APB benchmarks, where our method demonstrated faster execution times of 3.25 seconds and 3.18 seconds, respectively, compared to GoldMine's 207 seconds and 389 seconds. This consistent superiority underscores the efficiency of our method.

To provide a comprehensive assessment of the overall performance improvement, we employ the concept of speedup [28], which is calculated as the ratio of GoldMine's execution time to that of our method for each benchmark. Utilizing the geometric mean to consolidate these benchmarks' speedup values, we derive an overall speedup factor, yielding a single numerical representation of the comparative efficiency.

The calculated overall speedup factor stands at approximately 5.76, signifying a substantial improvement in efficiency achieved by our method across all benchmarks. This metric serves as a quantitative indicator of the consistent superiority of our method over GoldMine, thereby substantiating its potential for broader application in scenarios requiring expeditious task execution.

The amount of assertions extracted through mining is influenced by the intricacy of the design. Given that the APB is a simple combinational design, the quantity of assertions mined for it by both methods is relatively small when compared to other, more complex benchmarks.

In Table. III the number of valid assertions is obtained by deducting the count of failed assertions from the overall total of mined assertions. This yields the quantity of assertions that have successfully passed the verification process.

To calculate the number of effective assertions we adopt a two-step pruning procedure. Initially, we retain only those valid assertions that successfully kill at least one mutant, thereby ensuring that each assertion contributes to identifying potential design faults. Subsequently, we eliminate redundant assertions—those whose detected mutants are entirely subsumed by the mutants detected by other assertions. This elimination step is crucial as it streamlines the verification suite by removing superfluous assertions without compromising the thoroughness of the test. By implementing this strategy, we aim to curate a suite of assertions that not only demonstrate high efficacy in mutant detection but also encompass the broadest spectrum of DUV behaviors.

In comparing the mutant detection results for three benchmarks between our method and GoldMine, distinct performance trends emerge. GoldMine outperforms our method in the LBDR benchmark, scoring 78.9% compared to our method's 62.1%. However, our method demonstrates superiority in Arbiter and APB benchmarks, achieving mutant detection of 75.5% and 75%, respectively, while GoldMine lags behind with scores of 59.3% and 12.5%.

In the LBDR benchmark, GoldMine attains a marginally higher mutant detection by employing static code analysis to refine the quality of the extracted assertion from the simulation trace. However, this enhancement is accompanied by a noticeable time overhead. In contrast, our approach emphasizes efficiency by exclusively depending on simulation traces, substantially reducing overall processing time.

GoldMine's limitations come to light in the APB benchmark as it grapples with creating random stimuli. This challenge becomes significantly pronounced when specific input values are critical in determining certain outputs. The random approach employed by GoldMine often falls short of generating the required stimuli promptly, leading to a lack of essential output traces in the simulation. In contrast, our method tackles this challenge by efficiently generating comprehensive simulation traces that encompass all functionalities expected from the DUV. Consequently, our approach consistently outperforms GoldMine, presenting a notable advantage in scenarios where precision, speed, and adaptability are paramount considerations.

To quantify the overall performance difference, we can calculate the average percentage improvement of our method over GoldMine, which is approximately 20.63%. This indicates that, on average, our method achieves a 20.63% higher mutant detection than GoldMine across the considered benchmarks.

In our research, we present a metric known as "Mutant Detection Effectiveness," detailed in the last row of Table. III. This metric is critical for evaluating the effectiveness of each assertions, as it measures the ability of each assertion to detect mutants, calculated by the ratio of detected mutants to the number of effective assertions.

Our method exhibits a pronounced advantage in "Mutant Detection Effectiveness" across all three benchmarks. We not only surpass GoldMine in every benchmark but also achieve an average effectiveness of approximately 9.06%, compared to GoldMine's 5.06%. This translates to our method being roughly 79% more effective than GoldMine, affirming the exceptional quality and comprehensive nature of our assertions in probing the DUV.

In contrast, GoldMine's approach leads to overly specific assertions. Each becomes a lengthy and complex formula aimed at capturing minute behaviors of the DUV. While trying to cover an extensive range of DUV behaviors, GoldMine generates a vast number of assertions. This results in a bloated verification process that demands more time. A case in point is the LBDR benchmark, where out of 1507 mined assertions, only 14 are effective, with the remainder being unnecessary. This inefficiency in GoldMine's approach highlights the precision and effectiveness of our methodology.

In summary, our experiments clearly show that our method surpasses GoldMine across the LBDR, Arbiter, and APB benchmarks, not just in terms of speed and scalability but also in effectiveness. Our approach boasts significantly faster execution times, emphasizing its efficiency and capability to handle extensive testing scenarios without compromising on thoroughness. While GoldMine manages to detect a marginally higher number of mutants in the LBDR benchmark through the application of static code analysis, our strategy focuses on optimizing efficiency by exclusively utilizing simulation traces, which notably reduces the overall processing time.

Our method's effectiveness is especially prominent in the APB benchmark, where it excels at overcoming the hurdles associated with the generation of random stimuli. This superior performance demonstrates our method's ability to adapt and accurately cover the required testing ground, making it an outstanding option for situations that demand precision, speed, and a high degree of adaptability.

VI. Conclusions and future works

This article introduces a new method for rapidly producing high-quality stimuli for dynamic assertion miners used in ABV. The proposed method enhances the ABV process by improving speed and accuracy compared to existing techniques. It also includes a mechanism for validating the mined assertions using these stimuli. The effectiveness of the proposed method was demonstrated by comparing its results to those of GoldMine. The experimental results reveal a remarkable reduction in assertion mining time by 5.76 times and an identification of 20.63% more faults in the design than GoldMine, while also yielding assertions that are 79% more effective. These findings indicate that our method is significantly faster and more efficient than GoldMine regarding assertion mining.

Acknowledgement

This work was supported in part by the Estonian Research Council grants PSG837.

REFERENCES

[1] Witharana, H., Lyu, Y., Charles, S. & Mishra, P. A survey on assertion-based hardware verification. *ACM Computing Surveys (CSUR)*. **54**, 1-33 (2022).

[2] Foster, H., Krolnik, A. & Lacey, D. Assertion-based design.(Springer Science & Business Media,2004).

[3] Ngo, X., Danger, J., Guilley, S., Najm, Z. & Emery, O. Hardware property checker for run-time hardware trojan detection. *2015 European Conference On Circuit Theory And Design (ECCTD)*. pp. 1-4 (2015).

[4] Danese, A., Filini, F., Ghasempouri, T. & Pravadelli, G. Automatic generation and qualification of assertions on control signals: A time window-based approach. *VLSI-SoC: Design For Reliability, Security, And Low Power: 23rd IFIP WG 10.5/IEEE International Conference On Very Large Scale Integration, VLSI-SoC 2015, Daejeon, Korea, October 5-7, 2015, Revised Selected Papers 23*. pp. 193-221 (2016)

[5] Danese, A., Ghasempouri, T. & Pravadelli, G. Automatic extraction of assertions from execution traces of behavioural models. *2015 Design, Automation & Test In Europe Conference & Exhibition (DATE)*. pp. 67-72 (2015)

[6] Ammons, G., Bodik, R. & Larus, J. Mining specifications. *ACM Sigplan Notices*. **37**, 4-16 (2002)

[7] Li, W., Forin, A. & Seshia, S. Scalable specification mining for verification and diagnosis. *Proceedings Of The 47th Design Automation Conference*. pp. 755-760 (2010)

[8] Ghasempouri, T., Malburg, J., Danese, A., Pravadelli, G., Fey, G. & Raik, J. Engineering of an effective automatic dynamic assertion mining platform. *2019 IFIP/IEEE 27th International Conference On Very Large Scale Integration (VLSI-SoC)*. pp. 111-116 (2019)

[9] Trippel, T., Shin, K., Chernyakhovsky, A., Kelly, G., Rizzo, D. & Hicks, M. Fuzzing hardware like software. *31st USENIX Security Symposium (USENIX Security 22)*. pp. 3237-3254 (2022)

[10] Chakraborty, S., Fremont, D., Meel, K., Seshia, S. & Vardi, M. On parallel scalable uniform SAT witness generation. *Tools And Algorithms For The Construction And Analysis Of Systems: 21st International Conference, TACAS 2015, Held As Part Of The European Joint Conferences On Theory And Practice Of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings 21*. pp. 304-319 (2015)

[11] Mehta, A. & Mehta, A. Constrained random verification (crv). *ASIC/SoC Functional Design Verification: A Comprehensive Guide To Technologies And Methodologies*. pp. 65-74 (2018)

[12] Cieplucha, M. Metric-driven verification methodology with regression management. *Journal Of Electronic Testing*. **35**, 101-110 (2019)

[13] Hertz, S., Sheridan, D. & Vasudevan, S. Mining hardware assertions with guidance from static analysis. *IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems*. **32**, 952-965 (2013)

[14] Malburg, J., Flenker, T. & Fey, G. Property mining using dynamic dependency graphs. *2017 22nd Asia And South Pacific Design Automation Conference (ASP-DAC)*. pp. 244-250 (2017)

[15] Iman, M., Raik, J., Jenihhin, M., Jervan, G. & Ghasempouri, T. An automated method for mining high-quality assertion sets. *Microprocessors And Microsystems*. **97** pp. 104773 (2023)

[16] Danese, A., Riva, N. & Pravadelli, G. A-team: Automatic template-based assertion miner. *Proceedings Of The 54th Annual Design Automation Conference 2017*. pp. 1-6 (2017)

[17] Germiniani, S. & Pravadelli, G. Harm: a hint-based assertion miner. *IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems*. **41**, 4277-4288 (2022)

[18] Bonato, M., Di Guglielmo, G., Fujita, M., Fummi, F. & Pravadelli, G. Dynamic property mining for embedded software. *Proceedings Of The Eighth IEEE/ACM/IFIP International Conference On Hardware/software Codesign And System Synthesis*. pp. 187-196 (2012)

[19] Boulé, M. & Zilic, Z. Assertions and the Verification Landscape. *Generating Hardware Assertion Checkers: For Hardware Verification, Emulation, Post-Fabrication Debugging And On-Line Monitoring*. pp. 13-35 (2008).

[20] Kaplan, H. Effective random seeding of random number generators. *Behavior Research Methods & Instrumentation*. **13** pp. 283-289 (1981)

[21] Spear, C. SystemVerilog for verification: a guide to learning the testbench language features. (Springer Science & Business Media,2008)

[22] Nelson, E. Improving constrained random testing by achieving simulation verification goals through objective functions, rewinding and dynamic seed manipulation. *Proceedings Of The Design And Verification Conference And Exhibition US (DVCon)*. (2017)

[23] Accellera,UVM 1.2 UserGuide, 2015.

[24] Committee, D. & Others IEEE Standard for SystemVerilog Unified Hardware Design, Specification, and Verification Language Standard IEEE 1800. *Http://www. Edastds. Org/sv/*. (2005)

[25] COCOTB's documentation., https://docs.cocotb.org/en/stable/, Last accessed on 2023-9-12

[26] Project Bonfire Network-on-Chip., https://github.com/Project-Bonfire/Bonfire_handshake, Last accessed on 2023-9-12

[27] APB to I2C controller, OpenCores.org.,https://opencores.org/projects/apbi2c., Last accessed on 2023-9-12

[28] Sun, X. & Gustafson, J. Toward a better parallel performance metric. *Parallel Computing*. **17**, 1093-1109 (1991)

Appendix 8

VIII

MRH. Iman, P. Chikul, G. Jervan, H. Bahsi, and T. Ghasempouri, "Anomalous File System Activity Detection Through Temporal Association Rule Mining," *In Proceedings of the 9th International Conference on Information Systems Security and Privacy (ICISSP)*, Lisbon, Portugal, 2023, pp. 733-740, DOI: https://doi.org/10.5220/0011805100003405.

Anomalous File System Activity Detection Through Temporal Association Rule Mining*

M. Reza H. Iman[1], Pavel Chikul[2], Gert Jervan[1], Hayretdin Bahsi[2] and Tara Ghasempouri[1]
[1]*Department of Computer Systems, Tallinn University of Technology, Tallinn, Estonia*
[2]*Centre for Digital Forensics and Cyber Security, Tallinn University of Technology, Tallinn, Estonia*

Keywords: NTFS, USN Journal, Forensics, Pattern Recognition, Association Rule Mining, Anomaly Detection.

Abstract: NTFS USN Journal tracks all the changes in the files, directories, and streams of a volume for various reasons including backup. Although this data source has been considered a significant artifact for digital forensic investigations, the utilization of this source for automatic malicious behavior detection is less explored. This paper applies temporal association rule mining to data obtained from the NTFS USN Journal for malicious behavior detection. The proposed method extracts association rules from two data sources, the first one with normal behavior and the second one with a malicious one. The obtained rules, which have embedded the sequence of information, are compared with respect to their support and confidence values to identify the ones indicating malicious behavior. The method is applied to a ransomware case to demonstrate its feasibility in finding relevant rules based on USN journal activities.

1 INTRODUCTION

The detection and exploration of malicious behavior are one of the mainstream research directions in the digital forensics domain. A huge number of data sources can be utilized in cyber incident investigations for identifying such behavior. The sources include but are not limited to network traffic captures, processes in memory, system call sequences, or Windows registry modifications. Microsoft NTFS Change Journal or USN Journal is another alternative that accumulates information regarding all of the operations performed on the file system.

NTFS forensics stands out as one of the cornerstones of conventional PC forensics due to the usage of file systems across all of the Microsoft Windows operating system lines. USN Journal is often used in system forensics to manually determine malicious or criminal actions (Cohen, 2020). It can shed some light on the executables launched in the system. File deletion traces of these files can still confidently be recovered from the journal. It enables tracking the file system operations related to file creation, renaming, deletion, or changing security attributes, thus, providing valuable information for malicious behavior once the benign usage is profiled (Corey, 2013; Russinovich, 2000). It is easy to access and extract this data when compared to, for instance, network traces or system calls, requiring additional tools and usually having limited historical coverage. Despite its huge potential, limited work has been done to date regarding the automated analysis of this important source of evidence.

In this paper, we examine the ways of forensic pattern recognition in the NTFS USN Journal using the Apriori algorithm (Han et al., 2012) and Temporal Association Rules (TAR) (Antunes and Oliveira, 2001; Bilqisth and Mustofa, 2020). Apriori is a fast algorithm that can provide accurate association rules (Han et al., 2012). Association rules demonstrate interesting relations among variables and data in a large dataset (Zaki, 2000).

To this end, association rule mining became a promising technique for extracting and exploring useful information from a system for engineers. It has shown its strength in many different domains, such as market analysis (Brin et al., 1997), accident and traffic analysis (Shahin et al., 2022), intrusion detection infrastructures (Treinen and Thurimella, 2006) and health informatics (Altaf et al., 2017), as well as its huge application in dependability and reliability of safety-critical applications (Danese et al., 2015; Hei-

*This work was supported in part by the European Union through European Social Fund in the frames of the "ICT programme" ("ITA-IoIT" topic) and by the Estonian Research Council grants PSG837.

H. Iman, M., Chikul, P., Jervan, G., Bahsi, H. and Ghasempouri, T.
Anomalous File System Activity Detection Through Temporal Association Rule Mining.
DOI: 10.5220/0011805100003405
In *Proceedings of the 9th International Conference on Information Systems Security and Privacy (ICISSP 2023)*, pages 733-740
ISBN: 978-989-758-624-8; ISSN: 2184-4356

dari Iman et al., 2021), etc.

In this research, temporal association rule mining identifies rules that are applicable to the USN Journal data for the detection of anomalies caused by malicious behavior. Mainly, the data mining approach extracted two sets of rules, one from a snapshot of a benign file system and another one from a target file system that is suspected to be infected or attacked. These two rule sets are compared, and the rules that detect the anomalies are determined. Security experts can use these rules for revealing and enumerating the files used or infected by the actions of the adversary. Thus, the proposed method does not only predict the existence of anomalies, but it also enables to discriminate the infected files from the benign ones to assist in the impact assessment of the incidents and planning the recovery actions during incident handling processes.

In summary, the contributions of this paper are as follows:

- An automatic malicious behavior detection method is proposed to analyze the NTFS USN Journal and extract a set of association rules for detecting the anomalies induced by malicious behavior.
- The method does not require file-level labeled data, instead, a normal file system, which is easy to obtain, and a target file system which is the main subject of the analysis are enough.
- An incident regarding the ransomware analysis is presented to demonstrate the applicability of the method.

The outline of this paper is as follows: Section 2 gives background information and reviews the related work. The preliminaries of the proposed method are presented in Section 3. The datasets and their generation are detailed in Section 4. Section 5 introduces the proposed methodology. The case study and the relevant results are presented and discussed in Section 6. Section 7 concludes the study.

2 BACKGROUND INFORMATION AND RELATED WORK

The USN Journal or Update Sequence Number Journal is an advanced feature of the Windows NT file system introduced with version 3.1 of the file system (Russinovich, 2000). It was designed to keep a record of all changes made to the volume. There are several use cases for the file system to maintain a full log of changes within itself. Backup applications may use the change journal information in order to identify files that were created or modified since the last backup without the need to recursively parse the directory tree which is time- and resource-consuming. Another useful application of the journal is real-time antivirus protection: the AV application can monitor the live USN journal to identify any incoming files and scan them at the same moment.

The journal is stored in a system-maintained metafile \$Extend\$UsnJrnl in an alternate data stream called \$J and is comprised of a number of records consisting of the following fields: a USN ID (a 64-bit unique identifier which is incremented with each new record been created but not guaranteed to be contiguous (Cooperstein and Richter, 1999)), a timestamp, filename, reference to the parent Master File Table (MFT) ID, the update reason, and some other attributes. The presence of parent MFT ID in some cases can lead to the real location of the file. However, if the MFT entry was already reused the reference becomes invalid. Update reason is a 64-bit integer that uses bit flags to describe what changed in the file or directory. According to Microsoft's documentation (Microsoft, 2022), there are 23 flags available, including creation, renaming, deletion, and security information change. Multiple flags can be set into a single update reason record. For example, two flags USN_REASON_FILE_CREATE (0x100) and USN_REASON_CLOSE (0x80000000) combined together will result in an integer record 0x80000100 or 2147483904 in decimal. One of the most important aspects of the journal is the fact that it stores information about operations on files that may be already deleted and their entries in the Master File Table reused. Thus, it is possible to prove some data's existence even if the data is a long time gone.

Different approaches for analysis of the USN journal in order to discover patterns are presented in several works. Lees et al. in (Lees, 2013) explore identifying a user using Private Browsing mode or utilizing anti-forensic software such as CCleaner. The proposed method allowed them to clearly identify traces and, most importantly, patterns for such activities within the change journal. Corey in their article "Re-introducing \$UsnJrnl" (Corey, 2013) discusses ways of using the change journal for determining malware activity from the USN journal including self-destruction, hiding in unusual locations, and tampering with the file system metadata. Cohen in their article (Cohen, 2020) demonstrates real-time monitoring and capture of the change journal with Velociraptor software in order to update the modified files hash database to trace the malicious activity.

Association rule mining has been applied to the detection of ransomware by using the data regarding dynamic link libraries called by the programs (Subedi

et al., 2018). Another study extracts association rules from the user login information for the purpose of user profiling (Abraham and de Vel, 2002). A solution based on a classifier composed of association rules is proposed for the problem of email authorship attribution (Schmid et al., 2015).

All the observed works that use the USN journal as the evidence source demonstrate semi-manual processes in pattern recognition mostly relying on the investigators' observations and prior knowledge of specific behavior. Obviously, these approaches need human expertise, they are costly and error-prone due to human beings in the loop. Thus, we see a clear indication of the need for an automated way for file system behavior patterns extraction. To the best of the authors' knowledge, this work is the first automatic malicious file system behavior detection that adopted data mining methods for this purpose.

3 PRELIMINARIES

Definition 1. ***Apriori*** *is a seminal data mining algorithm for mining frequent itemsets for Boolean association rules (Han et al., 2012). To mine association rules, Apriori employs an iterative approach called level-wise search, where k-itemsets are used to explore (k + 1)-itemsets (Han et al., 2012).*

Definition 2. *Let $I = \{i_1, i_2, ..., i_n\}$ be a set of items and $D = \{d_1, d_2, ..., d_m\}$ be a data set, i.e., a set of observations, called transactions, with respect the set of items I. Each element in D contains a subset of the items in I. An* ***association rule*** *is defined as an implication of form $X \rightarrow Y$ where X, $Y \subseteq I$ and $X \cap Y = \emptyset$. X and Y are called itemsets (Han et al., 2012).*

Definition 3. ***Temporal Association Rule (TAR)*** *is a kind of association rule that considers time in the data sets when the sequence of data changes during the time (Antunes and Oliveira, 2001; Bilqisth and Mustofa, 2020).*

Definition 4. *In TAR mining, there are different patterns including* ***Next****, and* ***Before*** *that consider different time series in a data set (Antunes and Oliveira, 2001; Bilqisth and Mustofa, 2020). As an example, $X \rightarrow Next(5min)Y$ means that when X occurs then after 5 minutes Y will be implied. Moreover, rule $X \rightarrow Before(5min)Y$ means that When X occurs, 5 minutes before it Y should have occurred.*

Definition 5. ***Support*** *is an indication of how frequently the itemset appears in the data set (Han et al., 2012). This value is between 0 and 1. For the rule $X \rightarrow Y$, the value of support is calculated with the following formula (Han et al., 2012):*

$$Supp(X \rightarrow Y) = P(X \cup Y) \quad (1)$$

In (1), $P(X \cup Y)$ is the probability where $X \cup Y$ indicates that a transaction contains both X and Y, that is, the union of itemsets X and Y.

Furthermore, in Apriori, min_supp value is the threshold and a minimum value that is chosen by the expert to decide whether an itemset is frequent (i.e., occurs frequently in the data set) or not. If the frequency of the itemset is more than this threshold, the itemset is considered a frequent itemset.

Definition 6. ***Confidence*** *is an indication of how often the rule has been found to be true. For the rule $X \rightarrow Y$, the value of confidence is calculated with the following formula (Han et al., 2012):*

$$Conf(X \rightarrow Y) = P(Y|X) \quad (2)$$

Confidence assesses the degree of certainty of the detected association rule. This is taken to be the conditional probability $P(Y|X)$, that is, the probability that a transaction containing X also contains Y. This value is between 0 and 1. The min_conf is the threshold and the minimum value that is chosen by the expert for confidence.

4 DATA SETS

As noted in (Cohen, 2020) and (Lees, 2013), different software utilizes different approaches in regard to file manipulations depending on their needs and implementation specifics that usually result in several change records being created. For example, unpacking a file from an archive will in most cases result in three USN records being generated:

- 256 (FILE_CREATE)
- 258 (DATA_EXTEND FILE_CREATE)
- 2147483906 (DATA_EXTEND FILE_CREATE CLOSE)

Various software actions (both operating system and user applications) performing file operations result in a continuous flow of USN records created in the journal. Thus, our assumption is that it is possible to fingerprint specific software behavioral patterns and classify such actions (both legitimate and malicious).

To test our assumption with different behavioral patterns we created two datasets: the first one with legitimate behavior only and the second one introducing some malicious activity inside the normal operating system lifecycle. A fully patched Windows 7 virtual machine was set up and an origin snapshot was created (snapshot 1). For the "legitimate" dataset creation, some user activities were simulated.

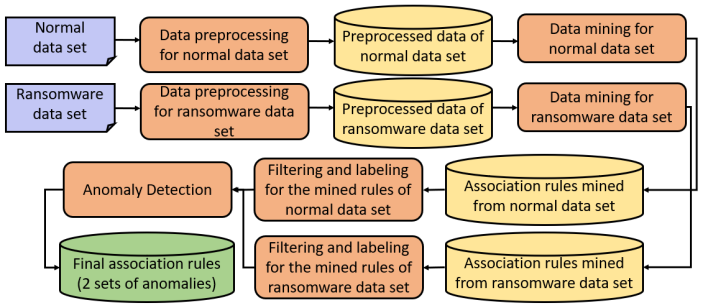


Figure 1: General flow of the proposed method.

These activities included web browsing, user document editing, and, most importantly, software installation. On a high level, software installation involves various file operations that might be common to malicious software actions as well: files unpacked, temporary files created in different locations to be later deleted, etc. which could make the analysis harder. After some time a snapshot was created representing the first "normal" dataset (snapshot 2). To introduce malicious activity the system was reverted to the origin snapshot and infected with the WannaCry malware as a typical ransomware representative. WannaCry is a ransomware crypto-worm that when triggered on a target machine iterates over user files encrypting them and by the end of the encryption phase displays a notification demanding ransom in order to decrypt the files. In a worldwide attack in 2017 WannaCry infected more than 200.000 machines in more than 150 countries dealing billions of dollars in damage (Trautman and Ormerod, 2018). When the system went into the ransom-demanding state another snapshot was created representing the second "infected" dataset (snapshot 3).

The file operation sequences that represent a single action (such as the un-archiving of a file mentioned above) tend to be atomic meaning that the records representing an action will stay close to each other in the journal. However, due to the parallel writing in the journal, the patterns relevant to several files may be mixed with each other. Thus, to overcome this behavior we do the initial preparation of the datasets so that the records related to a single file are batched together in a one-second timeframe. To demonstrate such preparation refer to Table 1.

We extracted USN journals from snapshots 2 and 3 and after running the preparation procedure on them as discussed above we then converted them into arrays of USN update reasons, *i.e.*, lists of 64-bit integers. Thus we resulted in two datasets representing file system activity under different circumstances: legitimate actions (∼19.000 records) and legitimate actions with some malicious actions mixed in (∼14.000 records).

Table 1: Raw journal data preprocessing.

	Original	Preprocessed
1 second	File-1 record 1	File-1 record 1
	File-2 record 1	File-1 record 2
	File-1 record 2	File-1 record 3
	File-2 record 2	File-2 record 1
	File-1 record 3	File-2 record 2

5 PROPOSED METHODOLOGY

The general flow of the proposed method has been illustrated in Fig. 1. As mentioned in Section 4, in our case study, one data set is related to the normal behavior of a user while he/she was using the system. The other data set is related to the behavior of the system when ransomware inflicted damage on it. As can be seen in Fig. 1, association rule mining is applied separately on both sets of data. Therefore, at first, a data preprocessing phase is performed on data sets to prepare suitable data for association rule mining. Afterward, in the data mining phase, the Apriori algorithm is applied to the prepared data sets separately. The outcome is two sets of association rules which have been mined from any of the normal and infected data sets.

As illustrated in Fig. 1, by reaching the association rules, a comparison is done between the two data sets based on the mined association rules. This comparison is performed with the aid of the values of Support (Definition 5) and Confidence (Definition 6) metrics. Based on our assumption, if both data sets are similar, the mined rules from each of them should be similar. This similarity means that in addition to the mined rules, the values of the Support and Confidence for these rules should be the same. Therefore, any difference in these values can show an anomaly. The result of the comparison will be two new sets of

association rules (shown in the green box in Fig 1), which both indicate the anomalies in the system. One of these two sets of anomalies contains the rules that have occurred only in the infected data set. The other one is the set of rules that have occurred in both data sets, however, the values of their *support* and *confidence* are different.

More details about each phase and how the mined rules will be compared are discussed in the following subsections.

5.1 Data Preprocessing

In this phase, data preprocessing is performed to provide suitable data for Apriori to extract TARs (Definition 3) from the datasets. The Apriori algorithm extracts frequent itemsets in the form of association rules without considering the sequence of events during the time. However, we are interested in rules which illustrate the sequence of events through time.

In other words, in Apriori, it does not matter whether an itemset is followed by another or preceded by another. It only finds those itemsets that have occurred together (without considering their sequences and orders). However, in NTFS USN Journal the concept of time or more specifically the sequence of operations that occur in the system matters. More precisely, if event X happens at second 1 and event Y happens at second 6, then the association rule regarding this sequence of events would be $X \rightarrow 5seconds\ Y$, which means Y happens 5 seconds after the happening of X. The mentioned association rule is what we are interested in extracting for this work. Mining these kinds of rules will be helpful for security experts to more accurately find the files or directories related to the malicious behavior in the system.

In this regard, in preprocessing step first, the user identifies the length of time for the rules. For instance, if a rule such as $X \rightarrow 5seconds\ Y$, is in the interest of the users, therefore number 5 should be identified. Second, all the events in the dataset with the identified length (in this case 5), are clustered in the same sub-dataset. Finally, the concept of time for each event in the sub-data set is removed and saved for future reference (authors do not describe technical details to make it easier to read). Finally, this sub-dataset is fed to the next step for mining the association rules.

5.2 Data Mining

In this phase, the Apriori algorithm (Definition 1) (Han et al., 2012) is applied to the Preprocessed data sets to generate association rules. According to Fig. 1, this phase takes two sets of data as the inputs, one for the normal set of data, and the other one for the infected set. The outputs of this phase are association rules related to both sets. Due to the space limit, we refer interested readers about the Apriori to the literature (Han et al., 2012).

5.2.1 Applying Temporal Filters and Labels

This phase aims to restore the time instance of events that were removed in the Preprocessing phase, Section 5.1. In accordance with our previous statement, the extracted association rules are generated in two formats, namely *next* and *before* (Definition 4). Detailed instructions on how time instances are set back to the rules are provided below:

After mining association rules in the previous phase (section 5.2), the method provides us a set of rules in the form of P $\rightarrow$ Q. By considering P $\rightarrow$ Q, we will have two different conditions as follows:

- *next:* If the value of P is equal to some events in the data set, and the value of Q is equal to the events that in the data set have appeared after the events of P, this means that the extracted association rule is *next*. Therefore, the mined rule is labeled as a *next* TAR.
- *before:* If the value of P is equal to the events that have appeared in the data set before the events of Q, this means that the extracted association rule is *before*. Therefore, the mined rule is labeled as a *before* TAR.

5.3 Anomaly Detection

This phase is in charge of automatically detecting malicious behavior in the NTFS USN Journal which is typically performed by ransomware. The assumption in the proposed method is that in the 'normal' scenario that there is no malicious behavior in the infected dataset, two data sets should be similar (normal and infected data sets). This means that if the Apriori algorithm is applied to both data sets, the mined rules, as well as the values of their *supports* (Definition 5) and *confidences* (Definition 6) should be similar.

In order to find the anomalies, the method compares the two sets of mined rules. In this comparison, two different conditions and two different sets of anomalies would occur. In fact, in this comparison, we are looking for the conditions that neglect our assumption (*i.e.*, similar behavior and similar mined rules for both data sets in a normal scenario)

The first set of rules is the one that has not occurred in the normal data set and occurs only in the infected data set. Based on our assumptions, these

rules show malicious behavior. The other set of mined rules is one that is the same in both data sets. For these rules, the *support* and *confidence* values of each rule are calculated. Next, according to the following formulas, we calculate the difference between their *supports* and *confidences*:

$$\mathcal{DS} = (Support1 - Support2) \times 100 \quad (3)$$

It should be noted that each rule that has been mined from the normal data set or infected data set has a support value. With the aid of the formula (3), we calculate the support difference for each pair of rules that has been mined from each data set and are exactly the same (*i.e.*, similar rules that have been mined from both data sets, but with different support values). In the above formula, *Support*1 is the calculated support for a specific rule that has been mined from the normal data set, and *Support*2 is the calculated support of that specific rule that has been mined from the infected data set.

If $\mathcal{DS} > 0$, it means that a malicious behavior has occurred, and it shows that in comparison with the normal data set, some parts of data have been removed from the infected data set. On the other hand, if $\mathcal{DS} < 0$, it means that there is malicious behavior again, however, in comparison with the normal data set, additional records of data have been added to the infected results file. Furthermore, the formula (4) and according to Definition 6 shows the probability of malicious behavior in a specific rule.

$$\mathcal{DC} = (Confidence1 - Confidence2) \times 100 \quad (4)$$

In the above formula, *Confidence*1 is the calculated confidence for a specific rule that has been mined from the normal data set, and *Confidence*2 is related to the calculated confidence for that specific rule that has been mined from the infected data set. For instance, if for a mined rule like $P \rightarrow Q$, *Confidence*1 – *Confidence*2 is equal to 95, it means that in 95% of the operations that this rule shows in data sets, we have a malicious behavior.

Table 2: Number of Mined Association Rules.

Rules	Unequal Support	Infected Only
#Association Rules	1	14
#Before Rules	0	1
#Next Rules	1	13

6 EXPERIMENTAL RESULTS

The experimental results of the proposed method have been elaborated in this section. The normal data set that we have used in this paper has 19055 records and the infected data set has 13721 records.

In Table 2, the number of all mined rules ('#Association Rules'), as well as the number of Before ('#Before Rules') and Next ('#Next Rules') rules have been presented. It should be noted that the length of the sequence of operations in the data set has been set to 9 based on the expert's decision. Thereby, for the preprocessing phase, the number of shifts is equal to 9. Since the detection of the attacks is significantly important, the minimum confidence value has been assigned to 0.80 out of 1. Note that this number can easily be changed by the expert as the proposed method is fully automated. For highly critical cases this value can be set to higher; otherwise, a lower degree can be set by the user to introduce more sensitivity in rule mining.

As mentioned in section 5.3 (Anomaly detection), the method provides two sets of rules (anomalies). One set is related to the rules that have unequal support values and their difference is calculated according to formula 3 (DS). However, the second rule set is the one that has occurred only in the infected data set. In Table 2, the number of mined rules have been demonstrated for both of these two sets, *i.e.*, 'Unequal Support' and 'Infected Only' columns. There is only one rule in both data sets with unequal support values mined with the Next pattern. The figure for the rules that have not occurred in the normal data set is 14 with 1 rule mined with the Before pattern and the rest with the Next pattern. Regarding the execution time, the method is able to mine both categories of rules in less than a second.

6.1 Digital Forensics Interpretation of Rules

All 14 rules in the unified format are presented in the table 3. Basically, the unified format represents all of the reasons records that are put in consecutive order the way they are supposed to be found in the dataset. If we take the first rule as an example, the original mined rule's consequent was 2147483652 and the list of antecedents was as follows: [6_before_4, 4_before_2147484160, 1_before_4, 8_before_256, 7_before_2147483904, 3_before_256, 5_before_2147483652, 2_before_2147483904]. It practically means that we will be looking for a record 256, followed by a record 2147483904, followed by 4, and so on until we find the exact match of the whole sequence ending with a 2147483652. It should be noted that all parts of the antecedent of this rule should occur together in the data set to finally imply the consequent.

As the first part of our validation, we ran all our mined rules against the infected dataset and extracted

Table 3: Association rules in unified format.

#	Rule	Confidence
1	256, 2147483904, 4, 2147483652, 2147484160, 256, 2147483904, 4, 2147483652	1
2	256, 2147483904, 4, 2147483652, 2147484160, 256, 2147483904, 4, 2147483652	0.831050228
3	2147483652, 2147484160, 256, 2147483904, 4, 2147483652, 2147484160, 256, 2147483904	0.965714286
4	2147483904, 4, 2147483652, 2147484160, 256, 2147483904, 4, 2147483652, 2147484160	1
5	33026, 2147516674, 4096, 256, 258, 32768, 2147516416, 8192, 2147491840	1
6	4096, 8192, 2147491840, 4096, 8192, 2147491840, 4096, 8192, 2147491840	0.886956522
7	4, 2147483652, 2147484160, 256, 2147483904, 4, 2147483652, 2147484160, 256	0.945945946
8	258, 32768, 2147516416, 8192, 2147491840, 33026, 2147516674, 4096, 256	0.843137255
9	32768, 2147516416, 8192, 2147491840, 33026, 2147516674, 4096, 256, 258	0.931818182
10	2147484160, 256, 2147483904, 4, 2147483652, 2147484160, 256, 2147483904, 4	0.982248521
11	8192, 2147491840, 4096, 8192, 2147491840, 4096, 8192, 2147491840, 4096	1
12	256, 258, 32768, 2147516416, 8192, 2147491840, 33026, 2147516674, 4096	1
13	49152, 2147532800, 8192, 2147491840, 256, 258, 33026, 2147516674, 4096	1
14	2147491840, 4096, 8192, 2147491840, 4096, 8192, 2147491840, 4096, 8192	0.982142857

a histogram of the affected file types (Table 4). The second and third most frequent file types are WNCRYT and WNCRY. These file types represent the temporary storage and the final encrypted container generated by the WannaCry ransomware accordingly (Team, 2017). As for the TMP files, we suppose that those are also temporary files generated by the malware since they were created in the infected directories (as indicated by the Parent File Reference entry in the record) and the timestamps match the timeframe of the attack. The rest of the files comprise less than 9% of the total detected records that were false-positively identified. Having this information we may conclude that the rules correctly detect the anomalies caused in the file system by malicious activity. To get the accuracy of the identification, we took all of the unique file entries that were affected by the attack and compared them with the ones detected by the rules: out of 235 affected files we detected 206 which makes an 87.7% accuracy.

Table 4: Detected file types histogram.

File Type	Number of Hits
tmp	1020
wncryt	710
wncry	411
png	101
txt	31
db	24
docx	18
zip	12
js	6
vbs	5
gif	3
lnk	1

If we look closer at the 14 mined rules we can identify that some of them are just shifted versions of others. For example, rules 1, 2, 3, 4, 7, and 10. This behavior was expected since the contiguous repetitive patterns in the USN Journal can be grabbed by the algorithm from different starting points. This leaves us with 4 groups representing the unique rules: (1, 2, 3, 4, 7, 10), (6, 11, 14), (5, 8, 9, 12), and (13). Only one rule number 13 does not have a shifted version of itself. We extracted individual outputs of single rules from the identified groups. A comparison of the outputs showed little to no difference in the identified records. Thus we end up with only 4 distinct rules for malicious behavior detection. Another aspect noted is the repetitiveness of the pattern in the mined rules. For example, rule number 6 [4096, 8192, 2147491840, 4096, 8192, 2147491840, 4096, 8192, 2147491840] is a repetition of the same 3-value pattern [4096, 8192, 2147491840] three times. It is a part of future work to address both the elimination of shifted rule versions and the shortening of repeated patterns.

Machine learning methods can be considered a significant alternative to the proposed method. However, there are some obstacles to applying them in this context. It is easy for a forensic expert to create a snapshot with a benign file system. The target snapshot which constitutes the subject of investigation usually contains benign and malicious files which are blended into one file system. Supervised learning models require file-level labels to provide scrutiny about each file, which is very hard to achieve in digital forensics tasks due to the high cost of labeling. One-class learning models, which may just learn from the files in the benign snapshot, cannot use the target snapshot while inducing the models, limiting the knowledge that can be obtained from both snapshots. Unsupervised methods (*e.g.*, clustering) that do not use any labeled data may give some intuition to the expert but they do not provide explicit rules. More importantly, machine learning models do not provide human-readable rules, which limits their applicability in this context enormously. Even the explainable methods such as decision trees may require additional steps to generate rules and strict pruning strategies should be applied to achieve comprehendible rule sets at expense of detection loss.

7 CONCLUSION AND FUTURE WORK

In this work, we proposed an automated way of discovering patterns in the NTFS USN change journal by utilizing Temporal association rule mining. A data preprocessing method is introduced which can customize the Apriori algorithm for mining Temporal association rules instead of mining traditional rules where time has no meaning. The method can be applied for both real-time and post-mortem pattern recognition. We assume that normal and malicious software leave distinct fingerprints in the file system that are recorded by the change journal. To test this theory we validate the method by trying to detect the patterns of ransomware presence in the system. This is achieved by practically infecting an operating system with malware and then running the proposed system against the extracted USN journal. As a result of such validation, we identified patterns specific to malware activity. More specifically, the files which are infected or generated by malicious activity are found by the association rules mined from normal and infected data sets.

As part of future work, we envision a system that will utilize the proposed method in real-time to monitor the activity of a live system in order to detect patterns at the moment close to emerging. Another prominent application would be the automatic generation of a forensic timeline that shows the system behavior and possible attack timeframes and volume. From the perspective of technical improvement, we are planning to address the shifted rules handling and merging in order to reduce the number of redundant patterns. The same applies to the repetitive patterns inside the rules: we need to shorten the identified pattern if it is just a repeated sub-pattern present in it.

REFERENCES

Abraham, T. and de Vel, O. (2002). Investigative profiling with computer forensic log data and association rules. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 11–18. IEEE.

Altaf, W., Shahbaz, M., and Guergachi, A. (2017). Applications of association rule mining in health informatics: A survey. *Artif. Intell. Rev.*, 47(3):313–340.

Antunes, C. and Oliveira, A. L. (2001). Temporal data mining: an overview.

Bilqisth, S. and Mustofa, K. (2020). Determination of temporal association rules pattern using apriori algorithm. *IJCCS (Indonesian Journal of Computing and Cybernetics Systems)*, 14:159.

Brin, S., Motwani, R., Ullman, J. D., and Tsur, S. (1997). Dynamic itemset counting and implication rules for market basket data. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 255–264.

Cohen, M. (2020). The windows usn journal.

Cooperstein, J. and Richter, J. (1999). Keeping an eye on your ntfs drives: the windows 2000 change journal explained. *MICROSOFT SYSTEMS JOURNAL-US EDITION-*, 14:17–30.

Corey, H. (2013). Re-introducing $usnjrnl.

Danese, A., Filini, F., Ghasempouri, T., and Pravadelli, G. (2015). Automatic generation and qualification of assertions on control signals: A time window-based approach. In *IFIP/IEEE International Conference on Very Large Scale Integration-System on a Chip*, pages 193–221. Springer.

Han, J., Kamber, M., and Pei, J. (2012). 6 - mining frequent patterns, associations, and correlations: Basic concepts and methods. In Han, J., Kamber, M., and Pei, J., editors, *Data Mining (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 243–278. Morgan Kaufmann, Boston, third edition edition.

Heidari Iman, M. R., Raik, J., Jenihhin, M., Jervan, G., and Ghasempouri, T. (2021). A methodology for automated mining of compact and accurate assertion sets. In *2021 IEEE Nordic Circuits and Systems Conference (NorCAS)*, pages 1–7.

Lees, C. (2013). Determining removal of forensic artefacts using the usn change journal. *Digital Investigation*, 10(4):300–310.

Microsoft (2022). Usn_record_v2 - win32 apps.

Russinovich, M. (2000). Inside win2k ntfs, part 1.

Schmid, M. R., Iqbal, F., and Fung, B. C. (2015). E-mail authorship attribution using customized associative classification. *Digital Investigation*, 14:S116–S126.

Shahin, M., Heidari Iman, M. R., Kaushik, M., Sharma, R., Ghasempouri, T., and Draheim, D. (2022). Exploring factors in a crossroad dataset using cluster-based association rule mining. *Procedia Computer Science*, 201:231–238. The 13th International Conference on Ambient Systems, Networks and Technologies (ANT).

Subedi, K. P., Budhathoki, D. R., and Dasgupta, D. (2018). Forensic analysis of ransomware families using static and dynamic analysis. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 180–185. IEEE.

Team, C. T. U. R. (2017). Wcry (wannacry) ransomware analysis.

Trautman, L. J. and Ormerod, P. C. (2018). Wannacry, ransomware, and the emerging threat to corporations. *Tenn. L. Rev.*, 86:503.

Treinen, J. J. and Thurimella, R. (2006). A framework for the application of association rule mining in large intrusion detection infrastructures. In Zamboni, D. and Kruegel, C., editors, *Recent Advances in Intrusion Detection*, pages 1–18, Berlin, Heidelberg. Springer Berlin Heidelberg.

Zaki, M. (2000). Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390.

Appendix 9

IX

M. Shahin, M. Burtl, MRH. Iman, T. Ghasempouri, R. Sharma, SA. Shah, and D. Draheim, "Significant Factors Extraction: A Combined Logistic Regression and Apriori Association Rule Mining Approach," *13th Computer Science On-line Conference 2024 (CSOC'24)*, 2024.

Significant Factors Extraction: A Combined Logistic Regression and Apriori Association Rule Mining Approach

Mahtab Shahin[1], Markus Burtl[2], M.Reza H.Iman[3], Tara Ghasempouri[3], Rahul Sharma[1], Syed Attique Shah[4], and Dirk Draheim[1]

[1] Information Systems Group, Tallinn University of Technology, 12616 Tallinn, Estonia
[2] Department of Health Technologies, Tallinn University of Technology, Akadeemia Tee 15A, Tallinn, 12618, Estonia
[3] Department of Computer Systems, Tallinn University of Technology, 12616 Tallinn, Estonia
[4] School of Computing and Digital Technology, Birmingham City University, Birmingham B4 7XG, U.K

Abstract. The global COVID-19 pandemic has become a phenomenon that has severely disrupted human life. It is widely recognized that taking faster, evidence-based measurements based on disease parameters is crucial for monitoring and preventing the further spread of COVID-19. One of the essential tasks in data mining is mining rules because rules provide concise statements of potentially important information that end users can easily understand. Therefore, attaining significant information in rules is the key to containing COVID-19 outbreaks. Our objective is to discover hidden but critical knowledge in the form of rules based on the risk factor dataset of COVID-19 patients. In this paper, we use association rule mining to extract information from rules in COVID-19 patients' risk factor data that could be used to initiate prevention strategies. We discovered the rules of dead and recovered or hospitalized patients to understand and compare their characteristics. This approach can assist clinicians in effectively managing and treating diseases by providing valuable insight.

Keywords: Knowledge discovery, data mining, association rule mining, rule generation, rule discovery, logistic regression

1 Introduction

Modern society has become increasingly reliant on data mining, a method consisting of various methodologies such as classification, grouping, regression, and correlation [1]. Data mining exposes previously unknown independent item sets and their intricate relationships within large databases through systematic processes. Among the myriad applications of data mining, association rules play a

pivotal role in real-world scenarios spanning web data analysis, consumer behavior researchjhb, cross-marketing, catalog design, and medical record analysis [2]. Moreover, in fields like biological sciences, Association Rule Mining (ARM) contributes significantly to accurate classification prediction and illness detection.

ARM, a subset of data mining, involves the exploration of patterns and correlations within datasets using various algorithms such as Apriori, FP-growth, and Eclat [3,4,5]. Through the application of support and confidence parameters, ARM uncovers associations between seemingly unrelated datasets, facilitating meaningful insights. The support parameter denotes the frequency of relationships within the database, while the confidence parameter indicates the accuracy of those relationships [6,7]. The primary objective of association rule mining is the identification of distinctive frequent itemsets, achieved through sequential steps including frequent itemset mining and rule generation. Rules failing to meet predefined confidence thresholds are pruned, refining the extracted associations.

This paper contributes significantly to the field by employing association rule mining algorithms to discern frequent risk factors among Covid-19 patients. This includes those who have died, been hospitalized, or recovered. By analyzing various factors such as travel history, symptoms, race differences, chronic diseases, and age groups, the study aims to elucidate the statistical significance of these variables in the context of Covid-19 outcomes. This endeavor represents a novel application of ARM techniques in public health, offering valuable insights into pandemic mitigation strategies and healthcare management.

A support and confidence parameter is used to discover links between unrelated datasets, while an ARM is created by looking for recurring patterns in the data. A support value reflects the frequency of relationships occurring in a database, whereas a confidence value reflects the likelihood that these relationships are accurate [6,7]. A dataset is generated with all itemsets that meet the minimum support requirements. In the second step, all frequent itemsets are used to develop all potential rules from the dataset. After that, rules that do not meet specified minimum confidence levels are removed. Identifying distinctive frequent itemsets is the main component of association rule mining. At present, numerous ARM algorithms are in use, including Apriori [3], FP-growth [4], and Eclat [5].

Contribution The research addresses numerous contributions, as summarized below:

- The statistical significance of travel history, symptoms, race differences, chronic diseases, and age group were determined in Covid-19 patients.
- To the best of our knowledge, this is the first study to use association rule mining algorithms to identify the frequent risk factors for Covid-19 patients, including those who have died, been hospitalized, or who have recovered.

Healthcare providers can obtain useful information by identifying the practical factors that influence patients whose Covid-19 tests are positive. This will

enable them to identify and treat patients with a greater risk of Covid-19 when an outbreak of infectious diseases or other mutation types of Covid-19 occurs. It is possible to detect patterns in a dataset using simple association rules, which is useful for analyzing clinical data. Furthermore, it allows professionals to make well-informed diagnoses, gather significant data, and construct critical knowledge bases within a short time. This study aimed to examine symptom patterns in Covid-19 patients and break them down based on age, race, chronic illness, symptoms, and travel history.

Following is an outline of the remainder of the paper. In Sect. 2, we review various related works in the field. In Sect. 3, we describe the details of the methodology, dataset, and pre-processing. In Sect.4, we demonstrate the experimental results. In Sect. 5, we discuss the study findings. Finally, in Sect. 6, we conclude the paper and present possible directions for future works.

2 Related Work

Biomedical research increasingly relies on machine learning approaches for prediction and knowledge discovery. Medical applications of machine learning include genomic analysis, disease-gene analysis, mortality prediction, personalized medicine, drug detection, adverse drug event prediction, patient similarity, and explainable approaches to artificial intelligence.

Agrawal et al. first proposed ARM [8]. Accordingly, this technique was initially developed to analyze market basket data to identify all the rules for predicting occurrences of specific products based on the occurrence of other products within the same "set of transactions." The ARM algorithm utilizes brute force as its basic concept. The method involves listing all feasible rules and then pruning those that do not satisfy the condition. The large number of possible combinations of this strategy makes it computationally prohibitive. R. Agrawal [8] devised the Apriori approach to decrease the number of candidates. The Apriori approach has two significant flaws. Initially, it generates many candidate itemsets from an extensive data set while also creating frequent itemsets. Additionally, several database scans are required, increasing computing costs. To overcome these limitations, Han et al. [9] proposed Frequent Pattern Growth (FP-growth). With the FP-growth method, a tree representation of the dataset is created, and the itemets of the dataset are associated with each other. There are several disadvantages associated with the FP-growth method. The process of constructing an FP tree is more complex than that of constructing an Apriori tree. If the database is too large, the algorithm may not be able to fit into shared memory. In both Apriori and FP-growth, horizontal data formats are used. In [10], Zaki et al. presented the equivalence class clustering and bottom-up lattice transversal technique for ARM, in which horizontal data could be converted into vertical data using Eclat. The advantage of Eclat over Apriori is that it requires less database scanning. Based on a cross-country Covid-19 dataset, Shahin et al. [11] assessed the performance of Apriori and FP-growth through different Spark components and seeks to understand how they differ. This strategy has the sig-

nificant disadvantage of consuming a large amount of memory when there are many transactions in the dataset. [12] describes an example of knowledge mining using association rules to identify indicator diseases associated with psychiatric disorders. ARM reliability can be confirmed by the fact that the association rules found in the study are consistent with clinical guidelines in psychiatry. This study has demonstrated that association rule mining can be used to extract comorbidities and identify indicator diseases from health insurance billing data.

ARM is becoming increasingly recognized as an active research area among data mining researchers [13,14,15,16,17,18]. Recently, different incremental methods have been presented for mining association rules to extract identified patterns [15,19]. The use of ARM in healthcare has been widespread for many years. Zhou et al. [20] systematically analyzed coupled hospital infection (HI) risks using a multimethod fusion model combining association rule mining and complex networks. The Apriori algorithm generates association rules based on coupled relations between risk factors. The risk factors associated with HI are constructed using existing rules.

Many hidden correlations exist between qualities (symptoms) and diseases. We can better understand the disease and its biomarkers by discovering these connections. Certain risk factors for heart disease have been identified in particular research [21]. The prevalence of early childhood caries was determined using the ARM method by Vladimir et al. [22]. To identify distinct risk factors for cardiovascular disease, hepatitis, and breast cancer, Borah and Nath [23] proposed a dynamic rare association rule mining approach. According to [24], ARM could help curb the obesity epidemic primarily caused by a lack of physical activity. To discover adverse reactions induced by drug-drug interactions, Cai et al. [25] employed ARM. Nirmala and Ramasamy [26] utilized ARM with a keyword-based clustering approach to predict disease. Kamalesh et al. used ARM to predict diabetes mellitus risk [27]. Pokharel et al. [27] employed sequential pattern mining with a gap limitation to uncover patient commonalities, including death prediction and sepsis identification. The study by Nahar et al. [28] identified factors contributing to heart disease for male and female cohorts in symptom mining utilizing ARM. Borah et al. [23] used ARM to find symptoms and risk variables for three diseases (cardiovascular disease, hepatitis, and breast cancer). Lau et al. [29] developed constraint-based ARM across subgroups to aid doctors in finding valuable patterns in dyspepsia patients.

This paper examines significant rules for Covid-19 patients using the Covid-19 patients' database [30]. When physicians educate patients about risk factors for Covid-19, rules can assist them in making informed decisions.

3 Methodology

3.1 Description of the WHO Covid Dataset

After extracting anonymized Covid-19 patient data from the WHO (World Health Organization) Covid-19 database from December 2019 to January 2020 [30], we

Table 1. Distribution according to travel history.

Travel History	Count
Yes	1,483,209
No	397,845

exported and cleaned the data with the data management software platform R, version 3.4. More information about the data for this study is available on github[5]. The study's primary purpose was symptom mining; therefore, we created a dataset for patients with symptom information and excluded all missing values. As there are relationships between the attributes within the dataset, we extracted only 5 of the 31 attributes or columns for our analysis. An illustration of the selected attributes can be found in table 6. The distribution of all features is shown in table 1 to table 5. Figure 2 presented the data extraction process. Among 1,881,054 patient records, 101,800 died, while 1,779,254 were recovered or hospitalized. Bar plots of the age group, race difference, symptoms, and chronic disease are shown in figure 1.

It is worth mentioning that to simplify the analysis, the authors classified the patients' ages into five main age groups. These groups are summarized in Table 5. Furthermore, WHO[1] has classified symptoms into three main groups: most common, less common, and serious. A fever, cough, tiredness, and loss of taste or smell are some of the most common symptoms. Less common symptoms include a sore throat, a headache, aches and pains, diarrhea, a rash on the skin, discoloration of fingers or toes, redness or irritation of the eyes, and finally, the most serious symptoms include difficulty breathing or shortness of breath, loss of speech or mobility, confusion, or chest pain. The authors followed the WHO symptom classification in this study as well.

Table 2. Distribution of symptoms.

Symptoms	Count
Most common symptoms	898,754
Less common symptoms	419,076
Serious symptoms	563,224

3.2 Conversion of the Data into a Transactions Database

The dataset has been converted into transactions for association and class rule mining. For instance, for a feature such as chronic diseases, there were a total of six values, namely cancer, diabetes, hypertension, stroke, heart disease, and pulmonary conditions; for that, six columns have been created accordingly with

[5] `https://github.com/beoutbreakprepared/nCoV2019`

[1] `https://www.who.int/health-topics/coronavirus#tab=tab_3`

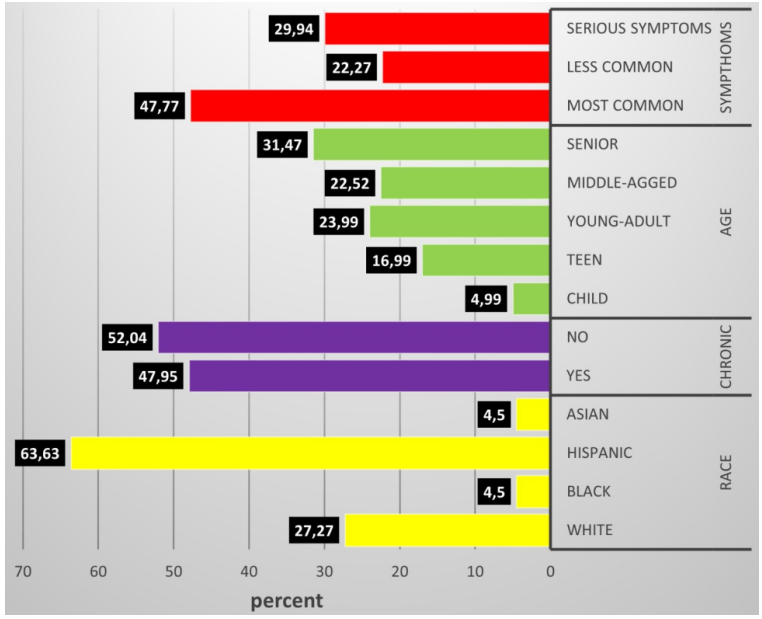


Fig. 1. Relative frequency of symptom, age, chronic disease and race in COVID-19 patients

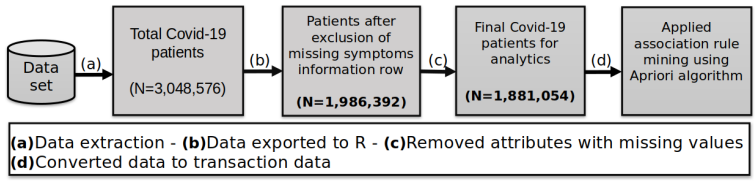


Fig. 2. Data extraction and management process

Table 3. Distribution of race differences.

Race Difference	Count
White	282,158
Black	47,013
Hispanic	658,368
Asian	47,035

Table 4. Distribution according to chronic disease.

Chronic Disease	Count
Yes	901,973
No	979,081

the values yes or no. For example, if an individual suffers from heart disease, then Yes or 1 would be in the corresponding column; if not, the value would be No or 0. In this way, a total of 46 columns have been created. So, in total, there were 46 items or columns.

3.3 Data Analysis Approach

As a first stage, we used the logit model on the Covid-19 dataset to identify relevant factors that may affect the likelihood of Covid-19 disease. After that, we applied association rule mining based on these factors to find significant rules for died and recovered or hospitalized patients.

Logit Model In the current study, the dependent attribute of recovered or hospitalized patients' condition (No or 0) or died (Yes or 1) is dichotomous and thus represented as a binary variable. The binary logit model is extensively used in clinical investigations where the response variable is binary [31]. The model takes the natural logarithm of the likelihood ratio meaning the dependent variable becomes 1 (breast cancer) or 0 (no breast cancer). Let p_1 and p_0 represent the probabilities of the response to variable categories recovered or hospitalized patients and dead patients, respectively. The binary logit model is given as:

$$Y = log\left(\frac{P_0}{P_1}\right) = \alpha + \beta_i X_i \tag{1}$$

Table 5. Distribution of age groups.

Age group	Count
4-12(Child)	94,052
13-19(Teen)	319,779
20-34(Young adult)	451,452
35-64(Middle-aged)	423,642
65+(Senior)	592,129

Table 6. Selected attributes.

Attribute	Description
Age group	Age group of the reported case.
Symptoms	List of reported symptoms in the case description.
Race	List of the patients' race.
Travel history binary	0 if the patient has no travel history,
	1 if the patient has a travel history.
Chronic disease binary	0 if the patient hasn't a chronic disease,
	1 if the patient has a chronic disease.

In Equation (1), the maximum likelihood estimation technique is used to estimate the parameters, where Y is the binary response or class variable. In this equation, α is the intercept to be calculated, β_i is the estimated vector of parameters, and X_i is the vector of independent variables. While keeping all the remaining factors constant, the unit increase in the independent variables X_i will increase the likelihood ratio by $exp(\beta_i)$. This states the relative magnitude by which the response outcome (patient's condition) increases or decreases while considering a one-unit increase in the explanatory variable. The probability of the patient being dead (P_1) is given by:

$$P_1 = \left(\frac{exp(\alpha + \beta_i X_i)}{(1 + exp(\alpha + \beta_i X_i)} \right) \tag{2}$$

Similarly, the probability of hospitalization of recovered patients (P_0) is given by:

$$P_0 = \left(\frac{1}{(1 + exp(\alpha + \beta_i X_i)} \right) \tag{3}$$

We used the logit model to identify and select relevant factors that may affect the likelihood of Covid-19 severity.

Association Rule Mining Association Rule Mining (ARM) is one of the key techniques to discover and extract useful information from a large dataset. Mining association rules [3] can formally be defined as Let $I = i_1, i_2, i_3, ..., i_n$, be a set of n binary attributes called items, and Let, $D = t_1, t_2, t_3, ..., t_m$ be a set of transactions called the database. Each transaction in D has a unique transaction ID and contains a subset of items in I. A rule is defined as an implication of form $X \rightarrow Y$ where $X, Y \subseteq I$. The sets of items or itemset X and Y are called antecedent (left-hand-side or LHS) and consequent (right-hand-side or RHS) of the rule, respectively. Often rules are restricted to only a single item in the consequent. Association rules are rules that surpass user-specified minimum support and minimum confidence thresholds. The support $supp(X)$ of

an itemset X is defined as the proportion of transactions in the dataset, which contains the item set and confidence of a rule as defined as:

Definition 1. *The* Support *of an itemset X for a set of transactions T, denoted by $Supp(X)$, is the ratio of transactions that contain all items of X (number of transactions that satisfy X) [32]:*

$$Supp(X) = \frac{|\{t \in T \mid X \subseteq t\}|}{|T|}$$

Definition 2. *The* confidence *of an association rule $X \Rightarrow Y$ concerning a set of transaction T, denoted by $Conf(X \Rightarrow Y)$ is the percentage of transactions that contains X which also includes Y. Technically, the confidence of an AR is an estimation of the conditional probability of Y over X:*

$$Conf(X \Rightarrow Y) = \frac{Supp(X \cup Y)}{Supp(X)}.$$

Definition 3. *The* lift *of an association rule $X \Rightarrow Y$, denoted by $Lift(X \Rightarrow Y)$, is used to measure misleading rules that satisfy minimum support and minimum confidence threshold. The Lift measure is also used to calculate the deviation between an antecedent X and a consequent Y, which is the ratio of the joint probability of X and Y divided by the product of their marginal probabilities.*

$$Lift(X \Rightarrow Y) = \frac{Supp(X \cup Y)}{Supp(X) \times Supp(Y)}$$

In ARM, when the number of association rules is too large to be presented to a data mining expert or even treated by a computer, measures of interestingness can filter the interesting association rules. After support, confidence, and lift, more than fifty different measures of interestingness are in the literature [33,34]. These measures of interestingness are discussed in detail in the literature [35,36]. Initially, ARM was limited to large transactional datasets. Still, later, Han et al., Lu et al., Imielinski et al., and Nguyen et al. [37,38,39,40] presented different views on multi-level and multi-dimensional ARM. Over the years, different ARM frameworks [41] and the use of ARM in varied application scenarios [42,43] have also been discussed in the state-of the-art [6].

It can be interpreted as the deviation of the support of the whole rule from the support expected under independence, given the support of both sides of the rule. Greater lift values (≥ 1) indicate stronger associations. Measures like support, confidence, and lift are generally called interest measures because they help focus on potentially more interesting rules. For example, consider a rule such as $\{milk, sugar\} \Rightarrow \{bread\}$ with support of 0.1, confidence of 0.9, and lift of 2. Now, we know that 10% of all transactions contain all three items together; thus, the estimated conditional probability of seeing bread in a transaction under the condition that the transaction also contains milk and sugar is 0.9; and we see the items together in transactions at double the rate we would expect under independence between the item sets milk, sugar, and bread [44]. Rules can

be generated from datasets with specified classes as their consequences under class association rule mining. These rules are $\{A_1, A_2, A_3, ..., A_n \Rightarrow class\}$. The objective is to use specific search techniques to find all rules with the specified classes as their consequences that satisfy support and confidence [45,46].

Appropriate support and confidence values are the key to generating rules since keeping a very low support value will generate extensive rules, and if the support value is too high, we may lose rare but essential rules. In this paper, we generated rules from the dataset having specified classes such as rules or characteristics of patients who have been hospitalized or recovered. We also generated or mined rules for dead patients. Our goal is to find rules or characteristics rules for these two groups.

The steps for the implementation were the following:

- Implement the required libraries
- Exploring the data
- Transformation of data to lists
- Constructing the model
- Visualize the results

4 Experiments and Results

Association rule mining has been applied to the dataset. By selecting the optimum support and confidence value, we mined strong rules for both patient groups(recovered and died). This section discusses the logit model and association rule mining results. Moreover, interprets a few strong rules for both groups.

4.1 Logit Model Estimations

The binary logit regression model was used to estimate the coefficients of significant explanatory variables in the final model. The software package SAS was used for the model development. For the model, all attributes were used as input for the likelihood of death and recovery or hospitalization. Table 7 shows the significant predictor variables at the corresponding significance levels in the binary logit model, which can contribute to our research. Positive coefficients show that the probability of deterioration of the condition of the patients will increase by a certain amount for the specific predictor variables. Table7, shows that chronic disease, age group, race, and symptoms have a positive relationship with the condition of the patients. However, travel history and race type have a negative relationship.

4.2 Generating Strong Rules

We aim to extract characteristics of Covid-19 patients who have died or been hospitalized and recovered. We generated rules using the association rule technique with the specified support and confidence. We defined the consequent of a

Table 7. Predictor Variables With Corresponding P Values.

Parameter	DF	Estimate	Std. Error	Wald Chi-Square	Pr >Chisq
Intercept	1	-8.1897	0.0445	26578	<.0001
Symptoms	1	0.336	0.00218	8560	<.0001
Age group	1	0.1076	0.0119	91	<.0001
Race	1	-1.9671	0.0382	6851	<.0001
Travel history binary	1	-0.0175	0.00461	215	<.0001
Chronic disease binary	1	0.0148	0.00461	69	<.0001

rule to get our target rules that represent the characteristics of the patients who have died (Died = Yes) or who have been hospitalized or recovered (Died = No). Support and confidence play an essential role in rule generation. Initially, we set the minimum support and confidence values to 30% and 80%, respectively. Also, we set the minimum length to 3, which means that the generated rules should have at least three items, including the consequent. With these specified parameters, the algorithm generated 48 rules, and after pruning redundant rules, we got 27. From these 25 rules, ten rules whose lift values are greater than or equal to one are shown in Table 8, sorted by higher lift value with corresponding support and confidence. The software R was used for the experiments. It is worth mentioning that we did not obtain any rules for patients who have died for the specified support and confidence. This is due to the given values of support and confidence and also a tiny number of instances in which Covid-19 patients have died compared to those recovered or hospitalized (the ratio is about 1:17). To obtain the rules of dead patients after several experiments, we assigned the value of support to 10% but a high confidence value of 90% and obtained 59 rules.

Here, we set the consequent or class value to "yes" (Died = Yes) so we can only get the rules for dead Covid-19 patients. From these 59 rules, the top seven rules sorted by lift are shown in Table 9.

4.3 Interpretation of the Generated Strong Rules

We can see significant differences if we consider the rules of both groups of patients, dead and hospitalized or recovered. For died and recovered or hospitalized individuals, its observed confidence, which indicates how often the rule is true in the dataset, is very high (close to 100%). Regarding support, which demonstrates how frequently the item set or factors appear in the dataset, it is high (more than 30%) for Covid-19 patients. However, for recovered and hospitalized patients, the support value is very low (about 0.003%). For both groups, we can see the differences in the lift value that measures the degree of dependence between the antecedent and the consequent value. For recovered or hospitalized patients, the lift value is just above 1.0, which means the relationship between factors of these rules (antecedent part) and consequent are very low. On the other hand, for the dead Covid-19 patients, the lift value is very high (more

Table 8. Rules generated using the association rule technique with minimum support and confidence values 30% and 80% respectively

#	Antecedents	Consequents	Supp	Conf	Lift
1	{Race: White, Age Group: Teen, Symptom: Most Common}	{Condition of the Patient: Recovered or Hospitalized}	59	99	1.09
2	{Travel History: No, Race: Hispanic, Age Group: Senior, Symptom: Most Common}	{Condition of the Patient: Recovered or Hospitalized}	65	99	1.09
3	{Race: White, Age Group: Teen, Chronic Disease: No, Age Group: Middle-Aged, Travel History: Yes}	{Condition of the Patient: Recovered or Hospitalized}	54	99	1.08
4	{Race: Asian, Symptom: Less Common, Travel History: No, Age Group: Child}	{Condition of the Patient: Recovered or Hospitalize}	58	99	1.08
5	{Race: Asian, Travel History: No, Age Group: Middle-Aged, Chronic Disease: Yes}	{Condition of the Patient: Recovered or Hospitalized}	58	99	1.08
6	{Race: Asian, Symptom: Most Common, Age Group: Middle-Aged, Chronic_disease: No}	{Condition of the Patient: Recovered or Hospitalized}	57	99	1.08
7	{Race: Black, Symptom: Most Common, Age Group: Teen, Chronic Disease: Yes}	{Condition of the Patient: Recovered or Hospitalized}	45	99	1.08
8	{Race: White, Symptom: Less Common, Travel History: No, Age Group: Senior, Chronic Disease: No}	{Condition of the Patient: Recovered or Hospitalized}	31	94	1.04
9	{Symptom: Serious, Travel History: No, Age Group: Young adult, Chronic Disease: No, Race: Black}	{Condition of the Patient: Recovered or Hospitalized}	34	94	1.04
10	{Travel History: Yes, Age Group: Middle-Aged, Chronic Disease: No}	{Condition of the Patient: Recovered or Hospitalized}	63	94	1.04

Table 9. Generated rules using association rule technique with minimum support and confidence of 10% and 90%, respectively and with fixed consequences for dead Covid-19 patients.

#	**Antecedents**	**Consequents**	Supp	Conf	Lift
1	{Symptom: Serious, Age Group: Senior, Chronic Disease: Yes, Race: Asian}	{Condition of the Patient: Died}	0.003	1.0	12.2
2	{Symptom: Serious, Age Group: Senior, Chronic Disease: Yes, Race: Hispanic}	{Condition of the Patient: Died}	0.003	1.0	12.2
3	{Symptom: Serious, Age Group: Senior, Chronic Disease: Yes, Race: Asian}	{Condition of the Patient: Died}	0.003	0.9	12.2
4	{Symptom: Serious, Age Group: Middle-Aged, Chronic Disease: No, Race=White}	{Condition of the Patient: Died}	0.003	0.9	12.2
5	{Symptom: Serious, Age Group: Young Adult, Chronic Disease: Yes, Race: Hispanic}	{Condition of the Patient: Died}	0.002	0.89	12.1
6	{Symptom: Serious, Age Group: Middle-Aged, Chronic Disease: No, Race: Asian}	{Condition of the Patient: Died}	0.002	0.89	11.8
7	{Symptom: Most Common, Age Group: Senior, Chronic Disease: Yes, Race: Hispanic, Travel History: Yes}	{Condition of the Patient: Died}	0.002	0.88	11.8
8	{Symptom: Most Common, Age Group: Senior, Chronic Disease: Yes, Race: Hispanic, Travel History: Yes}	{Condition of the Patient: Died}	0.002	0.88	11.8
9	{Symptom: Most Common, Age Group: Senior, Chronic Disease: Yes, Race: Hispanic, Travel History: Yes}	{Condition of the Patient: Died}	0.002	0.88	11.7
10	{Symptom: Most Common, Age Group: Senior, Chronic Disease: Yes, Race: Hispanic, Travel History: Yes}	{Condition of the Patient: Died}	0.002	0.88	11.5

than 12.0), indicating a more significant association between the antecedent and the consequent factors.

5 Discussion

One of the most challenging aspects of public health is predicting the occurrence of contagious diseases, as these predictions can significantly influence the way people live and the level of health care they receive. Using a reliable prediction, individuals and clinicians will be able to make informed decisions, and clinicians will be able to select the most effective treatment and prevention strategies for their patients based on the most accurate and reliable information. Despite recent research investigating various data mining techniques to assist clinicians in diagnosing patients with Covid-19, an accurate prediction model for this disease remains an elusive goal. We present an investigation of association rules for Covid-19 patients using data mining techniques. By utilizing clinical risk factors in the target population, association rules can be developed to predict severe cases of Covid-19. Nevertheless, any prediction should be combined with clinical judgment and one's assessment of the patient's situation. It is necessary to address several shortcomings in this paper. Although we used a large set of Covid-19 data, we had no control over the data quality, regardless of how robust the dataset was. We also have a limited number of features in our dataset. The support value for Covid-19 patients is low; however, we have established a high confidence value to demonstrate how predictive the rules are.

6 Conclusion

Association rule mining has been used to extract valuable rules from the Covid-19 dataset of risk factors. Using the logit model, we tested the statistical significance of all predictors before applying association rule mining. We analyzed data from dead and recovered patients and hospitalized patients with specific support and confidence. Based on the experimental outcomes, both groups of experiments produced the strongest confidence levels for the generated rules. The Covid-19 dataset contains fewer cases of patients dying, compared with a more significant number of patients recovering or hospitalized, which forces us to set the support level at a low level. As part of our analysis, we also extracted strong rules from a large set of generated rules and interpreted those rules accordingly. This research aims to improve risk prediction for individuals who may be exposed to infectious diseases in the future. We intend to expand this research by applying the concept of association rule mining to dynamic data sets in future work. Several updates are made to the Covid-19 web data statistics regularly. Our method for extracting the significant Covid-19 symptoms in the current scenario relies on static data sets; therefore, it is not applicable in a dynamic environment. As a result, the database patterns must be extracted using dynamic algorithms. The use of dynamic rule mining algorithms has been reported in the literature [47], but we aim to extend the same approach to Covid-19 data sets by applying

an association rule mining algorithm. However, it is essential to note that the main challenge associated with Covid-19 web data is that they are noisy. Hence, investigating the quality of the results produced in future studies is worthwhile.

Acknowledgements

This work has been conducted in the project "ICT programme" which was supported by the European Union through the European Social Fund.

References

1. Rasheed, J., Jamil, A., Hameed, A.A., Aftab, U., Aftab, J., Shah, S.A., Draheim, D.: A survey on artificial intelligence approaches in supporting frontline workers and decision makers for the COVID-19 pandemic. Chaos, Solitons & Fractals 141, 110337 (2020), https://www.sciencedirect.com/science/article/pii/S0960077920307323
2. Zhang, S., Webb, G.I.: Further pruning for efficient association rule discovery. In: Australian Joint Conference on Artificial Intelligence. pp. 605–618. Springer (2001)
3. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD international conference on Management of data. pp. 207–216 (1993)
4. Brijs, T., Swinnen, G., Vanhoof, K., Wets, G.: Using association rules for product assortment decisions: A case study. In: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 254–260 (1999)
5. Chen, Y., Li, F., Fan, J.: Mining association rules in big data with ngep. Cluster Computing 18(2), 577–585 (2015)
6. Shahin, M., Peious, S.A., Sharma, R., Kaushik, M., Syed Attiqe, S., Yahia, S.B., Draheim, D.: Big data analytic in association rule mining: A systematic literature review. In: Proceedings of the International Conference on Big Data Engineering and Technology ((in press) 2021)
7. Shahin, M., Heidari Iman, M., Kaushik, M., Sharma, R., Ghasempouri, T., Draheim, D.: Exploring factors in a crossroad dataset using cluster-based association rule mining. In: International Conference on Ambient Systems, Networks and Technologies (ANT) (2022)
8. Agrawal, R., Srikant, R., et al.: Fast algorithms for mining association rules. In: Proc. 20th int. conf. very large data bases, VLDB. vol. 1215, pp. 487–499. Citeseer (1994)
9. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. ACM sigmod record 29(2), 1–12 (2000)
10. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: Parallel algorithms for discovery of association rules. Data mining and knowledge discovery 1(4), 343–373 (1997)
11. Shahin, M., Inoubli, W., Shah, S.A., Yahia, S.B., Draheim, D.: Distributed scalable association rule mining over covid-19 data. In: International Conference on Future Data and Security Engineering. pp. 39–52 (2021)
12. Bertl, M., Shahin, M., Ross, P., Draheim, D.: Finding indicator diseases of psychiatric disorders in bigdata using clustered association rule mining. In: Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing. pp. 826–833 (2023)

13. Kaushik, M., Sharma, R., Peious, S.A., Shahin, M., Yahia, S.B., Draheim, D.: On the potential of numerical association rule mining. In: International Conference on Future Data and Security Engineering. pp. 3–20. Springer (2020)
14. Kaushik, M., Sharma, R., Peious, S.A., Shahin, M., Yahia, S.B., Draheim, D.: A systematic assessment of numerical association rule mining methods. SN Computer Science 2(5), 1–13 (2021)
15. TAŞER, P.Y., BİRANT, K.U., Birant, D.: Multitask-based association rule mining. Turkish Journal of Electrical Engineering & Computer Sciences 28(2), 933–955 (2020)
16. Shahin, M., Shah, S.A., Sharma, R., Ghasempouri, T., Poveda, J.A., Fahringer, T., Draheim, D.: Performance of a distributed apriori algorithm using the serverless functions of the apollo framework. In: Computer Science On-line Conference CSOC2024. pp. 1–14. Springer (2024)
17. Sharma, R., Kaushik, M., Peious, S.A., Shahin, M., Yadav, A.S., Draheim, D.: Towards unification of statistical reasoning, olap and association rule mining: semantics and pragmatics. In: International Conference on Database Systems for Advanced Applications. pp. 596–603. Springer (2022)
18. Arakkal Peious, S., Sharma, R., Kaushik, M., Shahin, M., Draheim, D.: On observing patterns of correlations during drill-down. In: International Conference on Information Integration and Web Intelligence. pp. 134–143. Springer (2023)
19. Liu, X., Niu, X., Fournier-Viger, P.: Fast top-k association rule mining using rule generation property pruning. Applied Intelligence 51(4), 2077–2093 (2021)
20. Zhou, Y., Wang, Y., Li, C., Ding, L., Mei, Y.: Coupled risk analysis of hospital infection: a multimethod-fusion model combining association rules with complex networks. Computers & Industrial Engineering p. 109720 (2023)
21. Sonet, K.M.H., Rahman, M.M., Mazumder, P., Reza, A., Rahman, R.M.: Analyzing patterns of numerously occurring heart diseases using association rule mining. In: 2017 Twelfth International Conference on Digital Information Management (ICDIM). pp. 38–45. IEEE (2017)
22. Ivančević, V., Tušek, I., Tušek, J., Knežević, M., Elheshk, S., Luković, I.: Using association rule mining to identify risk factors for early childhood caries. Computer Methods and programs in Biomedicine 122(2), 175–181 (2015)
23. Borah, A., Nath, B.: Identifying risk factors for adverse diseases using dynamic rare association rule mining. Expert systems with applications 113, 233–263 (2018)
24. Sharma, S.: Concept of association rule of data mining assists mitigating the increasing obesity. In: Healthcare Policy and Reform: Concepts, Methodologies, Tools, and Applications, pp. 518–536. IGI Global (2019)
25. Cai, R., Liu, M., Hu, Y., Melton, B.L., Matheny, M.E., Xu, H., Duan, L., Waitman, L.R.: Identification of adverse drug-drug interactions through causal association rule discovery from spontaneous adverse event reports. Artificial intelligence in medicine 76, 7–15 (2017)
26. Ramasamy, S., Nirmala, K.: Disease prediction in data mining using association rule mining and keyword based clustering algorithms. International Journal of Computers and Applications 42(1), 1–8 (2020)
27. Kamalesh, M.D., Prasanna, K.H., Bharathi, B., Dhanalakshmi, R., Aroul Canessane, R.: Predicting the risk of diabetes mellitus to subpopulations using association rule mining. In: proceedings of the international conference on soft computing systems. pp. 59–65. Springer (2016)
28. Nahar, J., Imam, T., Tickle, K.S., Chen, Y.P.P.: Association rule mining to detect factors which contribute to heart disease in males and females. Expert Systems with Applications 40(4), 1086–1093 (2013)

29. Lau, A., Ong, S.S., Mahidadia, A., Hoffmann, A., Westbrook, J., Zrimec, T.: Mining patterns of dyspepsia symptoms across time points using constraint association rules. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. pp. 124–135. Springer (2003)
30. Xu, B., Gutierrez, B., Mekaru, S., Sewalk, K., Goodwin, L., Loskill, A., Cohn, E.L., Hswen, Y., Hill, S.C., Cobo, M.M., et al.: Epidemiological data from the COVID-19 outbreak, real-time case information. Scientific Data 7(1), 1–6 (2020)
31. Seddik, A.F., Shawky, D.M.: Logistic regression model for breast cancer automatic diagnosis. In: 2015 SAI Intelligent Systems Conference (IntelliSys). pp. 150–154. IEEE (2015)
32. Daniel T. Larose, C.D.L.: Discovering Knowledge in Data, chap. Association Rules, pp. 247–265. John Wiley & Sons, Ltd (2014), https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118874059.ch12
33. Geng, L., Hamilton, H.J.: Interestingness measures for data mining: A survey. ACM Computing Surveys 38(3), 1–32 (Sep 2006), https://doi.org/10.1145/1132960.1132963
34. Liu, B., Hsu, W., Chen, S.: Using general impressions to analyze discovered classification rules. In: Proceedings of KDD'97 – the 3rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 31–36. AAAI Press (1997)
35. Bastide, Y., Pasquier, N., Taouil, R., Stumme, G., Lakhal, L.: Mining minimal non-redundant association rules using frequent closed itemsets. In: Proceedings of CL'2000 – the 1st International Conference on Computational Logic. pp. 972–986. Springer Berlin Heidelberg (2000)
36. Hilderman, R.J., Hamilton, H.J.: Measuring the interestingness of discovered knowledge: A principled approach. Intelligent Data Analysis 7(4), 347–382 (2003)
37. Han, J., Fu, Y.: Discovery of multiple-level association rules from large databases. In: VLDB. vol. 95, pp. 420–431. Citeseer (1995)
38. Lu, H., Feng, L., Han, J.: Beyond intratransaction association analysis: Mining multidimensional intertransaction association rules. ACM Transactions on Information Systems 18(4), 423–454 (Oct 2000), https://doi.org/10.1145/358108.358114
39. Imielinski, T., Khachiyan, L., Abdulghani, A.: Cubegrades: Generalizing association rules. Data Mining and Knowledge Discovery 6(3), 219–257 (2002)
40. Nguyen, K.N.T., Cerf, L., Plantevit, M., Boulicaut, J.F.: Multidimensional association rules in boolean tensors. In: Proceedings of the 2011 SIAM International Conference on Data Mining. pp. 570–581. SIAM (2011)
41. Fister, I., Fister Jr, I.: uARMSolver: A framework for association rule mining. CoRR arXiv:2010.10884 [cs.DB] (2020)
42. Fister Jr, I., Fister, I.: Association rules over time. CoRR arXiv:2010.03834 [cs.NE] (2020)
43. Fournier-Viger, P., Li, J., Lin, J.C.W., Chi, T.T., Uday Kiran, R.: Mining cost-effective patterns in event logs. Knowledge-Based Systems 191, 105241 (2020), https://www.sciencedirect.com/science/article/pii/S0950705119305581
44. Hahsler, M., Grün, B., Hornik, K.: Introduction to arules–mining association rules and frequent item sets. SIGKDD Explor 2(4), 1–28 (2007)
45. Paul, R., Groza, T., Hunter, J., Zankl, A.: Inferring characteristic phenotypes via class association rule mining in the bone dysplasia domain. Journal of biomedical informatics 48, 73–83 (2014)
46. Lin, C.W., Hong, T.P., Lu, W.H.: Using the structure of prelarge trees to incrementally mine frequent itemsets. New Generation Computing 28(1), 5–20 (2010)

47. Aqra, I., Abdul Ghani, N., Maple, C., Machado, J., Sohrabi Safa, N.: Incremental algorithm for association rule mining under dynamic threshold. Applied Sciences p. 5398 (2019)

Appendix 10

X

M. Shahin, MRH. Iman, M. Kaushik, R. Sharma, T. Ghasempouri, and D. Draheim, "Exploring Factors in a Crossroad Dataset Using Cluster-Based Association Rule Mining," *2022 the 13th International Conference on Ambient Systems, Networks and Technologies (ANT)*, Porto, Portugal, 2022, pp. 231-238, DOI: https://doi.org/10.1016/j.procs.2022.03.032.

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 201 (2022) 231–238

Procedia Computer Science

www.elsevier.com/locate/procedia

The 13th International Conference on Ambient Systems, Networks and Technologies (ANT)
March 22 - 25, 2022, Porto, Portugal

Exploring Factors in a Crossroad Dataset Using Cluster-Based Association Rule Mining

Mahtab Shahin[a,c,*], Mohammad Reza Heidari Iman[b], Minakshi Kaushik[a], Rahul Sharma[a], Tara Ghasempouri[b], Dirk Draheim[a]

[a]*Information Systems Group, Tallinn University of Technology, Akadeemia tee 15a, 12618 Tallinn, Estonia*
[b]*Department of Computer Systems, Tallinn University of Technology, Akadeemia tee 15a, 12618 Tallinn, Estonia*
[c]*Computer Science, University of Innsbruck, Innsbruck, Tirol, Austria, 6020*

Abstract

Investigating the contributory factors in crossroad accidents is a high-priority issue in the traffic safety analysis. This study exploits a method based on association rules to analyze these contributory factors. Using data about one year of crossroad traffic accidents in Isfahan, Iran, 63 and 156 association rules are generated for non-serious and serious accidents, respectively. The results show that both accident severity levels are associated with head-to-the-side collisions and the spring season. The frequency of non-serious accidents is about 38% higher than that of serious accidents. However, the association analysis results show that serious accidents are associated with more influencing factors than non-serious. Seat belt usage and road surface condition are additional decisive factors for serious accidents but not so for non-serious. The association analysis reveals that many influencing factors (such as traffic lights and the existence of a traffic enforcement camera) exhibit effects only under some specific circumstances (e.g., the peak of traffic).

Keywords: Association rule mining, frequent item generation, accident dataset, data mining, visualization

1. Introduction

Over the past decades, traffic safety problems have been increased continuously due to the rapid growth of traffic volume, resulting in over a million road traffic fatalities, up to 50 million injuries, and costs of trillions of dollars. Moreover, according to the WHO [1], 90 percent of these fatal accidents occur in low and middle-income countries. Damages can be financial or personal, which in some cases are irreparable.

* Corresponding author. Tel.: +372-5821-0650
E-mail address: mashah@ttu.ee

10.1016/j.procs.2022.03.032

The research conducted on the cost of traffic accidents in a middle-income country, Iran, by Ayati [2], estimates the average cost of traffic accidents and the related factors. He calculated motor vehicle accident costs, including fines, medical expenses, administrative costs, vehicle damage, and other items. The cost for all these items regarding the traffic accidents in Iran (urban and suburban) in 2001 was about 40 billion dollars, which is more than three percent of gross domestic product (GDP) in the same year [2]. A significant number of studies have been analyzed the traffic accidents data in countries with different income categories and have investigated the effect of various factors on accidents. Despite all progress in analyzing such data, there remain several challenges in estimating the number of fatal/injury accidents, including traffic parameters, geometrical design, and the features of the controlling traffic system. Reducing accidents in crossroads can only be done by identifying the factors contributing to accidents, carefully designing crossroads, and comprehensive traffic safety laws. Moreover, some other factors such as enforcing the law, educating drivers and pedestrians, and encouraging them to follow the rules can reduce accidents at crossroads.

The nature of accident data is heterogeneous, making it difficult to analyze. A problem with heterogeneous data is that some relationships between features are hidden. For more appropriate analysis and more accurate results, it is necessary to eliminate this anomaly. Matthew and Tarku [3] have divided the data into different groups (such as road conditions and accident cause) and examined each group separately. The main problem with this type of classification is the unequal distribution of features in each group. For example, some subgroups will have more samples, and some will have fewer samples.

Although several studies have been conducted on the analysis of crossroad accident data [4], their focus was mainly on the relationship between parameters. Thus, it is necessary to analyze the characteristics and contributory factors which can lead to accident casualties. Hence, special attention should be paid to the associated factors that simultaneously impact the crossroad accident risk. This study employs the association rule approach to examine a crossroad accident dataset's characteristics and contributory factors. The contributions of this study are as follows.

- First, we extract numerous intriguing rules by mining the association rules to study the hidden correlations among the crossroad accident dataset's fundamental characteristics and contributory elements. In addition, we look at the interactions between these variables to better comprehend the crossroad accident dataset's overall trends.

- Second, we can comprehend these connection rules using the data visualization technique, providing helpful information for prioritizing countermeasures in minimizing the crossroad accident dataset risk.

The rest of this paper is organized as follows. Background and related work in Sect. 2, followed by our methodology in Sect. 3. The experimental results from implementation are presented in Sect. 4 and finally Sect. 5 concludes the paper.

2. Related Work

Up to now, numerous researches have been developed to analyze accident risk parameters. The majority of the studies applied parametric models. For example, Chang and Wang [5] proposed a non-parametric tree-based model to evaluate the influenced risk factors to injury severity in traffic accidents. They analyzed the Taipei area traffic dataset and showed that pedestrians, motorcycles, and bicycle riders are the most vulnerable groups on the road. However, note that the non-parametric methods may suffer from an overfitting problem. More importantly, such methods also require a large amount of data for the modeling analysis, especially when there are many explanatory variables. Valent et al. [6] have studied the effects of restraint devices such as seatbelts or helmets on the injury severity levels. The findings indicated that using restraint devices mainly reduces the injury severity in traffic accidents. Zhang et al. [7] attempted to identify groups of drivers with a greater risk of being injured or killed in traffic accidents. The results showed that elderly drivers were the most vulnerable in traffic accidents.

Other researchers employed advanced statistical and artificial intelligence methods to investigate different accident datasets. For instance, Xu et al. [8] applied geographically weighted regression to link crash frequency at traffic analysis zone (TAZ) with jobs-housing ratio and other contributing factors. Prato et al. [9] used Kohonen neural networks to a database of fatal pedestrian accidents.

The findings revealed that most fatal run-off-road ROR collisions are caused by complex interactions between humans, roads, and cars. When creating countermeasures for minimizing fatal ROR crash frequency, such interacting consequences should be considered.

The association rule approach is one of the most fundamental and well-known data mining techniques. It can deal with datasets comprising several variables and explore their relationships if proper support and confidence are provided. Compared with traditional parametric approaches, association rule mining does not require any assumptions or functional forms to be specified. Association rules have the advantage over non-parametric methods in that they can also be used for a few observations [10]. Geurts et al. [11] used standard item sets to identify accident patterns. Montella et al. [12] explored the correlation between the contributing factors of different types of collisions that occur at urban roundabouts.

3. Methodology

3.1. Dataset Description

The crossroad accident data used in this experiment were taken from [4]. This collected dataset comprised a record of 576 vehicles involved in an accident in 2014 and was collected from the accident database in Isfahan, Iran. Each record extracted from the database includes the following information: (i) accident data, (ii) external environment, (iii) traffic characteristics, and (iv) control status. The accident severity was categorized into two levels: serious and non-serious accidents. Serious accidents are the resulting death or serious injuries that can last for a long time(i.e., coma, paralyzation). On the other hand, non-serious accidents mainly caused financial loss, not life-threatening injuries. Moreover, peak refers to when traffic reaches its highest level in the morning or afternoon; 7:30-9 AM and 12-13:30 PM have been considered as the peak of traffic. The details of the dataset are provided in Table 1. This table also presents the variable's proportions for different severity levels. For instance, the proportion of non-serious accidents was much higher than the corresponding proportion for serious accidents. In addition, it can be seen that serious accidents are a little more likely to occur at night, whereas non-serious accidents are more likely to occur during the day. Because the effects of influencing factors may vary with different accident severity levels, there is a significant need to investigate the association rules for non-serious and serious accidents separately.

3.2. Association Rule Mining

Association rule mining is a well-known technique for exploring relationships among variables in large databases [13]. The main objective of association rule mining is to examine groups of items that frequently occur together in the given dataset. Compared with the classical parametric and non-parametric methods, the association rule technique has the advantage of flexible application because no specified function and no dependent variables are needed. Based on the obtained association rules, countermeasures can be taken to break the associations and decrease the likelihood of serious accidents for useful applications. For instance, one association rule for serious accidents is the following: {Using seat belt=Not in use, Lighting=Night, Pedestrian=Yes} $\rightarrow$ {Accident severity=Fatal}. This rule indicates that serious accidents are associated with the circumstances in seat belts and lighting. Hence, one primary focus should be on avoiding accidents at night and the drivers who were not wearing seat belts.

3.3. Definition

Association rule mining intends to find out the strong rules by using diverse measurements [14]. Three parameters measure the number of rules to be generated: Support, Confidence and Lift. Suppose X, Y are the independent attributes; therefore, these three parameters for Rule X $\rightarrow$ Y can be calculated as defined below.

- *Support* The value of support indicates the proportion of an accident occurrence by finding several accident cases containing a particular accident type divided by the total number of accidents, which can be determined

Table 1: Descriptive statistics of crossroad dataset.

Item	Related factor	Description	Proportion		
			Total	Non-serious	Serious
Accident data	Gender of driver	Female	191	83	108
		Male	385	289	96
	Age of driver	$15 \leq$ age ≤ 19	31	28	3
		$19 \leq$ age ≤ 40	396	298	98
		$41 \leq$ age ≤ 60	101	22	79
		$61 \leq$ age ≤ 80	48	31	17
	License status	Yes (valid)	467	369	98
		Expired	18	11	7
		No (no license)	90	28	62
	Using a seat belt	In use	523	386	137
		Not in use	53	12	41
	Collision type	Pedestrian involvement	91	87	4
		Side swipe	144	69	74
		Rear-end	23	16	7
		Head-on	31	25	6
		Head to side	181	121	60
		Stationary object	106	99	7
	Accident severity	Injury	185	102	83
		Fatal	56	-	56
		Financial	335	201	134
	Pedestrian involved	Yes	151	95	56
		No	425	239	186
External environment	Lighting	Night	85	23	62
		Day	491	323	168
	Time	Off peak	80	69	11
		Morning peak	195	102	93
		Evening peak	301	191	110
	Season	Spring	136	101	35
		Summer	153	89	64
		Fall	99	83	16
		Winter	188	161	27
	Road surface conditions	Dry	268	215	53
		Slippery (wet, snow)	308	215	93
Traffic characteristics	Number of lanes	One-lane	96	29	67
		Two-lane	480	350	130
	Angle between branches of crossroad	obtuse	207	195	12
		quadrant	369	311	58
Control status	Traffic light	Pre-scheduled	496	268	228
		Intelligent	80	43	37
	Traffic enforcement camera	Yes	75	36	39
		No	501	192	309

as follows:

$$Supp(X \rightarrow Y) = \frac{\mathsf{P}(X \cap Y)}{N} \tag{1}$$

- *Confidence* The value of confidence is the proportion of events A and B together to event A alone. The higher values of confidence indicate the more likelihood of happening B with the occurrence of A.

$$Conf(X \rightarrow Y) = \frac{P(X \cup Y)}{P(X)} \tag{2}$$

Furthermore, $Supp(X \cup Y) \geqslant \sigma$, and $Conf(X \cup Y) \geqslant \delta$. Where σ, and δ are the minimum support and minimum confidence, respectively.

In general, we can gain $2^k - 2$ association rules at maximum from each frequent $k - itemset$ indicated by F, ignoring rules that have empty antecedents or consequences. As there are many association rules satisfying the support and the confidence, a practical measure to filter or rank the found rules is the lift, which suggests the deviation of the support of the whole rule from the one expected under independence given both sides of the rule's supports.

- *Lift* The value of lift can be interpreted in the three cases: (i) if Lift ($X \rightarrow Y$)=1, then X and Y are independent. (ii) if Lift (X→Y)>1 (positive correlation) X and Y, most likely happen together. (iii) if ($X \rightarrow Y$)<1 (negative correlation) X and Y, are very rare to happen together.

$$Lift(X \rightarrow Y) = \frac{(X \cup Y)}{P(X) \times P(Y)} \tag{3}$$

3.4. Frequent Itemset Generation

There are three primary steps employed to generate frequent itemsets satisfying the min support threshold: (i) scan the database and compute the support, (ii) generate and compare frequent itemsets, and (iii) generate candidate itemsets. More precisely, let C_k indicate the set of candidate k-itemsets, and F_k refer to the set of frequent k-itemsets. Initially, we create a single pass through the dataset to discover the support of each item and reach the set of all frequent 1-itemsets. Next, we iterative generate new candidate k-itemsets C_k applying the frequent (k-1)-itemsets discovered in the previous iteration. Afterward, we will recognize all candidate itemsets C_k contained in each transaction t by computing the support of the candidates. Those candidate itemsets whose support counts are less than min support are eliminated in this step. Finally, the sub-procedure of generated frequent itemset is finished when new frequent itemsets are not created, namely, $F_k = \emptyset$.

4. Results and Findings

4.1. Rule Generation Results

Three well-known algorithms are available for mining the frequent itemsets: Apriori, FP-growth, and Eclat. It has been considered that Apriori Algorithm shows phenomenal performance due to its high accuracy [13, 15, 16]. Hence, this calculation is chosen to mine the association rules for the transaction dataset in this research. Apriori algorithm includes two separate steps: (1) All of the frequent itemsets in the database are found using minimum support, and (2) these frequent itemsets, along with the minimum confidence constraint, are utilized to build rules. The Apriori algorithm [10] provided by the "arules" package of the R software was employed to mine association rules from a crossroad accident dataset, including 576 transactions related to two different types of accident in this study. The support and confidence thresholds were valued at 0.3 and 0.5, respectively. To get rules of a high-quality, lift amounts

greater than 1.1 were accepted. Lastly, 63 rules for serious accidents and 156 rules for non-serious accidents were generated.

Fig. 1 demonstrates the association rules using group matrix plots in R-extension packages [17]. Using the k-means clustering technique, the antecedents were separated into 10 groups. The grouped matrix plot in Figure 1 is a balloon plot in which the antecedents are grouped as columns, and the consequences are grouped as rows. The colors of the balloons represent the total lift, which means the relative strength of the elements' inter-dependency. The aggregated support, which indicates the relative frequency of occurrence of the factor combination(s) involved, is represented by the size of the balloon. A tiny dark balloon, for example, would suggest moderately strong factor inter-dependency but the relatively rare occurrence of the factor combination (s), and a sizeable light balloon denotes a weaker (but still significant) interdependence of factors but a higher frequency of the factor combination (s). As shown in Figure 1 among all the rules, the important association rules relate to lighting, pedestrian involvement, and road surface.

4.2. Association rule analysis of contributory factors

In Tables 2 and 3, association rules were reported separately for serious and non-serious accidents, along with support, confidence, and lift.

Moreover, the first nine generated association rules are ranked according to the lift value in each table, respectively. Rule #1 in Table 2 illustrates that if an accident occurred during a day, it is probable to be a financial accident severity. Rules #2 and #3 show the lighting condition, where non-serious accidents occurring in a day are more likely to involve stationary type collision.

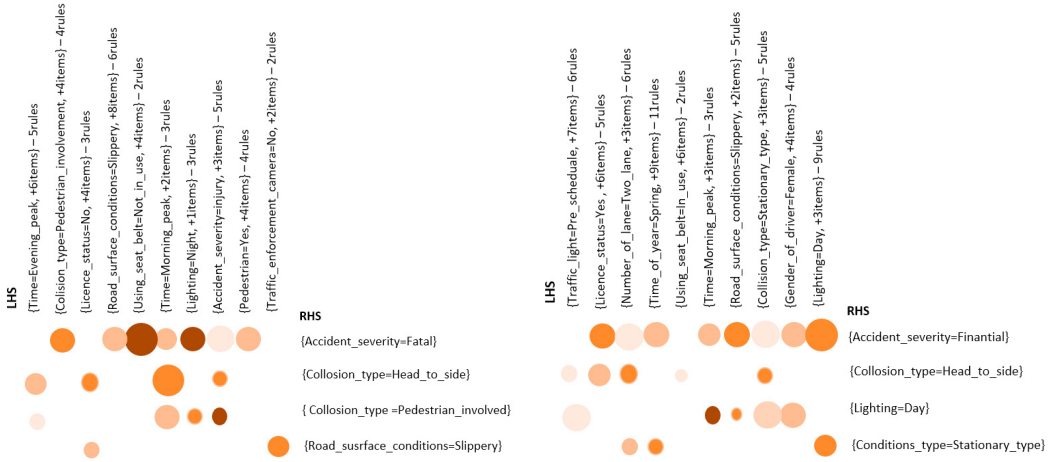
In Table 3 Rule #9 is related to slippery, wet, or icy road surface conditions. This rule suggests that an accident on a morning peak is more likely fatal. The support of this rule is 0.40, indicating that the rule has a relatively high frequency in serious accident data. Accordingly, increased attention should be given to developing effective countermeasures on these kinds of road surface conditions and time of day. Other important factors contributing to serious cross-road accidents are related to pedestrians. Rules #1, #2, #7 suggest that serious cross-road accidents are probably involved with pedestrians.

Table 2: The high lift rules for non-serious accidents.

Rules	**Association Rule Mining**		**S(%)**	**C(%)**	**L**
	Antecedent	**Consequent**			
1	{Lighting=Day, Collision type=Stationary type}	{Accident severity=Financial}	0.33	0.79	1.49
2	{Lighting=Day, Number of lanes=One-lane Time=Off peak}	{Accident severity=Financial}	0.35	0.77	1.45
3	{Lighting=Day, Season=Spring, Road surface conditions=Dry}	{Accident severity=Financial}	0.41	0.78	1.44
4	{Traffic light=Pre-scheduled, Time=Evening peak}	{Collision type=Head to side}	0.34	0.85	1.43
5	{License status=Yes(valid), Gender of driver=Female, Lighting=Day}	{Collision type=Head to side}	0.31	0.85	1.43
6	{Gender of driver=Female, Collision type=Head to side}	{Accident severity=Financial}	0.36	0.77	1.42
7	{Season=Spring, Lighting=Night, Road surface condition=Dry}	{Accident severity=Financial}	0.36	0.76	1.42
8	{Using seat belt=In use, Number of lane= Two lane, Road surface conditions=Slippery}	{Accident severity=Financial}	0.36	0.75	1.42
9	{Season=Spring, Using seat belt=Yes Traffic enforcement camera=Yes}	{Collision type=Side swipe}	0.36	0.75	1.42

Table 3: The high lift rules for serious accidents.

Rules	Association Rule Mining		S(%)	C(%)	L
	Antecedent	Consequent			
1	{Using seat belt=Not in use, Lighting= Night, Pedestrian=Yes}	{Accident severity=Fatal}	0.33	0.51	1.55
2	{Collision type=Pedestrian involvement, Pedestrian=Yes}	{Accident severity=Fatal}	0.31	0.53	1.54
3	{Time=Evening peak Accident severity=injury}	{Collision type=Head to side}	0.34	0.79	1.53
4	{Lighting=Day, Road surface condition=Slippery}	{Accident severity=Fatal}	0.31	0.80	1.50
5	{Time=Evening peak Lighting=Day}	{Collision type=Head to side}	0.36	0.79	1.49
6	{License status=No, Age of driver=15 ≤ age ≤ 19}	{Collision type=Pedestrian involvement}	0.35	0.79	1.47
7	{License status=No, Road surface conditions=Slippery, Pedestrian=Yes}	{Accident severity=Fatal}	0.41	0.57	1.47
8	{Time=Morning peak, Accident severity=Injury}	{Collision type=Head to side}	0.38	0.59	1.46
9	{Road surface condition=Slippery, Time= Morning peak, Collision type=Stationary object }	{Accident severity=Fatal}	0.40	0.63	1.44



(i) Serious accident (ii) Non-serious accident

Fig. 1: Visualization of association for two accident severity levels by group matrix

5. Conclusion

The present study aims to investigate factors contributing to serious casualty crashes and their inter-dependencies. The serious casualty crash data in 2014 was gathered from the crossroads of Isfahan, Iran. Each crash report was particularly examined and employed to investigate the characteristics of serious and non-serious accidents in terms of accident data, external environment, traffic characteristics, and control status. By applying different values for support and confidence, we gathered helpful information about the combination of accident characteristics to analyze the potential causes of non-serious and serious accidents, respectively. Together with the data visualization technique,

it provides more understandable results for researchers and traffic road officials. We generated 63 association rules for non-serious accidents and 156 for serious accidents using the Apriori algorithm. The results of the association rules show that {Using seat belt = In use} and {Traffic enforcement camera = No} are the two items with the highest frequency of the two accident severity levels, indicating that most crossroad accidents are related to using any seat belt or existence of any enforcement camera.

The findings of this study indicated that the mechanisms of serious accidents are different from those of typical non-serious accidents. For example, the accidents occurred on the rush hour and pedestrian involvement are more likely to be fatal accident, compared with stationary involvement and non-peak hour. The impacts of weather conditions were also different between serious accidents and non-serious accidents. Therefore, policymakers need to develop various safety improvement policy initiatives and technical countermeasures to reduce fatalities and injuries from major accidents involving accidents in certain circumstances. For example, to prevent pedestrians involved in accidents, some engineering improvements, such as installing warning signs, improving pavement conditions, and identifying crosswalks with sufficient light for drivers, should be implemented on crossroads. Finally, stricter speed requirements and other regulations should be considered to prevent serious accidents in adverse weather conditions.

Acknowledgements

This work has been conducted in the project “ICT programme” which was supported by the European Union through the European Social Fund.

References

[1] World Health Organization, Global status report on road safety 2015, World Health Organization, 2015.
[2] E. Ayati, The cost of traffic accidents in iran, Ferdowsi University of Mashhad, Mashhad, Iran (2002).
[3] M. G. Karlaftis, A. P. Tarko, Heterogeneity considerations in accident modeling, Accident Analysis & Prevention 30 (4) (1998) 425–433.
[4] M. Shahin, S. Syed Attiqe, M. Kaushik, R. Sharma, S. A. Peious, D. Draheim, Cluster-based association rule mining for an intersection accident dataset, in: Proceedings of the IEEE International Conference on Computing, Electronic and Electrical Engineering(ICECUBE), (in press) 2021.
[5] L.-Y. Chang, H.-W. Wang, Analysis of traffic injury severity: An application of non-parametric classification tree techniques, Accident Analysis & Prevention 38 (5) (2006) 1019–1027.
[6] F. Valent, F. Schiava, C. Savonitto, T. Gallo, S. Brusaferro, F. Barbone, Risk factors for fatal road traffic accidents in udine, italy, Accident Analysis & Prevention 34 (1) (2002) 71–84.
[7] J. Zhang, J. Lindsay, K. Clarke, G. Robbins, Y. Mao, Factors affecting the severity of motor vehicle traffic crashes involving elderly drivers in ontario, Accident Analysis & Prevention 32 (1) (2000) 117–125.
[8] C. Xu, H. Li, J. Zhao, J. Chen, W. Wang, Investigating the relationship between jobs-housing balance and traffic safety, Accident Analysis & Prevention 107 (2017) 126–136.
[9] C. G. Prato, V. Gitelman, S. Bekhor, Mapping patterns of pedestrian fatal accidents in israel, Accident Analysis & Prevention 44 (1) (2012) 56–62.
[10] J. Weng, J.-Z. Zhu, X. Yan, Z. Liu, Investigation of work zone crash casualty patterns using association rules, Accident Analysis & Prevention 92 (2016) 43–52.
[11] K. Geurts, I. Thomas, G. Wets, Understanding spatial concentrations of road accidents using frequent item sets, Accident Analysis & Prevention 37 (4) (2005) 787–799.
[12] A. Montella, Identifying crash contributory factors at urban roundabouts and using association rules to explore their relationships to different crash types, Accident Analysis & Prevention 43 (4) (2011) 1451–1463.
[13] R. Agrawal, T. Imieliński, A. Swami, Mining association rules between sets of items in large databases, in: Proceedings of the 1993 ACM SIGMOD international conference on Management of data, 1993, pp. 207–216.
[14] M. Kaushik, R. Sharma, S. A. Peious, M. Shahin, S. B. Yahia, D. Draheim, A systematic assessment of numerical association rule mining methods, SN Computer Science 2 (5) (2021) 1–13.
[15] M. Shahin, S. Arakkal Peious, R. Sharma, M. Kaushik, S. Ben Yahia, S. A. Shah, D. Draheim, Big data analytics in association rule mining: A systematic literature review, in: 2021 the 3rd International Conference on Big Data Engineering and Technology (BDET), 2021, pp. 40–49.
[16] M. Shahin, W. Inoubli, S. A. Shah, S. B. Yahia, D. Draheim, Distributed scalable association rule mining over covid-19 data, in: International Conference on Future Data and Security Engineering, Springer, 2021, pp. 39–52.
[17] M. Hahsler, S. Chelluboina, K. Hornik, C. Buchta, The arules r-package ecosystem: analyzing interesting patterns from large transaction data sets, The Journal of Machine Learning Research 12 (2011) 2021–2025.

Curriculum Vitae

1. Personal data

Name	Mohammad Reza Heidari Iman
Date of birth	16/09/1990
Place of birth	Mashhad, Iran
Nationality	Iranian

2. Contact information

E-mail	mohammadreza.heidari@taltech.ee mheydari.t@gmail.com

3. Education

2020–2024	PhD in Information and Communication Technology, Tallinn University of Technology, Tallinn, Estonia
2013–2015	MSc in Computer Engineering-Software Ferdowsi University of Mashhad Mashhad, Iran
2008–2013	BSc in Computer Engineering-Software Sadjad University of Technology Mashhad, Iran

4. Language competence

Persian	Native
English	Fluent
Italian	Beginner

5. Professional employment

2020–2024	Department of Computer Systems, Tallinn University of Technology, Early-Stage Researcher
2013–2016	Sadjad University of Technology, Lecturer
2010–2013	Sadjad University of Technology, & Eqbal Institute of Higher Education Teaching Assistant

6. Honours and awards

- 2013–2015, Ranked 3rd among Computer Software Engineering MSc students.

7. Supervision Experiences

- 2023–2024, MSc thesis, Improvement in the Apriori Algorithm to Enhance the Efficiency of Association Rule Mining Techniques, Cem Samiloglu.
- 2023–2024, BSc thesis, Behavior Analysis and Prediction of Vulnerabilities in IoT Devices using Data Mining and Machine Learning Techniques, Paula Pakina.
- 2023–2024, BSc thesis, Anomaly Detection in Autonomous Vehicles with the Aid of Association Rule Mining and Machine Learning, Mariia Svimonishvili.

8. Field of research

- Verification
- Assertion-Based Verification
- Formal Verification
- Security Verification
- Autonomous Vehicles
- Cyber-Physical Systems

Elulookirjeldus

1. Isikuandmed

Nimi	Mohammad Reza Heidari Iman
Sünniaeg	16/09/1990
Sünnikohtg	Mashhad, Iraan
Kodakondsus	Iraanlane

2. Kontaktandmed

E-post	mohammadreza.heidari@taltech.ee mheydari.t@gmail.com

3. Hariduskäik

2020–2024	PhD Infotehnoloogias ja kommunikatsioonitehnoloogias Tallinna Tehnikaülikool, Tallinn, Eesti
2013–2015	Magistrikraad Arvutitehnika-tarkvara erialal Ferdowsi Ülikool Mashhadis Mashhad, Iraan
2008–2013	Bakalaureusekraad Arvutitehnika-tarkvara erialal Sadjadi Tehnikaülikool Mashhad, Iraan

4. Keelteoskus

Pärsia keel	Emakeel
Inglise keel	Kõrgtase
Itaalia keel	Algtase

5. Professionaalne töökogemus

2020–2024	Arvutisüsteemide instituut, Tallinna Tehnikaülikool, nooremteadur
2013–2016	Sadjadi Tehnikaülikool, õppejõud
2010–2013	Sadjadi Tehnikaülikool, & Eqbal'i Kõrgkooli Instituut õpetajate assistent

6. Autasud ja tunnustused

- 2013–2015, Kolmas koht Arvutitehnika-tarkvara magistriõppe üliõpilaste seas.

7. Juhendamiskogemused

- 2023–2024, Magistritöö, Apriori algoritmi täiustamine assotsiatiivsete reeglikaevandamise tehnikate efektiivsuse suurendamiseks, Cem Samiloglu.
- 2023–2024, Bakalaureusetöö, IoT-seadmete haavatavuste analüüs ja ennustamine andmekaeve ja masinõppe meetodite abil, Paula Pakina.
- 2023–2024, Bakalaureusetöö, Anomaaliadetektsioon autonoomsetes sõidukites assotsiatiivsete reeglikaevandamise ja masinõppe abil, Mariia Svimonishvili.

8. Teadusvaldkond

- Verifitseerimine
- Asetuste-Põhine Verifitseerimine
- Formaalne Verifitseerimine
- Turbeverifitseerimine
- Autonoomsed Sõidukid
- Küberteabe Süsteemid

ISSN 2585-6901 (PDF)
ISBN 978-9916-80-177-2 (PDF)