Dynamic Property Ordering for Efficient Multi-Property Bounded Model Checking

Vivek Kumar

Dynamic Property Ordering for Efficient Multi-Property Bounded Model Checking

Dissertation submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
in
Computer Science

by

Vivek Kumar
[Roll No: CS2436]

under the guidance of

Ansuman Banerjee
Professor
Advanced Computing and Microelectronics Unit

Indian Statistical Institute
Kolkata-700108, India

June 2026

CERTIFICATE

This is to certify that the dissertation titled **"Dynamic Property Ordering for Efficient Multi-Property Bounded Model Checking"** submitted by **Vivek Kumar** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Ansuman Banerjee
Advanced Computing and Microelectronics Unit (ACMU)
Indian Statistical Institute, Kolkata

Acknowledgement

I would like to thank my supervisor, *Dr. Ansuman Banerjee*, Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata for his continuous guidance and unwavering support. For the entire duration of the thesis, I have had many opportunities to learn and improve myself and my work. His guidance has given me a much better appreciation of the research sphere and the value of good quality work.

My deepest thanks to the faculties of Indian Statistical Institute, for their support and assistance throughout the duration of the thesis.

I would also like to thank *Satyam Shubham* and *Saumya Jaipuria*, ACMU, Indian Statistical Institute, Kolkata for their continued mentorship.

Last but not the least, I would like to thank my family, friends and peers for their help and support. I thank all those, whom I have missed out from the above list.

Vivek Kumar
Roll No. CS2436
Indian Statistical Institute
Kolkata - 700108, India.

Abstract

Formal verification plays a critical role in ensuring the correctness of modern hardware designs. As the complexity of digital systems increases, designs are often associated with a large number of verification properties that must be analyzed within limited computational resources. In conventional multi-property bounded model checking (BMC), all properties are verified simultaneously. While this approach enables parallel analysis, difficult properties can consume a disproportionate amount of resources, causing simpler properties to be delayed and reducing the overall efficiency of bug detection.

This thesis presents dynamic property ordering techniques for efficient multi-property verification using SAT-based bounded model checking in the ABC verification framework. The central idea is to verify properties individually and dynamically prioritize them based on their observed verification progress, allowing computational resources to be directed toward properties that are more likely to yield results within a given time budget.

Two dynamic property ordering algorithms are proposed. The first algorithm, ALG1, employs a round-robin style strategy in which unsolved properties are periodically reordered according to the maximum verification depth (frame) reached, prioritizing properties that demonstrate greater progress. The second algorithm, ALG2, adopts a priority-based scheduling approach where each property's priority is determined by its verification rate, measured as frames explored per second. Properties with higher progress rates are allocated greater verification resources.

The proposed approaches are evaluated on benchmark suites from the Hardware Model Checking Competition (HWMCC) 2012 and 2013 and compared against two baselines: the conventional ABC multi-property verification method and an Equal Time Bounding (ETB) strategy that distributes the available verification time equally among all properties. Experimental results demonstrate that dynamic property ordering significantly improves verification efficiency. Both ALG1 and ALG2 solve more properties and achieve greater verification depth within the same time budget, while also accelerating bug discovery. Across the benchmark set, the proposed methods provide improvements exceeding 40% over the baseline approaches in key performance metrics.

The results demonstrate that dynamic property scheduling is an effective technique for improving the scalability and effectiveness of multi-property bounded model checking, offering a practical solution for faster bug detection and enhanced utilization of verification resources.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

The increasing complexity of modern hardware systems has made verification one of the most challenging stages of the design cycle. Contemporary digital designs often contain millions of gates and implement sophisticated functionalities, making manual validation and traditional testing techniques insufficient for ensuring correctness. Undetected design errors may lead to costly redesigns, delayed product releases, and failures in safety-critical applications. Consequently, verification has become a major component of modern hardware development flows.

Formal verification provides a mathematically rigorous approach for establishing design correctness [11]. Unlike simulation-based techniques, which examine only a subset of possible executions, formal verification systematically analyzes all behaviors of a design within a given model. This capability enables the detection of subtle bugs that may be missed during conventional testing.

Among various formal verification techniques, model checking has emerged as a widely adopted approach for verifying hardware systems [11]. Model checking determines whether a design satisfies a set of specified properties by exhaustively exploring the state space of the system. If a property is violated, a counterexample demonstrating the violation is automatically generated.

Bounded Model Checking (BMC) is a SAT-based model checking technique that has proven highly effective for bug detection [4]. BMC transforms the verification problem into a satisfiability problem by unrolling the transition relation of the design for a finite number of time frames and checking whether a property violation can occur within the specified bound. Advances in SAT solving technology have significantly improved the effectiveness of BMC, making it an important component of industrial verification frameworks [7].

Modern hardware designs are typically associated with a large number of verification properties. These properties capture different aspects of design correctness and must be analyzed during verification. To improve efficiency, verification tools often employ multi-property verification strategies where multiple properties are analyzed simultaneously. While such approaches allow sharing of verification resources, they also introduce challenges due to the varying complexities of individual properties.

1.2 Problem Statement

Traditional multi-property bounded model checking approaches verify all properties concurrently using a shared verification engine. Although this strategy reduces redundant computations and enables resource sharing, it may result in inefficient utilization of verification resources.

In practice, verification properties exhibit varying levels of difficulty. Some properties can be disproved or verified quickly, while others require significantly greater computational effort. When all properties compete for the same verification resources, difficult properties may consume a substantial portion of the available computational budget, thereby delaying the analysis of easier properties.

This resource contention can negatively impact bug-finding efficiency and reduce the total number of properties solved within a fixed verification timeout. Consequently, there is a need for techniques that can intelligently prioritize verification effort among properties based on their observed verification behavior.

The central hypothesis of this thesis is that dynamically ordering properties according to runtime verification statistics can improve the effectiveness of multi-property bounded model checking. By prioritizing properties that are more likely to yield results, verification resources can be utilized more efficiently, leading to faster bug discovery and improved verification coverage.

1.3 Research Objectives

The primary objective of this thesis is to improve the efficiency of multi-property bounded model checking through dynamic property ordering and resource allocation strategies.

The specific objectives of this work are as follows:

- Improve bug-finding efficiency in multi-property bounded model checking.
- Increase the number of properties solved within a fixed verification budget.
- Improve the verification depth achieved by individual properties.
- Develop dynamic scheduling techniques that utilize verification progress information.
- Evaluate the effectiveness of the proposed techniques on standard hardware verification benchmarks.

1.4 Contributions

This thesis proposes dynamic property ordering techniques for improving the efficiency of multi-property bounded model checking. The main contributions of this work are summarized below.

- *Dynamic Property Ordering Framework*: A framework is proposed for dynamically scheduling verification properties during bounded model checking.
- *ALG1*: A round-robin inspired property ordering algorithm is developed. In each iteration, unsolved properties are reordered according to the maximum verification depth reached, prioritizing properties that demonstrate greater progress.

- *ALG2*: A priority-based scheduling algorithm is developed. Property priorities are determined using the verification rate measured as frames reached per unit execution time.
- *Dynamic Resource Allocation*: The proposed algorithms dynamically allocate verification effort among properties based on observed runtime behavior.
- *Implementation*: The proposed techniques are implemented within the ABC verification framework for SAT-based bounded model checking.
- *Experimental Evaluation*: The proposed approaches are evaluated using benchmark suites from the Hardware Model Checking Competition (HWMCC) 2012 and HWMCC 2013.
- *Performance Improvement*: Experimental evaluation is conducted to assess the effectiveness of the proposed approaches relative to conventional multi-property verification and Equal Time Bounding strategies.

1.5 Organization of the Thesis

The remainder of this thesis is organized as follows.

Chapter 2: Presents the background and related work on formal verification, model checking, bounded model checking, SAT solving, and multi-property verification.

Chapter 3: Describes the proposed dynamic property ordering framework, including the design and implementation of ALG1 and ALG2.

Chapter 4: Presents the experimental methodology, benchmark selection procedure, implementation details, and evaluation metrics.

Chapter 5: Discusses the experimental results and compares the proposed techniques against conventional multi-property verification and Equal Time Bounding approaches.

Chapter 6: Summarizes the contributions of this thesis, discusses limitations, and outlines directions for future work.

Chapter 2

Background and Related Work

In this chapter, we describe the background concepts and prerequisites related to the systems that are discussed and referenced throughout the thesis. We also demonstrate the working of the systems discussed to have a better understanding of the work described in the subsequent chapters. We finally describe the software tools and programming libraries used in the coming chapters.

2.1 Formal Verification

The increasing complexity of modern hardware systems has made verification one of the most critical and resource-intensive phases of the design cycle. Errors discovered after fabrication can result in substantial financial losses, delayed product deployment, and in some cases catastrophic failures of safety-critical systems. Consequently, ensuring correctness before deployment has become a fundamental requirement in modern hardware design flows [10].

Traditionally, hardware verification has relied heavily on simulation and testing methodologies. In simulation-based verification, a collection of test vectors is applied to the design and the resulting outputs are compared against expected behavior. Although simulation remains the dominant verification technique in industrial practice due to its flexibility and ease of deployment, it suffers from a fundamental limitation: only a finite subset of the possible input space and execution scenarios can be explored. As a result, the absence of observed errors during simulation does not imply the absence of design bugs [10].

Formal verification addresses this limitation by employing mathematically rigorous techniques to determine whether a design satisfies a given specification. Instead of examining individual execution traces, formal verification reasons about all possible behaviors of the system within a mathematical model. This exhaustive analysis enables formal methods to provide stronger correctness guarantees than conventional simulation-based approaches.

A formal verification problem typically consists of three components: a mathematical model of the system under verification, a specification describing the desired behavior of the system, and a verification engine that determines whether the model satisfies the specification. If the specification is violated, the verification procedure produces a counterexample demonstrating a sequence of states that leads to the violation. Such counterexamples are extremely valuable during debugging because they provide concrete evidence of the underlying design error [10].

Formal verification techniques can be broadly classified into two categories: theorem proving and

model checking. Theorem proving relies on logical deduction to establish system correctness and often requires significant user guidance. While theorem proving is highly expressive and capable of handling complex systems, the verification process may involve substantial manual effort. In contrast, model checking is largely automated and systematically explores the state space of a system to determine whether specified properties hold. Due to its automation and effectiveness, model checking has become one of the most widely adopted formal verification techniques for hardware systems [10].

The properties verified using formal methods are commonly expressed in temporal logics and capture correctness requirements of the design. These requirements are generally classified as safety properties and liveness properties. Safety properties specify that undesirable events never occur during system execution. Examples include the absence of deadlocks, mutual exclusion violations, or illegal state transitions. Liveness properties, on the other hand, specify that desirable events eventually occur. For example, a request issued by a component should eventually receive a corresponding response [10].

The effectiveness of formal verification has led to its widespread adoption in industrial verification flows, particularly for detecting corner-case bugs that are difficult to expose through simulation. However, exhaustive state-space exploration often encounters scalability challenges due to the state explosion problem, where the number of reachable states grows exponentially with the number of system variables. Over the years, numerous techniques have been developed to address this challenge, including symbolic representations, abstraction methods, satisfiability-based verification, and bounded analysis techniques [10].

Among these approaches, satisfiability-based verification techniques have achieved remarkable success due to advances in modern SAT solvers. In particular, Bounded Model Checking (BMC) has emerged as a highly effective bug-finding methodology for hardware verification. BMC combines symbolic system modeling with efficient SAT solving to discover property violations within a bounded execution depth. The concepts underlying model checking and bounded model checking are discussed in the subsequent sections.

2.2 Model Checking

Model checking is an automated formal verification technique used to determine whether a system satisfies a given specification. Introduced in the early 1980s, model checking has become one of the most successful approaches for hardware and protocol verification due to its ability to automatically analyze complex systems and generate counterexamples when violations are detected [10].

The fundamental idea behind model checking is to represent the behavior of a system as a mathematical model and systematically explore all reachable states to determine whether specified correctness properties hold. If the property is satisfied, the model checker reports success. Otherwise, it generates a counterexample demonstrating a sequence of state transitions that leads to a violation of the property. Such counterexamples significantly simplify the debugging process by providing concrete evidence of erroneous behavior [10].

2.2.1 Transition Systems

Model checking typically represents a system using a finite-state transition system. A transition system can be formally defined as a tuple

$$M = (S, I, T)$$

where S denotes the set of system states, $I \subseteq S$ represents the set of initial states, and $T \subseteq S \times S$ denotes the transition relation describing how the system evolves from one state to another.

Starting from the initial states, repeated application of the transition relation generates all reachable states of the system. The objective of model checking is to determine whether every reachable execution of the transition system satisfies a given specification.

2.2.2 Specification of Properties

The correctness requirements of a system are commonly expressed using temporal logics, which allow properties to be specified over sequences of states. Two widely used temporal logics are Linear Temporal Logic (LTL) and Computational Tree Logic (CTL) [10].

Temporal logic extends classical propositional logic by introducing operators that describe how properties evolve over time. Examples of temporal operators include:

- X (Next): a property holds in the next state.
- F (Eventually): a property holds at some future state.
- G (Globally): a property holds in all future states.
- U (Until): one property remains true until another becomes true.

Using these operators, designers can express both safety and liveness requirements in a mathematically precise manner.

2.2.3 Explicit-State Model Checking

The earliest model checking algorithms relied on explicit-state exploration, where each reachable state of the system is generated and examined individually. Starting from the initial state, the model checker traverses the state space and verifies whether the specified properties hold for every reachable state [10].

Although explicit-state techniques are conceptually simple and effective for small systems, they quickly become impractical as system complexity increases. The number of reachable states often grows exponentially with the number of state variables, resulting in excessive memory and computational requirements.

2.2.4 Symbolic Model Checking

To address the limitations of explicit-state exploration, symbolic model checking was introduced. Instead of representing states individually, symbolic methods represent sets of states and transitions using compact mathematical structures such as Binary Decision Diagrams (BDDs) [9].

By manipulating symbolic representations rather than individual states, symbolic model checking can analyze significantly larger systems than explicit-state techniques. During the 1990s, BDD-based symbolic model checking became one of the dominant approaches for hardware verification and enabled the verification of industrial-scale designs [9].

Despite their success, symbolic techniques based on BDDs suffer from scalability limitations. For many verification problems, the size of the BDD representation can grow exponentially, leading to excessive memory consumption. This challenge motivated the development of satisfiability-based verification methods that leverage advances in SAT solving technology.

2.2.5 State Explosion Problem

A major challenge faced by all model checking techniques is the state explosion problem. As the number of state variables increases, the number of possible system states grows exponentially. Consequently, exhaustive exploration of the state space becomes increasingly difficult for large designs.

Numerous techniques have been proposed to mitigate state explosion, including abstraction, compositional verification, symbolic representations, partial-order reduction, and satisfiability-based verification [10]. Among these approaches, SAT-based techniques have demonstrated remarkable effectiveness for bug detection in hardware systems.

The limitations of traditional model checking techniques and the rapid progress of SAT solvers led to the development of Bounded Model Checking (BMC), a satisfiability-based verification approach that has become one of the most widely used bug-finding techniques in modern hardware verification. The principles of BMC are discussed in the next section.

2.3 SAT Solving Fundamentals

The Boolean Satisfiability Problem (SAT) is one of the most fundamental problems in computer science and forms the foundation of many modern verification techniques. Given a Boolean formula, the objective of SAT solving is to determine whether there exists an assignment of truth values to the variables such that the formula evaluates to true. If such an assignment exists, the formula is said to be satisfiable; otherwise, it is unsatisfiable [7].

SAT was the first problem proven to be NP-complete through Cook's theorem, making it a central problem in computational complexity theory [12]. Despite its theoretical difficulty, significant advances in SAT solving algorithms over the past two decades have enabled the efficient solution of extremely large practical instances arising from hardware verification, software analysis, planning, and optimization problems [7].

2.3.1 Boolean Formulas and CNF Representation

A Boolean formula consists of variables that can assume either the value true or false and are connected through logical operators such as conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$). Modern SAT solvers typically require formulas to be represented in Conjunctive Normal Form (CNF).

A formula is in CNF if it is expressed as a conjunction of clauses, where each clause is a disjunction of literals. A literal is either a Boolean variable or its negation. For example,

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_4) \wedge (x_2 \vee \neg x_4)$$

is a CNF formula consisting of three clauses.

Most verification problems are therefore transformed into equivalent CNF representations before being provided to a SAT solver. Efficient transformations such as Tseitin encoding are commonly employed to avoid excessive growth in formula size during this conversion process [18].

2.3.2 SAT Solving Algorithms

The earliest successful SAT solvers were based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [14]. DPLL performs a systematic search over possible variable assignments while employing several pruning techniques to reduce the search space.

The key components of the DPLL procedure include:

- **Decision Assignment**: Assign a truth value to an unassigned variable.
- **Unit Propagation**: Infer mandatory assignments implied by unit clauses.
- **Conflict Detection**: Identify assignments that violate one or more clauses.
- **Backtracking**: Reverse previous decisions and explore alternative assignments.

Modern SAT solvers extend DPLL through Conflict-Driven Clause Learning (CDCL), which has become the dominant SAT solving paradigm [15]. When a conflict is encountered, CDCL analyzes the cause of the conflict and generates a learned clause that prevents the same conflict from recurring in the future. This significantly reduces redundant exploration and improves solver performance.

Additional techniques such as non-chronological backtracking, variable activity heuristics, restart strategies, and clause database management have further enhanced the scalability of CDCL solvers, enabling them to handle industrial-scale verification problems involving millions of variables and clauses [7].

2.3.3 SAT Solving in Hardware Verification

The success of modern SAT solvers has made satisfiability-based methods one of the most effective approaches for hardware verification. Hardware circuits can be represented as Boolean formulas, allowing verification problems to be translated into SAT instances.

Many verification tasks can be expressed as satisfiability queries. For example, checking whether a Boolean property violation exists can be reduced to determining whether a corresponding Boolean formula is satisfiable. If the SAT solver finds a satisfying assignment, the assignment often directly corresponds to a counterexample demonstrating the violation. Conversely, if the formula is unsatisfiable, the Boolean property holds within the analyzed verification scope.

Compared to earlier Binary Decision Diagram (BDD)-based approaches, SAT-based methods often provide superior scalability for large industrial designs. This success has led to the development of numerous SAT-based verification techniques, including equivalence checking, combinational verification, sequential verification, and bounded model checking [5].

2.3.4 Motivation for SAT-Based Bounded Model Checking

The introduction of SAT-based verification techniques significantly improved the scalability of formal verification. A major advantage of SAT-based approaches is their ability to directly leverage advances in Boolean satisfiability solving technology. As SAT solvers become more efficient, the performance of SAT-based verification methods correspondingly improves without requiring fundamental changes to the verification framework itself.

One of the most influential developments in this area was the introduction of Bounded Model Checking (BMC) by Biere et al. [5]. BMC transforms the verification problem into a Boolean satisfiability problem by unrolling the transition relation of a design for a finite number of time frames and checking whether a property violation can occur within the specified bound. If the resulting SAT instance is satisfiable, the satisfying assignment corresponds to a counterexample demonstrating a violation of the property.

The rapid advancement of Conflict-Driven Clause Learning (CDCL) SAT solvers further improved the effectiveness of SAT-based verification methods [7]. Modern SAT solvers incorporate sophisticated techniques such as clause learning, non-chronological backtracking, efficient branching heuristics, and incremental solving, enabling the analysis of increasingly complex verification problems.

By combining the exhaustive reasoning capabilities of model checking with the scalability of modern SAT solving technology, BMC has become one of the most widely adopted bug-finding methodologies in hardware verification. Consequently, SAT-based bounded model checking forms the foundation of many contemporary verification frameworks, including the ABC verification system considered in this thesis.

Despite these advances, verification performance in multi-property settings remains highly dependent on how computational resources are distributed among properties during verification. The formulation and operation of bounded model checking are discussed in detail in the next section.

2.4 Bounded Model Checking

Bounded Model Checking (BMC) is a satisfiability-based verification technique introduced by Biere et al. [5]. Unlike traditional model checking approaches that attempt to explore the complete reachable state space of a system, BMC searches for counterexamples of bounded length. This bounded exploration allows BMC to leverage the power of modern SAT solvers and makes it particularly effective for bug detection in large hardware designs.

The key idea behind BMC is to determine whether a property violation can occur within a fixed number of execution steps. In general, a transition system may exhibit an infinite number of possible executions, and individual executions may themselves be of infinite length due to the presence of loops in the state-transition graph. Instead of exploring this potentially infinite behavior space, the verification problem is reduced to determining whether there exists a finite execution trace that violates the property within a specified bound. If such a trace exists, a SAT solver can produce a satisfying assignment corresponding to a counterexample. Otherwise, the absence of a counterexample is guaranteed only up to the chosen bound.

2.4.1 Transition System Representation

A hardware design can be modeled as a finite-state transition system

$$M = (S, I, T)$$

where:

- S denotes the set of states.
- $I(S)$ denotes the initial state predicate.
- $T(S, S')$ denotes the transition relation.

The transition relation describes how the system evolves from one state to another during execution. Starting from an initial state, repeated applications of the transition relation generate execution traces of the system.

2.4.2 Bounded Unrolling

The central idea of BMC is to unroll the transition relation for a finite number of steps. Suppose the verification depth is bounded by k. The transition relation is replicated k times to construct all possible executions of length k.

Figure 2.1 illustrates the unrolling process.

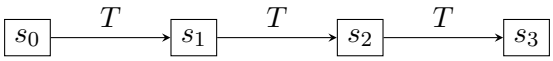

Figure 2.1: Unrolling of the transition relation in BMC.

Each state variable is replicated at every time frame, resulting in a finite propositional encoding of the behavior of the system up to depth k.

2.4.3 SAT Encoding of BMC

The verification problem is translated into a Boolean satisfiability problem. Given a safety property P, BMC searches for an execution trace of length at most k that violates the property.

The resulting SAT formula is given by

$$I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge \neg P_j(s_k) \tag{2.1}$$

where $I(s_0)$ denotes the initial-state constraint, $T(s_i, s_{i+1})$ represents the transition relation between consecutive time frames, and P_j denotes the property under verification.

The formula is satisfiable if and only if there exists a counterexample of length k that violates the property.

A visual representation of the encoding is shown below.

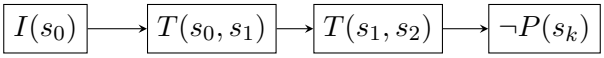

Figure 2.2: SAT encoding generated by BMC where a property violation occurs at depth 2.

2.4.4 Counterexample Generation

If the generated SAT formula is satisfiable, the satisfying assignment corresponds to an execution trace that violates the property. This trace is called a counterexample and can be used to diagnose and debug design errors.

One of the primary strengths of BMC is its ability to generate short counterexamples quickly. This characteristic has made BMC one of the most widely adopted bug-finding techniques in industrial verification flows.

2.4.5 Advantages and Limitations

Bounded Model Checking offers several advantages:

- Efficient utilization of modern SAT solvers.
- Ability to find deep bugs quickly.
- Automatic generation of counterexamples.
- Good scalability for large hardware designs.

Despite its success, BMC also has certain limitations. Since the analysis is bounded, the absence of a counterexample only guarantees correctness up to the explored depth. To establish complete correctness, additional proof techniques such as induction or Property Directed Reachability (PDR) are often required.

2.4.6 BMC in Multi-Property Verification

Industrial hardware designs are typically associated with hundreds or thousands of properties. Applying BMC independently to each property can result in substantial computational overhead. Consequently, modern verification frameworks often employ multi-property verification strategies that attempt to verify multiple properties simultaneously.

While multi-property verification improves resource sharing and reduces redundant computations, it introduces challenges related to property scheduling and resource allocation. Properties often differ significantly in difficulty, causing easy properties to compete with harder properties for verification resources.

These challenges motivate the need for intelligent property ordering and scheduling techniques, which form the primary focus of this thesis. The next section discusses multi-property verification and existing approaches for handling large collections of properties.

2.5 Multi-Property Verification

Modern hardware systems are typically verified against a large collection of correctness properties. These properties capture different aspects of system behavior, including protocol compliance, control logic correctness, data integrity, and safety requirements. Industrial verification environments often contain hundreds or even thousands of properties associated with a single design. Consequently, efficient verification of multiple properties has become an important challenge in formal verification [17].

2.5.1 Single-Property Verification

The simplest approach to formal verification is to verify each property independently. In this approach, a separate verification instance is created for every property. For a set of properties

$$P = \{P_1, P_2, \ldots, P_n\},$$

the verification engine performs n independent verification runs.

Figure 2.3 illustrates the single-property verification approach.

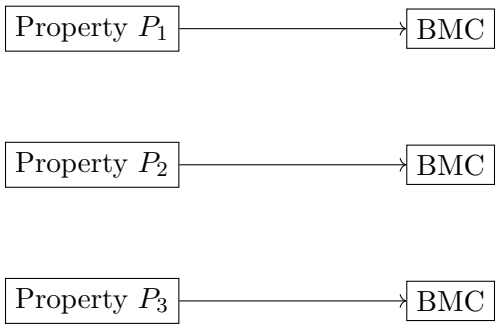

Figure 2.3: Single-property verification. Each property is verified independently.

Although this approach is conceptually simple, it results in substantial computational redundancy. Many properties are verified over the same design model and therefore repeatedly explore similar portions of the state space. As the number of properties increases, the overall verification cost grows significantly.

2.5.2 Multi-Property Verification

To reduce redundant computation, modern verification frameworks often employ multi-property verification. Instead of verifying each property separately, multiple properties are analyzed simultaneously within a single verification run [17].

Figure 2.4 illustrates the multi-property verification paradigm.

The primary motivation behind multi-property verification is the ability to reuse computations across properties. Since all properties are analyzed on the same design, information generated during verification can often be shared among multiple properties. This sharing reduces memory consumption and verification runtime compared to verifying properties independently.

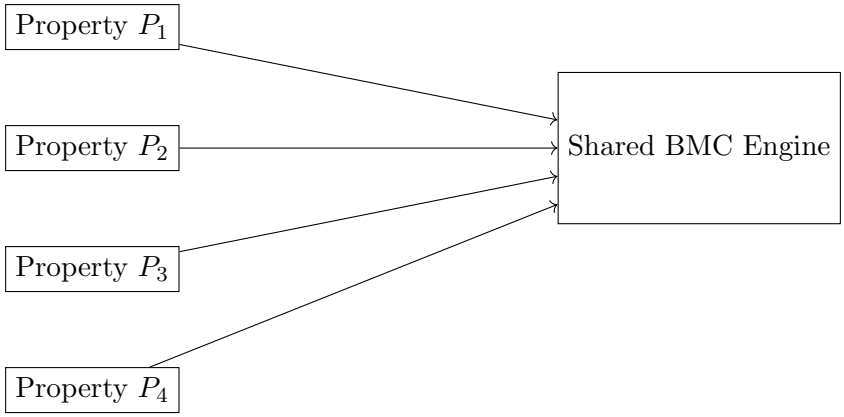

Figure 2.4: Multi-property verification using a shared verification engine.

2.5.3 Advantages of Multi-Property Verification

Multi-property verification offers several advantages over independent verification:

- Reduced duplication of verification effort.
- Improved utilization of verification resources.
- Lower memory requirements.
- Better scalability for designs with large numbers of properties.
- Reduced overall verification runtime.

These advantages have led to widespread adoption of multi-property verification techniques in industrial verification frameworks, including the ABC verification system.

2.5.4 Challenges in Multi-Property Verification

Despite its advantages, multi-property verification introduces several challenges. The most significant challenge arises from the varying difficulty levels of individual properties.

In practice, some properties may be disproved quickly through shallow counterexamples, while others may require deep exploration of the state space before they can be verified or disproved. Consequently, different properties compete for the same computational resources during verification.

Figure 2.5 illustrates this phenomenon.

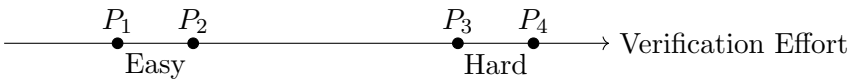

Figure 2.5: Properties may exhibit significantly different verification difficulties.

When all properties are processed simultaneously, difficult properties may consume a disproportionate amount of computational effort. As a result, easier properties may be delayed despite requiring relatively little verification effort.

This resource contention can negatively affect bug-finding efficiency and reduce the number of properties solved within a fixed verification timeout.

2.5.5 Property Scheduling

The above observations suggest that verification performance depends not only on the verification engine itself but also on how verification resources are distributed among properties. Property scheduling refers to the strategy used to determine the allocation of computational effort among multiple properties during verification.

An ideal scheduling strategy should prioritize properties that are likely to produce useful verification results while minimizing resource wastage on difficult properties. However, determining the relative difficulty of properties before verification is generally challenging.

Consequently, many verification frameworks employ static scheduling strategies that allocate resources uniformly across properties. Although simple to implement, such approaches often fail to utilize verification resources efficiently.

2.5.6 Motivation for Dynamic Property Ordering

The varying complexity of verification properties motivates the use of dynamic scheduling techniques. Instead of treating all properties equally, dynamic approaches use runtime verification information to guide resource allocation decisions.

Verification progress metrics such as explored depth, counterexample discovery, and verification rate can provide valuable information regarding the relative difficulty of properties. By exploiting this information, verification resources can be directed toward properties that are more likely to yield useful results within a given time budget.

This observation forms the central motivation of this thesis. The proposed work investigates dynamic property ordering techniques that prioritize properties based on observed verification progress with the objective of improving bug-finding efficiency and increasing the number of solved properties in multi-property bounded model checking.

2.6 ABC Framework

ABC is an academic verification and logic synthesis framework developed at the University of California, Berkeley. It provides a comprehensive collection of algorithms for logic synthesis, technology mapping, equivalence checking, model checking, and formal verification of sequential circuits [8, 16]. Due to its scalability and support for industrial verification flows, ABC has become one of the most widely used open-source verification platforms in both academia and industry.

ABC supports multiple verification methodologies, including bounded model checking, induction-based verification, property directed reachability, and sequential equivalence checking. The framework is designed around efficient circuit representations and highly optimized verification engines, enabling the analysis of large hardware designs.

2.6.1 And-Inverter Graph Representation

The primary internal representation used by ABC is the And-Inverter Graph (AIG). An AIG is a directed acyclic graph consisting of two-input AND gates and complemented edges representing logical negation [16].

The use of AIGs provides several advantages:

- Compact representation of Boolean functions.
- Efficient structural hashing.
- Scalable logic optimization.
- Efficient SAT-based verification.

By representing circuits as AIGs, ABC can apply numerous synthesis and optimization transformations before verification. These transformations often reduce circuit complexity and improve verification performance.

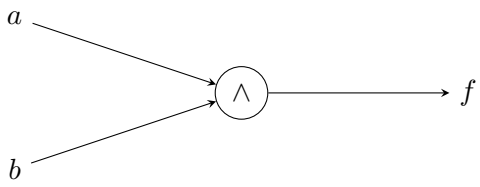

Figure 2.6: Simple And-Inverter Graph structure.

2.6.2 Verification Flow in ABC

The typical verification flow in ABC begins with the construction of an AIG representation from the input hardware description. The resulting AIG is then optimized using a collection of synthesis and reduction techniques before being passed to a verification engine.

Figure 2.7 illustrates the overall verification flow.

2.6.3 Bounded Model Checking in ABC

ABC provides several SAT-based verification engines, among which bounded model checking is one of the most widely used for bug detection. As mentioned earlier, BMC translates the existence of a counterexample within a bounded depth into a satisfiability problem and uses a SAT solver to determine whether such a counterexample exists [5].

For a given bound k, ABC incrementally unrolls the transition relation of the design and generates a SAT instance corresponding to the bounded model checking formulation discussed in Section 2.4. If the SAT instance is satisfiable, a counterexample trace is produced. Otherwise, the verification bound is increased and the process continues.

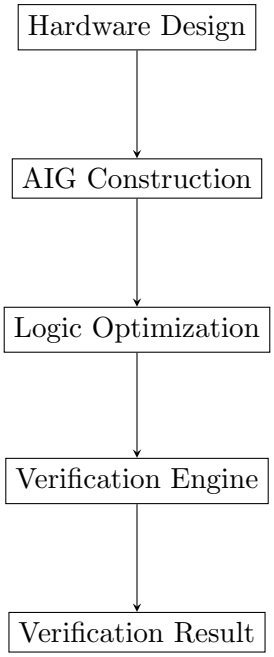

Figure 2.7: Simplified verification flow in ABC.

2.6.4 The `bmc3` Engine

The experiments presented in this thesis are based on ABC's `bmc3` verification engine. The `bmc3` engine performs incremental bounded model checking by progressively extending the verification depth while reusing information obtained from previous SAT solving iterations.

Incremental SAT solving significantly improves performance because clauses generated during earlier verification depths can be reused in subsequent iterations. This avoids rebuilding the SAT problem from scratch at every bound and reduces overall verification time.

The verification process employed by `bmc3` is illustrated in Figure 2.8.

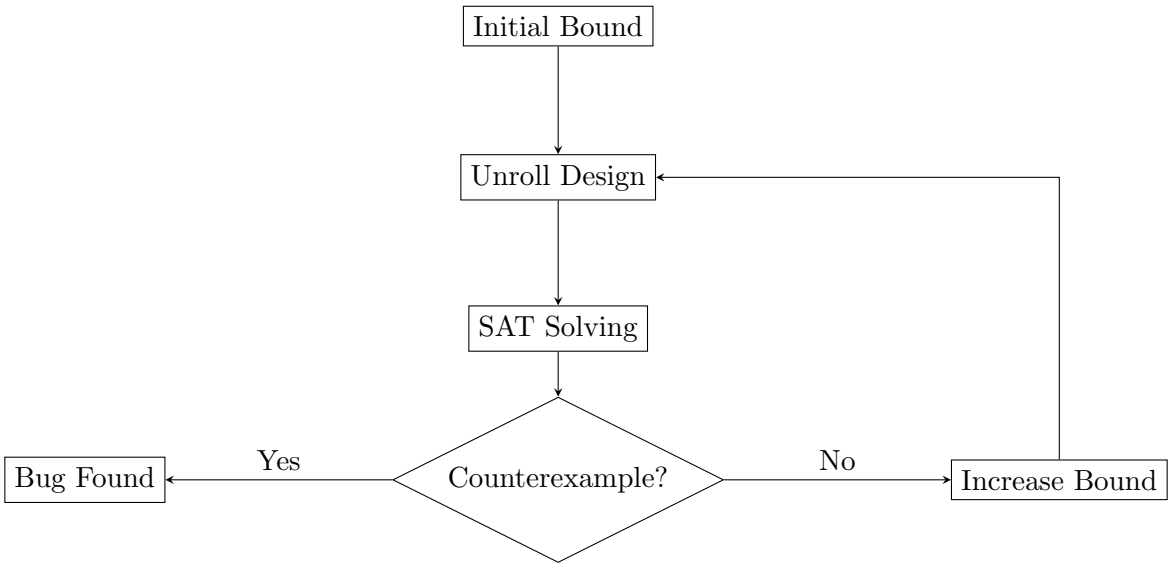

Figure 2.8: Incremental bounded model checking in `bmc3`.

2.6.5 Glucose SAT Solver

The `bmc3` engine relies on modern Conflict-Driven Clause Learning (CDCL) SAT solvers for solving the generated SAT instances. In this work, the Glucose SAT solver is employed as the underlying satisfiability engine.

Glucose extends the CDCL framework by introducing efficient clause learning and clause management strategies based on Literal Block Distance (LBD) metrics [3]. These techniques enable Glucose to identify highly relevant learned clauses and significantly improve solving performance on industrial SAT instances.

The combination of incremental bounded model checking and the Glucose SAT solver provides a scalable platform for analyzing large multi-property verification problems.

2.6.6 Multi-Property Verification Support

ABC supports native multi-property verification in which multiple properties are analyzed simultaneously within a single verification run. Instead of constructing separate verification instances for each property, the verification engine maintains a common circuit representation and shares verification resources across properties.

This approach reduces redundant computation and improves overall verification efficiency. However, as discussed in Section 2.5, properties often exhibit widely varying levels of verification difficulty. Consequently, some properties may consume a disproportionate amount of verification effort, leading to inefficient utilization of available resources.

2.6.7 Relevance to this Thesis

The work presented in this thesis extends the native multi-property verification capabilities of ABC. Specifically, modifications are introduced within the `bmc3` framework to dynamically schedule and prioritize properties during verification.

The proposed algorithms utilize runtime verification information to guide resource allocation decisions with the objective of improving bug-finding efficiency, increasing the number of solved properties, and achieving greater verification depth within a fixed verification budget. The design and implementation of these techniques are described in Chapter 3.

2.7 Related Work

The increasing complexity of modern hardware systems has motivated significant research in improving the scalability of formal verification techniques. In particular, multi-property verification has received considerable attention due to the large number of properties typically associated with industrial hardware designs. Researchers have proposed various approaches to improve verification efficiency through property decomposition, resource sharing, property ordering, and scheduling techniques.

2.7.1 Multi-Property Verification

Traditional verification approaches analyze each property independently, resulting in substantial duplication of verification effort. To address this limitation, multi-property verification techniques were introduced to enable the simultaneous verification of multiple properties within a shared verification framework [17].

Roorda and Claessen demonstrated that significant verification savings can be achieved by exploiting relationships among properties and sharing verification resources across multiple verification tasks [17]. Their work established the foundation for subsequent research on efficient multi-property verification frameworks.

Modern verification tools, including ABC, provide native support for multi-property verification, allowing multiple properties to be analyzed within a common verification environment. While resource sharing improves scalability, it also introduces challenges related to the efficient allocation of verification effort among properties of varying complexity.

2.7.2 Property Ordering and Scheduling

A growing body of research has investigated the use of property ordering and scheduling techniques to improve verification efficiency. The central observation behind these approaches is that verification properties often exhibit significantly different levels of difficulty. Consequently, treating all properties equally may lead to inefficient utilization of computational resources.

Recent work by Das et al. introduced PURSE (Property Ordering Using Runtime Statistics for Efficient Multi-Property Verification), a dynamic property ordering framework that utilizes runtime verification statistics to guide the verification process [13]. The authors observed that verification performance can be improved by intelligently prioritizing properties based on their observed runtime behavior rather than relying on static scheduling strategies.

PURSE dynamically reorders properties during verification and demonstrates that runtime statistics can provide useful indicators of property difficulty. Experimental results reported by the authors show that adaptive ordering strategies can significantly improve multi-property verification performance compared to conventional approaches. These findings highlight the importance of dynamic scheduling in verification frameworks and provide strong evidence that verification efficiency can be improved through informed property prioritization [13].

2.7.3 Verification Resource Allocation

Resource allocation has also emerged as an important research direction in formal verification. Various approaches have been proposed to determine how verification time and computational resources should be distributed among competing verification tasks.

Static allocation techniques distribute resources uniformly across properties. While simple to implement, such methods do not account for differences in property complexity and may result in inefficient utilization of available verification budgets.

Dynamic allocation approaches attempt to adapt resource distribution based on observed verification progress. These methods continuously monitor verification behavior and allocate additional resources to promising verification tasks. Such adaptive techniques have demonstrated improved scalability in

a variety of verification settings.

2.7.4 Discussion

The existing literature demonstrates that property ordering and resource allocation play an important role in determining the efficiency of multi-property verification. Previous research has shown that verification properties exhibit diverse runtime characteristics and that dynamic scheduling can significantly improve verification performance.

Among the existing approaches, PURSE represents one of the closest works to the research presented in this thesis. Both approaches exploit runtime verification information to guide verification decisions. However, while PURSE focuses on runtime-statistics-based property ordering in a general multi-property verification setting, the present work investigates dynamic property ordering strategies specifically within the context of SAT-based bounded model checking using the ABC `bmc3` framework.

Furthermore, this thesis introduces two alternative dynamic scheduling strategies, ALG1 and ALG2, that prioritize properties using verification-depth and verification-rate information respectively. The effectiveness of these approaches is evaluated using HWMCC benchmark suites and compared against conventional multi-property verification and equal-time allocation strategies.

2.8 Research Gap and Motivation

The previous sections presented the background and related work relevant to multi-property bounded model checking. Existing verification frameworks have significantly improved the scalability of formal verification through the use of SAT-based verification, efficient circuit representations, and shared verification infrastructures. In particular, multi-property verification has emerged as an effective technique for reducing redundant computation and improving resource utilization when verifying large collections of properties.

Despite these advances, several challenges remain. The primary challenge arises from the heterogeneous nature of verification properties. In practical verification environments, different properties often exhibit substantially different verification characteristics. Some properties can be disproved quickly through shallow counterexamples, while others require significantly deeper exploration of the state space before useful verification results can be obtained.

Traditional multi-property verification approaches typically treat all properties uniformly during verification. Although such approaches benefit from resource sharing, they do not explicitly account for differences in property difficulty. Consequently, difficult properties may consume a disproportionate amount of verification effort, reducing the overall effectiveness of the verification process.

Recent work such as PURSE has demonstrated that runtime verification statistics can be used to improve property ordering and verification efficiency [13]. The success of such approaches indicates that dynamic scheduling strategies can provide significant benefits over static verification methodologies.

However, several opportunities remain for further investigation. In particular, the effectiveness of verification-progress-based scheduling strategies within SAT-based bounded model checking has not been extensively explored. Existing approaches primarily focus on general runtime statistics, while the relationship between verification depth, verification rate, and property prioritization remains an open area of investigation.

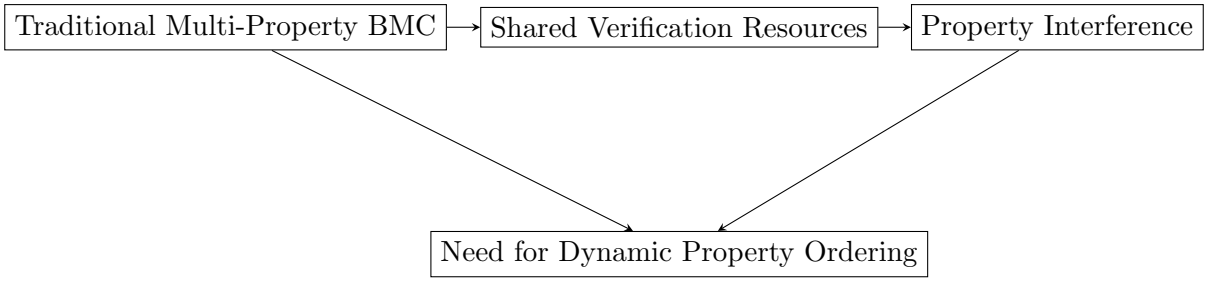

Figure 2.9: Research gap motivating dynamic property ordering in multi-property bounded model checking.

The observations above motivate the central research question addressed in this thesis:

> *Can runtime verification progress information be utilized to dynamically prioritize properties and improve the efficiency of multi-property bounded model checking?*

To address this question, this thesis investigates dynamic property ordering strategies that exploit verification progress information collected during bounded model checking. Rather than allocating verification effort uniformly across properties, the proposed approaches continuously adjust property priorities based on observed runtime behavior.

Two dynamic scheduling algorithms are proposed. The first approach, ALG1, utilizes the maximum verification depth reached by each property as an indicator of verification progress and periodically reorders properties during verification. The second approach, ALG2, employs a priority-based scheduling mechanism in which property priorities are determined by verification rate, measured in terms of frames explored per unit execution time.

Both approaches are implemented within the ABC `bmc3` framework and evaluated using benchmark suites from HWMCC 2012 and HWMCC 2013. The objective is to determine whether dynamic property ordering can improve bug-finding efficiency, increase the number of solved properties, and achieve greater verification depth under fixed verification budgets.

The proposed framework and algorithms are presented in detail in the next chapter.

Chapter 3

Proposed Framework

3.1 Overview

The previous chapter discussed the background of formal verification, bounded model checking, multi-property verification, and existing approaches for improving verification efficiency. The literature review highlighted that while modern verification frameworks support efficient multi-property verification, the allocation of verification resources among properties remains a significant challenge. Existing approaches often treat properties uniformly despite substantial differences in their verification characteristics. As a result, difficult properties may consume a disproportionate amount of computational effort, limiting the overall effectiveness of the verification process.

This chapter presents the proposed framework for dynamic property ordering in multi-property bounded model checking. The central idea is to utilize runtime verification information to guide the allocation of verification resources among competing properties. Instead of processing properties using a fixed ordering or allocating equal verification effort to all properties, the proposed framework continuously monitors verification progress and dynamically adjusts property priorities throughout the verification process.

The proposed approach is motivated by the observation that verification properties exhibit significantly different levels of difficulty. During bounded model checking, some properties demonstrate rapid progress by reaching deeper verification depths within a short period of time, while others progress more slowly. Such runtime information can provide valuable insight into the relative difficulty of properties and can be exploited to improve verification efficiency.

To investigate this hypothesis, two dynamic property ordering algorithms are proposed. The first algorithm, referred to as ALG1, prioritizes properties according to the maximum verification depth reached during previous verification iterations. The second algorithm, referred to as ALG2, employs a priority-based scheduling strategy that utilizes the verification rate, measured in terms of frames explored per unit execution time. Both approaches aim to direct verification effort toward properties that exhibit promising verification progress while reducing the impact of resource contention among properties.

The proposed algorithms are implemented within the `bmc3` framework of the ABC verification system using the Glucose SAT solver. The implementation extends the native multi-property verification flow by introducing a dynamic scheduling layer responsible for collecting runtime statistics, computing

property priorities, and determining the order in which properties receive verification resources.

Figure 3.1 presents a high-level overview of the proposed framework.

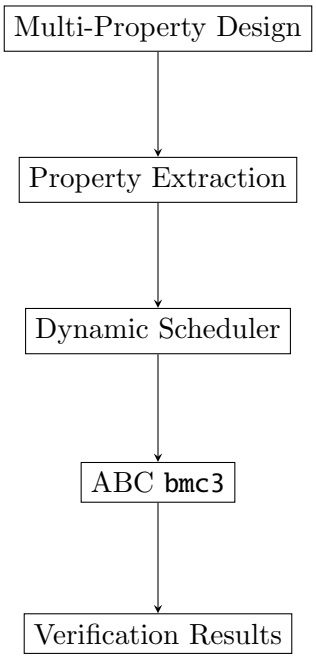

Figure 3.1: Overview of the proposed dynamic property ordering framework.

The framework operates by first extracting the properties associated with a design and initializing the verification process. During execution, verification statistics are collected for each property. These statistics are then utilized by the dynamic scheduler to determine the ordering of properties and allocate verification effort accordingly. The reordered property set is subsequently processed by the bounded model checking engine. This process is repeated throughout verification, enabling the framework to adapt its scheduling decisions based on observed runtime behavior.

The remainder of this chapter is organized as follows. Section 3.2 describes the baseline approaches used for comparison. Section 3.3 introduces the concept of dynamic property ordering and the verification progress metrics employed in this work. Sections 3.4 and 3.5 present the proposed algorithms in detail. Section 3.6 discusses the implementation of the framework within ABC, and Section 3.7 summarizes the chapter.

3.2 Baseline Approaches

To evaluate the effectiveness of the proposed dynamic property ordering algorithms, their performance is compared against two baseline verification approaches. The first baseline corresponds to the native multi-property bounded model checking framework provided by ABC, referred to as *Vanilla ABC*. The second baseline is an *Equal Time Bounding (ETB)* strategy that distributes the available verification budget uniformly among all properties.

These baselines represent two fundamentally different approaches to resource allocation. Vanilla ABC relies on the default scheduling mechanism of the verification engine, whereas ETB explicitly allocates equal verification effort to each property. Comparing the proposed algorithms against these baselines enables the impact of dynamic property ordering to be evaluated independently of the underlying bounded model checking engine.

3.2.1 Vanilla ABC

The first baseline considered in this work is the native multi-property bounded model checking implementation available in the ABC verification framework. In this approach, all properties are verified simultaneously within a single verification run using the `bmc3` engine and the Glucose SAT solver [8].

Let

$$P = \{P_1, P_2, \ldots, P_n\}$$

denote the set of properties associated with a design. Vanilla ABC processes the entire property set using a common verification infrastructure. The verification engine incrementally increases the verification depth while attempting to verify all properties concurrently.

Figure 3.2 illustrates the native multi-property verification flow.

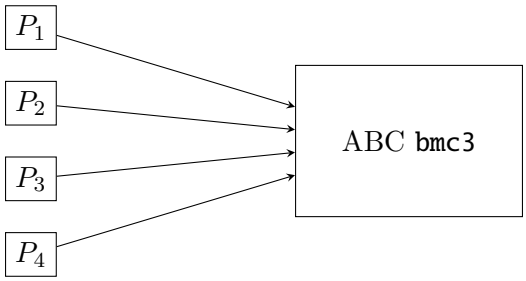

Figure 3.2: Native multi-property verification in ABC.

The primary advantage of this approach is the ability to share verification resources among properties, thereby reducing redundant computations. However, all properties compete for the same verification resources, and no explicit prioritization mechanism is employed.

As discussed in Chapter 2, verification properties often exhibit significantly different levels of difficulty. Consequently, difficult properties may consume a substantial portion of the available verification effort, potentially delaying the discovery of bugs associated with easier properties.

3.2.2 Equal Time Bounding (ETB)

The second baseline considered in this work is Equal Time Bounding (ETB). The objective of ETB is to ensure that every property receives an equal share of the available verification budget.

Let

$$T_{total}$$

denote the total verification time budget and let

$$n$$

represent the number of properties.

The verification time allocated to each property is given by

$$T_i = \frac{T_{total}}{n} \tag{3.1}$$

where T_i denotes the verification budget assigned to property P_i.

For example, if the total verification time budget is 3600 seconds and the design contains 40 properties, then each property receives

$$\frac{3600}{40} = 90$$

seconds of verification time.

Unlike Vanilla ABC, ETB verifies each property independently and allocates an equal fraction of the total verification budget to every property. If the total time budget is denoted by T_{total} and the design contains n properties, then each property receives a verification budget of T_{total}/n. Once the allocated budget is exhausted, verification of that property is terminated and the process proceeds to the next property.

Figure 3.3: Equal Time Bounding (ETB): each property receives an equal fraction of the total verification budget.

Although ETB guarantees fairness among properties, it does not utilize any information regarding verification progress or property difficulty. As a result, properties that demonstrate promising verification behavior receive the same amount of verification effort as properties that make little progress.

Consequently, ETB may underutilize available verification resources when significant differences exist among property complexities.

3.2.3 Discussion

The two baseline approaches represent contrasting resource allocation strategies. Vanilla ABC relies on the native scheduling behavior of the verification engine, while ETB enforces uniform resource distribution across properties.

Neither approach utilizes runtime verification information to guide resource allocation decisions. In particular, both methods ignore verification progress metrics such as explored depth and verification rate, which may provide useful indicators of property difficulty.

The proposed algorithms presented in the following sections address this limitation by dynamically adjusting property priorities according to observed verification behavior. By exploiting runtime statistics, the proposed framework seeks to improve bug-finding efficiency and increase the number of solved properties within a fixed verification budget.

3.3 Dynamic Property Ordering Concept

As discussed in Chapter 2, verification properties often exhibit significantly different levels of difficulty. During bounded model checking, some properties may quickly reach deep verification bounds or produce counterexamples, while others may require substantial computational effort before meaningful progress is achieved. Consequently, allocating verification resources uniformly across all properties may not always result in efficient utilization of the available verification budget.

The central idea behind the proposed framework is to dynamically allocate verification effort based on the observed verification progress of individual properties. Rather than relying on a fixed ordering throughout the verification process, the framework continuously monitors runtime statistics and adjusts property priorities during execution. Properties demonstrating stronger verification progress are prioritized, while solved properties are removed from subsequent scheduling decisions.

3.3.1 Active Property Set

Let

$$P = \{P_1, P_2, \ldots, P_n\}$$

denote the set of properties associated with a design.

At any point during verification, a subset of properties may already be solved. These include both:

- Properties that have been proven.
- Properties for which counterexamples have been discovered.

Such properties no longer require verification effort and are removed from future scheduling decisions.

The set of properties that still require verification is referred to as the active property set and is denoted by

$$A \subseteq P.$$

Initially,

$$A = P.$$

As verification progresses, solved properties are removed from A, causing the active property set to shrink over time.

3.3.2 Verification Intervals

The proposed framework operates iteratively. Verification is performed over a sequence of verification intervals. During each interval, bounded model checking is applied to the properties currently contained in the active property set.

At the end of each interval, runtime statistics are collected for every active property. These statistics are subsequently used to determine the ordering of properties for the next verification interval.

This iterative process enables the framework to adapt its scheduling decisions according to the observed behavior of the verification engine.

3.3.3 Verification Progress Metric

An important requirement of dynamic scheduling is the ability to estimate the verification progress achieved by individual properties.

In this work, verification progress is measured using the maximum bounded model checking depth reached during a property's allocated verification interval.

For a property P_i, the verification progress metric is defined as

$$FR(P_i) \tag{3.2}$$

where $FR(P_i)$ denotes the BMC frame reached by property P_i during the current verification interval.

A larger value of $FR(P_i)$ indicates that the verification engine was able to explore the state space more deeply for that property within the allocated verification effort.

Figure 3.4 illustrates the concept of verification progress.

Figure 3.4: Example of verification progress measured using maximum frames reached.

3.3.4 Runtime Statistics Collection

In addition to verification depth, the framework maintains runtime statistics for each active property throughout verification. These statistics are collected after every verification interval and are subsequently used by the scheduling algorithms.

The collected information includes:

- Maximum frames reached.
- Verification runtime.
- Verification status (unsolved, proved, or falsified).

The runtime statistics provide valuable information regarding the relative difficulty and verification behavior of properties.

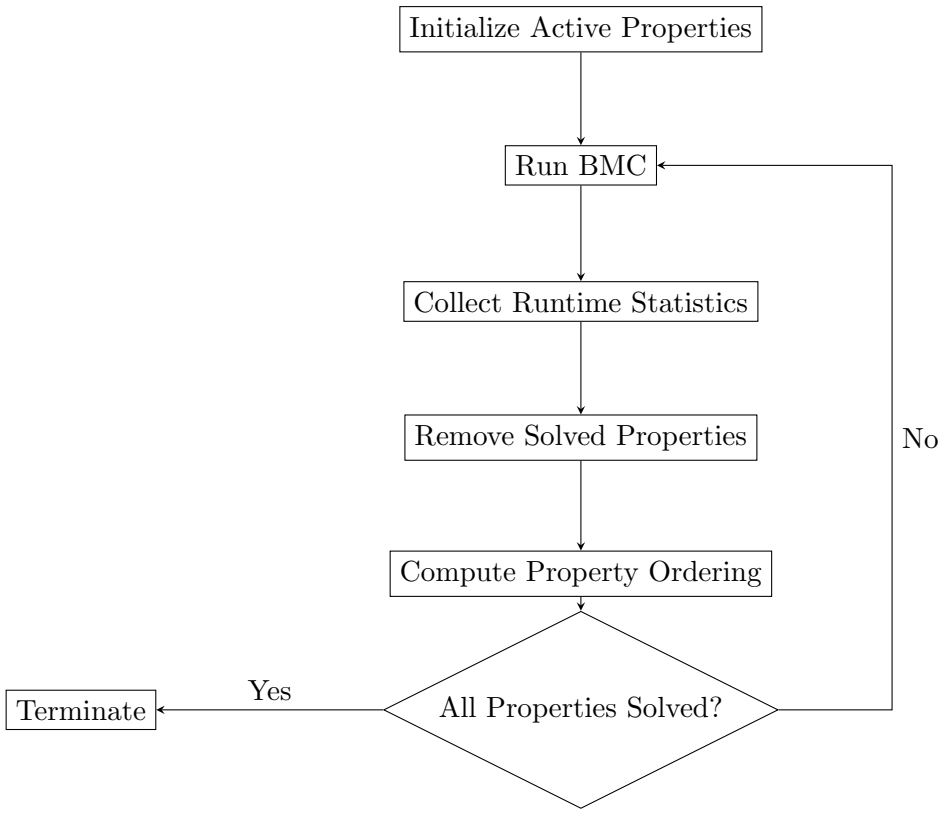

Figure 3.5: Dynamic property ordering workflow.

3.3.5 Dynamic Scheduling Workflow

The overall dynamic scheduling procedure employed by the proposed framework is shown in Figure 3.5.

The difference between the proposed algorithms lies in the manner in which the property ordering is computed. While both approaches utilize the same runtime statistics collection framework, they employ different ranking criteria to determine property priorities.

The following sections describe the proposed scheduling algorithms in detail. Section 3.4 presents ALG1, which prioritizes properties based on verification depth, while Section 3.5 introduces ALG2, which utilizes verification rate as the scheduling criterion.

3.4 Proposed Algorithm ALG1

This section presents the first proposed dynamic property ordering algorithm, referred to as ALG1. The algorithm is motivated by the observation that different properties exhibit different verification behaviors during bounded model checking. Some properties are able to reach deeper verification depths within a limited amount of verification effort, while others demonstrate relatively slow progress.

The central hypothesis behind ALG1 is that properties reaching deeper verification depths are more likely to eventually produce useful verification results, either in the form of a proof or a counterexample. Consequently, verification resources should be preferentially allocated toward such properties.

To exploit this observation, ALG1 dynamically reorders unsolved properties according to the maximum verification depth reached during previous verification intervals. Properties exhibiting greater

progress are assigned higher priority during subsequent verification rounds.

3.4.1 Motivation

Traditional multi-property verification approaches generally do not distinguish between properties of varying difficulty. As a result, verification effort may be spent on properties that make little progress, while more promising properties receive insufficient attention.

Consider the example shown in Figure 3.6. After a verification interval, different properties may reach different verification depths.

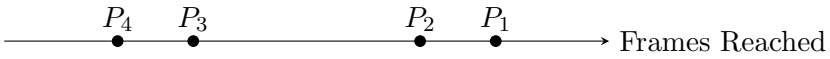

Figure 3.6: Example verification progress observed after a verification interval.

In this example, properties P_1 and P_2 have reached substantially deeper verification depths than P_3 and P_4. ALG1 interprets this behavior as evidence that P_1 and P_2 are making stronger verification progress and therefore prioritizes them during subsequent verification rounds.

3.4.2 Property Ranking Strategy

The ranking criterion employed by ALG1 is based on the maximum bounded model checking depth reached by a property.

The ranking function used by ALG1 is defined as

$$Rank(P_i) = FR(P_i) \tag{3.3}$$

Properties are sorted in descending order of their ranking values. Consequently, properties that reach deeper verification depths receive higher priority.

For example, consider the property statistics shown in Table 3.2.

Table 3.1: Example property ordering in ALG1.

Property	Frames Reached
P_1	120
P_2	95
P_3	40
P_4	15

The resulting ordering is

$$P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4.$$

3.4.3 Adaptive Verification Schedule

ALG1 employs an adaptive verification schedule controlled by a verification time budget parameter T.

Initially,

$$T = 1.$$

Each active property is verified using bounded model checking with the current time budget T. During this process, properties that are proved or falsified are removed from the active property set.

At the end of a verification round, the algorithm examines whether any property has been solved.

- If at least one property is solved, the current time allocation is retained.
- If no property is solved, the time budget is doubled.

The update rule is given by

$$T \leftarrow 2T \tag{3.4}$$

This adaptive strategy allows the algorithm to gradually increase verification effort only when necessary.

3.4.4 Dynamic Reordering Procedure

Figure 3.7 illustrates the operation of ALG1.

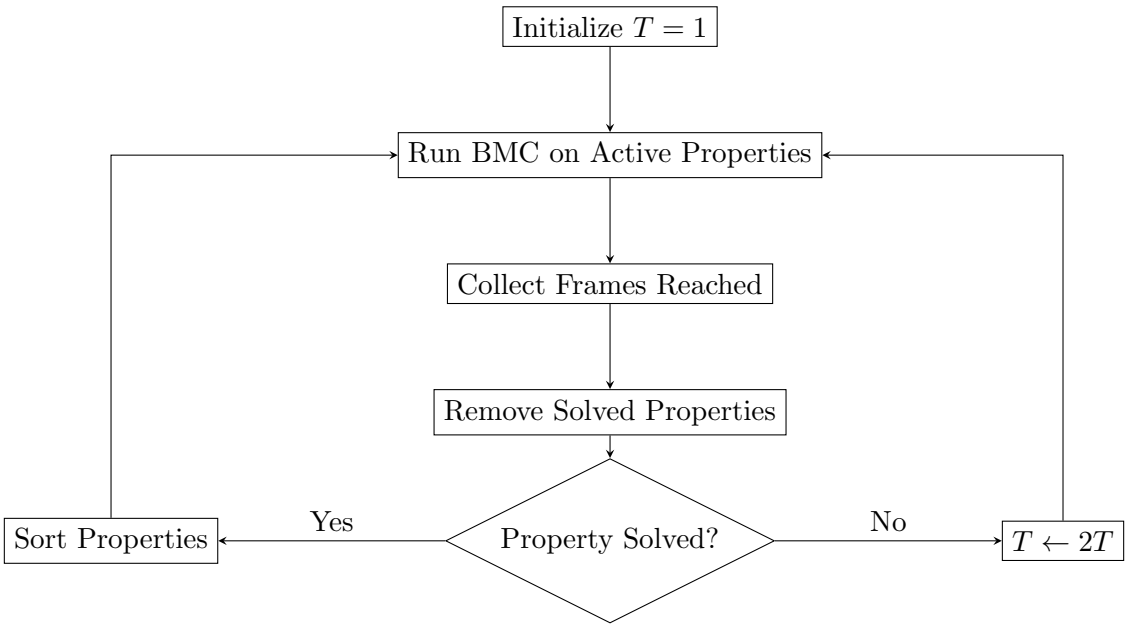

Figure 3.7: Overview of ALG1.

3.4.5 Algorithm Description

The complete procedure of ALG1 is presented in Algorithm 1. The algorithm repeatedly applies bounded model checking to active properties, collects verification statistics, removes solved properties, and updates the ordering of remaining properties according to the frames reached metric.

3.4.6 Complexity Analysis

Let n denote the number of active properties.

The dominant operation performed by ALG1 outside bounded model checking is property sorting. Since properties are sorted according to frames reached, the complexity of the ranking phase is

$$O(n \log n). \tag{3.5}$$

The computational cost of SAT solving dominates the overall runtime of the algorithm. Consequently, the overhead introduced by dynamic property ordering is negligible compared to the verification cost itself.

3.4.7 Illustrative Example

To illustrate the behavior of ALG1, consider a hypothetical verification task consisting of four properties $\{P_1, P_2, P_3, P_4\}$ and a total verification budget of 30 seconds. Initially, all properties are active and each property is allocated a verification duration of $T = 1$ second. The algorithm dynamically reorders properties according to the maximum frame reached and doubles the verification interval whenever no property is solved during an iteration.

Table 3.2: Illustrative execution of ALG1 on a design containing four properties.

Iter.	Rem. Time (s)	T (s)	Active Properties	Frames	Outcome
0	30	1	$\{P_1, P_2, P_3, P_4\}$	$\{0, 0, 0, 0\}$	Initial state. No runtime information available.
1	26	2	$\{P_3, P_2, P_1, P_4\}$	$\{100, 30, 20, 20\}$	No property solved. Properties reordered by frame count. T doubled from 1 to 2 seconds.
2	18	2	$\{P_2, P_1, P_4\}$	$\{80, 50, 30\}$	P_3 solved at frame 110 and removed from the active list. Since a property was solved, T is retained.
3	12	4	$\{P_4, P_2, P_1\}$	$\{100, 90, 60\}$	No property solved. Properties reordered by frame count. T doubled from 2 to 4 seconds.
Final	0	–	$\{P_2, P_1\}$	$\{95, 61\}$	P_4 solved at frame 120. Verification budget exhausted. Solved Properties = 2 Maximum Frame Reached = 120 Average Frame on Unsolved Properties = 78

The example demonstrates the central idea of ALG1. Properties that reach larger verification depths

are progressively promoted in the scheduling order, allowing verification effort to be concentrated on properties that exhibit greater verification progress. Furthermore, solved properties are removed from the active set, enabling the remaining verification budget to be focused on unresolved properties.

3.4.8 Discussion

ALG1 provides a simple yet effective mechanism for utilizing runtime verification information during multi-property bounded model checking. By prioritizing properties that demonstrate stronger verification progress, the algorithm attempts to focus verification effort on properties that are more likely to produce useful results.

A key advantage of ALG1 is its simplicity. The algorithm requires only a single runtime statistic, namely the maximum frame reached, and introduces minimal computational overhead.

However, the approach also has limitations. Verification depth alone does not account for the amount of time required to achieve that depth. Consequently, two properties reaching similar depths may exhibit substantially different verification efficiencies. This observation motivates the second proposed algorithm, ALG2, which incorporates both verification depth and execution time into the scheduling decision.

Algorithm 1 ALG1

```
1: P ← List(properties)
2: T ← 1
3: rem_time ← 3600
4: while len(P) > 0 and rem_time > 0 do
5:     solved_in_round ← False
6:     for each property p ∈ P do
7:         start_time ← TIME
8:         BMC3(p, T)                          ▷ bmc3 engine as discussed in Subsection 2.6.4
9:         end_time ← TIME
10:        rem_time ← rem_time − (end_time − start_time)
11:        if p is solved then
12:            Remove p from P
13:            solved_in_round ← True
14:        end if
15:        if rem_time ≤ 0 then
16:            break for
17:        end if
18:    end for
19:    if not solved_in_round then
20:        T ← T × 2                           ▷ Double T if nothing solved in this pass
21:    end if
22:    Sort(P)
23: end while
```

3.5 Proposed Algorithm ALG2

This section presents the second proposed dynamic property ordering algorithm, referred to as ALG2. While ALG1 prioritizes properties according to their absolute verification progress, ALG2 focuses

on the efficiency of verification progress. The underlying intuition is that properties reaching deeper verification depths in less time are more likely to produce useful verification outcomes and therefore should receive greater verification attention.

Unlike ALG1, which operates in rounds and periodically reorders all active properties, ALG2 adopts a priority-driven scheduling strategy. At each scheduling decision point, the algorithm selects the most promising property according to a dynamically computed priority value and allocates verification effort to that property.

3.5.1 Motivation

Although verification depth provides valuable information regarding verification progress, it does not capture the amount of computational effort required to achieve that progress. Two properties may reach similar verification depths while requiring significantly different execution times.

Consider the example shown in Table 3.3.

Table 3.3: Example illustrating the limitation of depth-only ranking.

Property	Frames Reached	Runtime (s)
P_1	100	10
P_2	100	100

Although both properties reach the same verification depth, property P_1 achieves this progress ten times faster than property P_2. Consequently, allocating additional verification effort to P_1 may yield greater verification benefits.

This observation motivates the use of a progress-efficiency metric rather than relying solely on verification depth.

3.5.2 Priority Function

ALG2 utilizes a priority function based on the ratio of verification progress to execution time.

Let

$$FR(P_i)$$

denote the maximum frame reached by property P_i, and let

$$RT(P_i)$$

denote the execution time spent verifying property P_i.

The priority value assigned to property P_i is defined as

$$Priority(P_i) = \frac{FR(P_i)}{RT(P_i)}. \tag{3.6}$$

A larger priority value indicates that a property is achieving greater verification progress per unit time and should therefore receive higher scheduling priority.

Properties are ordered in descending order of their priority values.

3.5.3 Property Selection Strategy

In contrast to ALG1, which processes all active properties during a verification round, ALG2 continuously selects the property with the highest priority value.

Let

$$A$$

denote the active property set.

At each scheduling step, ALG2 selects

$$P^* = \arg\max_{P_i \in A} Priority(P_i). \tag{3.7}$$

The selected property is then verified using the bounded model checking engine.

This greedy scheduling strategy continuously directs verification effort toward the property exhibiting the strongest verification efficiency.

3.5.4 Adaptive Verification Bound

Similar to ALG1, ALG2 maintains a verification bound parameter T that controls the depth explored by bounded model checking. However, unlike ALG1, which increases the verification bound only when no property is solved during a verification round, ALG2 employs a dynamic bound adjustment strategy that takes into account the cumulative verification effort allocated to individual properties.

Let

$$D(P_i)$$

denote the total verification time allocated to property P_i since the beginning of verification.

Furthermore, let

$$D_{\max} = \max_{P_j \in A} D(P_j)$$

denote the maximum cumulative verification time among all active properties.

After selecting a property P_i and completing a verification interval, ALG2 computes

$$\Delta = D_{\max} - D(P_i). \tag{3.8}$$

The value of Δ measures how much less verification effort has been allocated to the currently selected property compared to the most explored active property.

The verification bound is then updated according to the following rule:

$$T = \begin{cases} 2T, & \text{if } \Delta = 0, \\ T + \Delta, & \text{otherwise.} \end{cases} \tag{3.9}$$

When $\Delta = 0$, the selected property has already received verification effort comparable to the most explored active property. In this case, the verification bound is doubled in order to encourage deeper exploration.

When $\Delta > 0$, the selected property has received less verification effort than some other active properties. The verification bound is therefore increased more conservatively by adding Δ, allowing the property to gradually catch up while maintaining balanced resource allocation across the active property set.

This adaptive update strategy combines aggressive depth expansion with fairness-aware resource allocation. As a result, ALG2 attempts to balance two competing objectives: prioritizing promising properties while avoiding excessive concentration of verification effort on a small subset of properties.

Table 3.4: Example of adaptive bound adjustment in ALG2.

Property	Total Verification Time (s)
P_1	200
P_2	150
P_3	100

Suppose P_3 is selected for verification.

$$D_{\max} = 200$$

$$\Delta = 200 - 100 = 100$$

If the current bound is $T = 64$, the updated bound becomes

$$T = 64 + 100 = 164.$$

3.5.5 Dynamic Scheduling Workflow

The operation of ALG2 is illustrated in Figure 3.8.

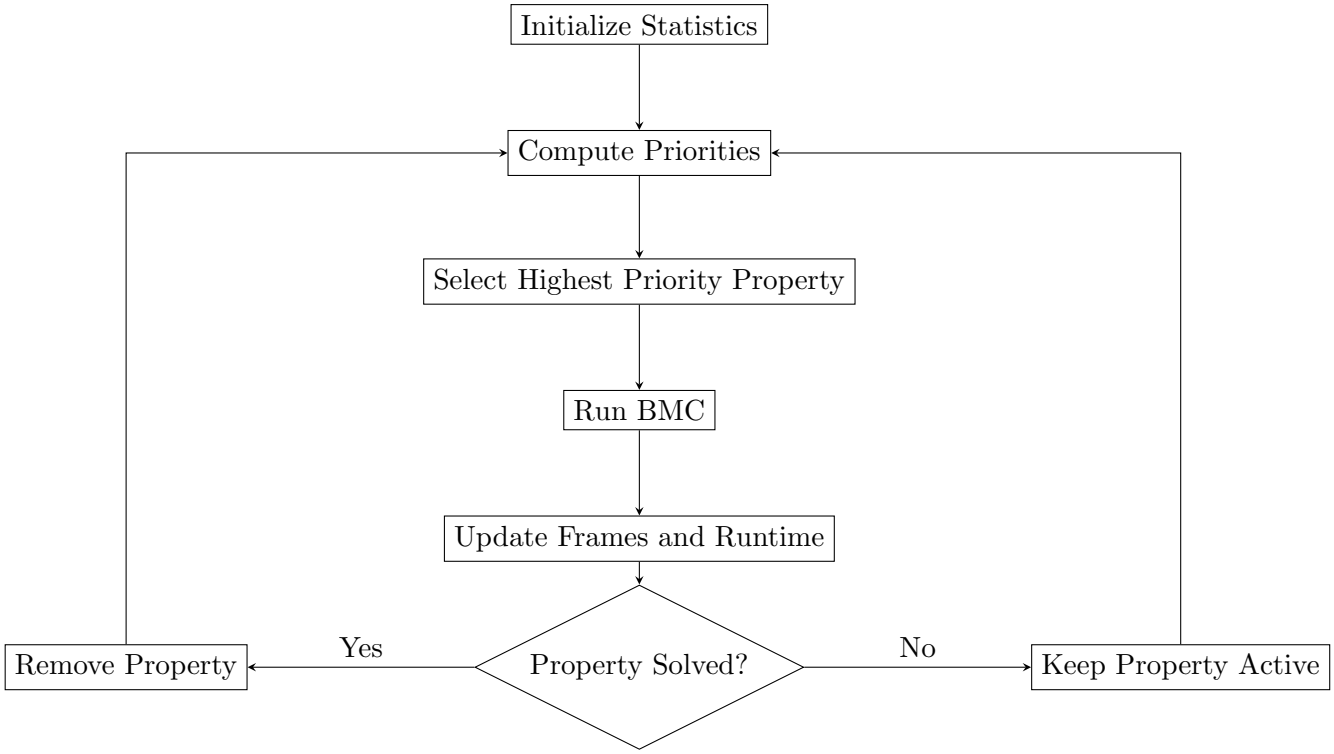

Figure 3.8: Priority-driven scheduling in ALG2.

3.5.6 Algorithm Description

Algorithm 2 presents the complete procedure of ALG2. During each iteration, the property with the highest priority value is selected and verified. Runtime statistics are subsequently updated and used to guide future scheduling decisions.

Solved properties are removed from the active property set, reducing the verification workload as execution progresses.

3.5.7 Complexity Analysis

Let n denote the number of active properties.

If priorities are maintained using a sorted structure or priority queue, selecting the highest-priority property requires

$$O(1)$$

time.

While priority update require

$$O(\log n)$$

time per scheduling decision.

As in ALG1, SAT solving dominates the overall execution cost. Consequently, the overhead introduced by priority computation and scheduling remains negligible compared to the verification effort required by bounded model checking.

3.5.8 Illustrative Example

To illustrate the behavior of ALG2, consider a hypothetical verification task consisting of four properties $\{P_1, P_2, P_3, P_4\}$ and a total verification budget of 30 seconds. Initially, no runtime statistics are available, and therefore each property is executed once for an initial duration of $T = 1$ second to collect performance information. Thereafter, properties are selected according to their Frames/Sec score, and the verification interval is adjusted using the adaptive resource allocation strategy described earlier.

Table 3.5: Illustrative execution of ALG2 on a design containing four properties.

Iter.	Rem. Time	T	Active Properties	Frames	Time Used	Outcome
0	30	1	$\{P_1, P_2, P_3, P_4\}$	$\{0, 0, 0, 0\}$	$\{0, 0, 0, 0\}$	Initial state. Each property is executed once to gather runtime statistics.
4	26	2	$\{P_1, P_2, P_3, P_4\}$	$\{20, 30, 100, 20\}$	$\{1, 1, 1, 1\}$	P_3 has the highest Frames/Sec score and is selected. P_3 is solved at frame 110 and removed.
5	24	2	$\{P_1, P_2, P_4\}$	$\{20, 30, 20\}$	$\{1, 1, 1\}$	P_2 has the highest Frames/Sec score and is selected. It reaches frame 80 but remains unsolved.
6	22	4	$\{P_1, P_2, P_4\}$	$\{20, 80, 20\}$	$\{1, 3, 1\}$	P_2 is selected again and solved at frame 120.
7	18	2	$\{P_1, P_4\}$	$\{20, 20\}$	$\{1, 1\}$	Both properties restart acceleration from exponent 1. P_1 is selected and reaches frame 50.
8	16	2	$\{P_1, P_4\}$	$\{50, 20\}$	$\{3, 1\}$	Catch-up allocation. P_4 is selected and reaches frame 30.
9	14	2	$\{P_1, P_4\}$	$\{50, 30\}$	$\{3, 3\}$	P_1 achieves the highest Frames/Sec score and reaches frame 60.
10	12	4	$\{P_1, P_4\}$	$\{60, 30\}$	$\{5, 3\}$	P_1 remains the highest-priority property and reaches frame 61.
11	8	6	$\{P_1, P_4\}$	$\{61, 30\}$	$\{9, 3\}$	Catch-up allocation. P_4 is selected and solved at frame 120.
12	2	2	$\{P_1\}$	$\{61\}$	$\{9\}$	Remaining budget allocated to P_1, which reaches frame 80 but remains unsolved.
Final	0	–	$\{P_1\}$	$\{80\}$	$\{11\}$	Solved Properties = 3 Maximum Frame Reached = 120 Average Frame on Unsolved Properties = 80

Table 3.5 illustrates the operation of ALG2. Unlike ALG1, which periodically reorders all active properties according to verification depth, ALG2 continuously selects the property with the highest Frames/Sec score. This strategy directs verification effort toward properties exhibiting the greatest verification progress per unit time. The example also demonstrates the adaptive resource allocation mechanism, where properties with smaller accumulated verification time receive catch-up allocations. As a result, ALG2 solves three properties within the available verification budget and achieves a maximum verification depth of 120 frames.

3.5.9 Discussion

ALG2 extends the ideas introduced in ALG1 by incorporating both verification depth and execution time into the scheduling decision. By considering verification efficiency rather than absolute progress alone, the algorithm attempts to identify properties that provide the greatest verification benefit per unit computational effort.

A major advantage of ALG2 is its ability to continuously focus verification resources on promising properties. The greedy scheduling mechanism enables rapid adaptation to changing verification conditions and naturally prioritizes properties exhibiting strong verification performance.

Compared to ALG1, which relies solely on verification depth, ALG2 provides a more refined measure of property usefulness by accounting for the computational cost required to achieve that depth. The effectiveness of both approaches is evaluated experimentally in Chapter 5.

Algorithm 2 ALG2

```
1: P ← List(properties)
2: T ← 1
3: rem_time ← 3600
4: while len(P) > 0 and rem_time > 0 do
5:     p ← pick best property from P by frames/sec          ▷ using priority queue
6:     start_time ← TIME
7:     BMC3(p, T)                    ▷ bmc3 engine as discussed in Subsection 2.6.4
8:     end_time ← TIME
9:     rem_time ← rem_time − (end_time − start_time)
10:    if SOLVED(p) then
11:        Remove p from P
12:    end if
13:    Increase T adaptively                    ▷ Adjust budget based on heuristic
14: end while
```

3.6 Framework Integration

The proposed dynamic property ordering algorithms were implemented within the ABC verification framework. Rather than modifying the underlying bounded model checking engine, the implementation introduces an external scheduling layer that manages property selection, runtime statistics collection, and verification resource allocation.

The implementation leverages the existing `bmc3` infrastructure provided by ABC and utilizes the Glucose SAT solver without modification. This approach preserves the correctness and efficiency of the underlying verification engine while enabling experimentation with alternative property scheduling strategies.

3.6.1 Integration Strategy

ABC provides bounded model checking functionality through the `Abc_NtkDarBmc3()` procedure implemented within the `bmcBmc3.c` module. This procedure accepts a network representation of a design together with a collection of verification parameters and performs SAT-based bounded model checking.

Instead of modifying the internal operation of the `bmc3` engine, the proposed framework operates at a higher abstraction level. Given a design containing multiple properties, the framework first decomposes the design into a collection of single-property verification instances. Each property is then represented as an independent verification network and managed by the scheduling layer.

Figure 3.9 illustrates the overall architecture of the implementation.

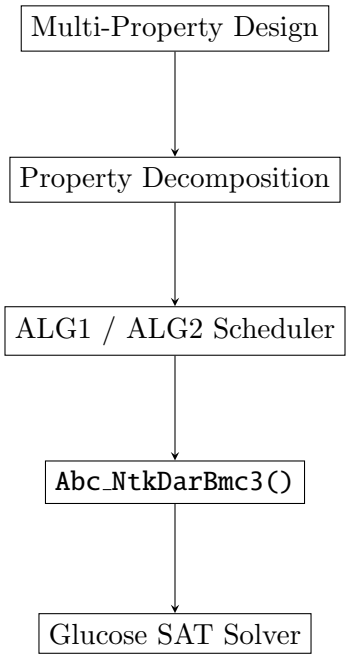

Figure 3.9: Integration of the proposed scheduling framework with ABC.

3.6.2 Property Decomposition

The input benchmark designs contain multiple verification properties represented within a single network. The first stage of the framework extracts individual properties and creates separate verification networks corresponding to each property.

Let

$$P = \{P_1, P_2, \ldots, P_n\}$$

denote the property set associated with a design.

The decomposition process transforms the original multi-property network into a collection of single-property verification instances

$$N = \{N_1, N_2, \ldots, N_n\},$$

where network N_i corresponds to property P_i.

This decomposition enables the scheduler to independently control the verification effort allocated to each property while continuing to utilize the existing `bmc3` implementation.

3.6.3 Runtime Statistics Collection

Dynamic property ordering requires runtime information describing the verification progress of each property. To support this functionality, the framework maintains a statistics record for every active property.

The implementation stores property-specific information using the `PoemData_t` structure. The collected statistics include:

- Verification status.
- Maximum frame reached.
- SAT results.
- Runtime information.
- SAT solver statistics.
- Scheduling score.

The primary fields utilized by the scheduling algorithms are:

- `nFrame`: maximum frame reached.
- `nClk`: cumulative verification time.
- `score`: scheduling priority.
- `nSolved`: property status.

These statistics are updated after every invocation of the bounded model checking engine and subsequently used by the scheduling layer.

3.6.4 Implementation of ALG1

The implementation of ALG1 maintains a list of active properties. After each verification round, runtime statistics are collected and solved properties are removed from the active set.

The remaining properties are then sorted according to the number of frames reached. The standard C library sorting routine is employed to perform the ranking operation.

The resulting ordering is subsequently used during the next verification round. This process continues until all properties are solved or the verification timeout is exhausted.

3.6.5 Implementation of ALG2

The implementation of ALG2 utilizes a priority-driven scheduling mechanism. Instead of processing all active properties during a verification round, ALG2 repeatedly selects the property with the highest scheduling priority.

To support efficient property selection, a priority queue data structure is employed. Priority values are computed using the verification efficiency metric introduced in Section 3.5. Properties exhibiting higher verification efficiency receive higher scheduling priority.

After each verification interval, the corresponding property statistics are updated and the priority queue is adjusted accordingly. Solved properties are removed from the active set and no longer participate in future scheduling decisions.

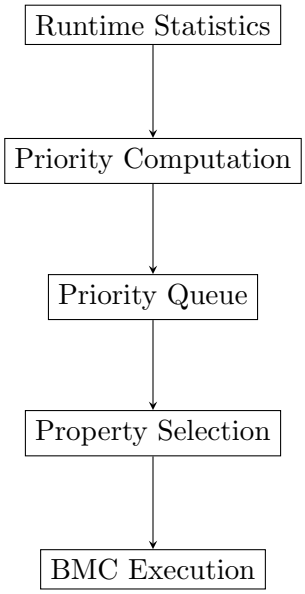

Figure 3.10: Priority-driven implementation of ALG2.

3.6.6 Interaction with the BMC Engine

The scheduling framework interacts with the bounded model checking engine exclusively through calls to the `Abc_NtkDarBmc3()` procedure. The scheduler is responsible for selecting the property to be verified and determining the verification parameters supplied to the BMC engine.

Once verification begins, the internal operation of the bounded model checking engine remains unchanged. Similarly, the Glucose SAT solver continues to operate without modification.

This separation between scheduling and verification simplifies implementation and ensures that observed performance improvements can be attributed to the proposed scheduling strategies rather than modifications to the underlying SAT solver or bounded model checking algorithms.

3.6.7 Discussion

The proposed implementation demonstrates that dynamic property ordering can be incorporated into an existing verification framework without modifying the underlying verification engine. By separating scheduling decisions from SAT solving and bounded model checking, the framework remains modular and can potentially be adapted to other verification engines.

Furthermore, the implementation introduces only a small computational overhead consisting primarily of statistics collection, sorting, and priority queue maintenance. Since SAT solving dominates the overall verification cost, the additional scheduling overhead is negligible in comparison to the total

verification runtime.

3.7 Summary

This chapter presented the proposed framework for dynamic property ordering in multi-property bounded model checking. The framework was motivated by the observation that verification properties often exhibit significantly different verification characteristics and therefore may benefit from non-uniform allocation of verification resources.

The chapter first introduced the baseline approaches used throughout this work, namely the native multi-property bounded model checking implementation provided by ABC and the Equal Time Bounding (ETB) strategy. While these approaches provide useful reference points for evaluation, neither utilizes runtime verification information to guide scheduling decisions.

To address this limitation, a dynamic property ordering framework was proposed. The framework continuously monitors verification progress, maintains an active set of unsolved properties, and utilizes runtime statistics to adapt verification decisions throughout execution. Verification depth and execution time were identified as important indicators of verification progress and were subsequently used to guide resource allocation decisions.

Two scheduling algorithms were developed within this framework. The first algorithm, ALG1, employs a round-based scheduling strategy in which properties are dynamically reordered according to the maximum verification depth reached during previous verification intervals. The algorithm is based on the intuition that properties reaching deeper verification bounds are more likely to eventually produce useful verification results and should therefore receive greater verification attention.

The second algorithm, ALG2, extends this idea by incorporating verification efficiency into the scheduling decision. Instead of relying solely on verification depth, ALG2 computes a priority value using both the achieved verification depth and the execution time required to reach that depth. A priority-driven scheduling mechanism is then used to continuously select the most promising property for verification. In addition, ALG2 employs an adaptive bound expansion strategy that balances aggressive exploration with fairness-aware resource allocation.

The chapter also described the implementation of the proposed framework within the ABC verification system. The implementation was constructed around the existing `bmc3` engine and utilized the Glucose SAT solver without modification. Dynamic scheduling functionality was introduced through an external scheduling layer responsible for property decomposition, statistics collection, property ranking, and resource allocation. This design enabled the proposed algorithms to be integrated into an established verification framework while preserving the underlying bounded model checking infrastructure.

The proposed framework introduces only modest scheduling overhead while providing the ability to adapt verification effort according to observed runtime behavior. By exploiting verification progress information, the framework aims to improve bug-finding efficiency, increase the number of solved properties, and achieve greater verification depth within a fixed verification budget.

Having described the proposed algorithms and their implementation, the next chapter presents the experimental methodology used to evaluate their effectiveness. The benchmark selection procedure, hardware environment, tool configuration, and evaluation metrics employed in this study are discussed in detail.

Chapter 4

Experimental Setup

4.1 Overview

The previous chapter presented the proposed dynamic property ordering framework together with the two scheduling algorithms, ALG1 and ALG2. Both algorithms were designed to improve the efficiency of multi-property bounded model checking by utilizing runtime verification information to guide the allocation of verification resources. While ALG1 employs a depth-based property ordering strategy, ALG2 combines verification progress and execution time to prioritize properties according to their verification efficiency.

To evaluate the effectiveness of the proposed approaches, a comprehensive experimental study was conducted using benchmark designs from the Hardware Model Checking Competition (HWMCC). The experiments compare the proposed algorithms against two baseline approaches: the native multi-property bounded model checking implementation provided by ABC and the Equal Time Bounding (ETB) strategy described in Chapter 3.

The primary objective of the evaluation is to determine whether dynamic property ordering can improve verification performance under a fixed verification budget. In particular, the experiments seek to answer the following research questions:

1. Does dynamic property ordering increase the number of solved properties?
2. Can dynamic scheduling achieve greater verification depth within a fixed timeout?
3. How do ALG1 and ALG2 compare against existing verification strategies?
4. What is the impact of different scheduling policies on overall verification efficiency?

To answer these questions, all approaches are evaluated under identical experimental conditions. Each verification method is executed using the same benchmark set, verification engine, SAT solver, and timeout configuration. This ensures that any observed performance differences can be attributed to the scheduling strategies rather than variations in the underlying verification infrastructure.

Figure 4.1 presents an overview of the experimental methodology adopted in this work.

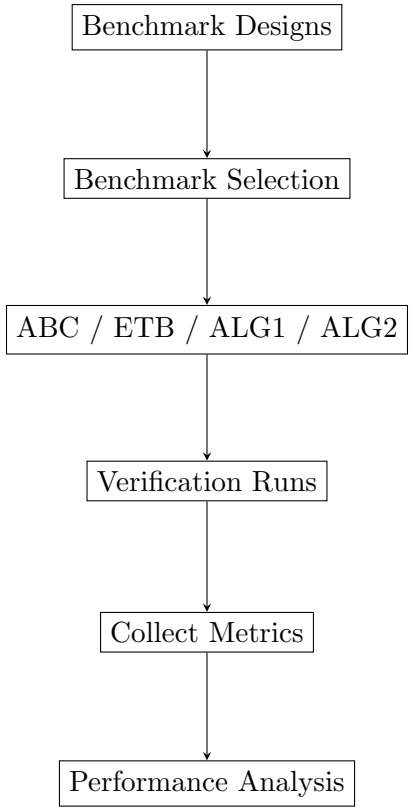

Figure 4.1: Overall experimental methodology used for evaluating the proposed algorithms.

The benchmark suite used in this study was constructed from designs originating from the HWMCC 2012 and HWMCC 2013 benchmark collections. Since these suites contain designs of widely varying complexity, a systematic benchmark selection methodology was employed to ensure that the final benchmark set represents a broad spectrum of circuit sizes and verification difficulties.

The performance of each verification approach is evaluated using several metrics, including the number of solved properties and the maximum verification depth reached. These metrics provide insight into both bug-finding effectiveness and state-space exploration capability.

The remainder of this chapter is organized as follows. Section 4.2 describes the benchmark suite and benchmark selection methodology. Section 4.3 presents the experimental hardware environment. Section 4.4 discusses the verification tool configuration. Section 4.5 introduces the evaluation metrics used throughout the study. Finally, Section 4.6 describes the experimental procedure followed during the evaluation.

4.2 Benchmark Suite

The effectiveness of the proposed dynamic property ordering algorithms was evaluated using benchmark designs from the Hardware Model Checking Competition (HWMCC). The HWMCC benchmark suites are widely used within the formal verification community and contain a diverse collection of industrial and academic verification problems exhibiting varying levels of complexity and verification difficulty [1, 2].

The benchmark designs employed in this study were obtained from the HWMCC 2012 and HWMCC 2013 benchmark collections. These benchmark suites contain sequential hardware designs represented

in the AIGER format and have been extensively used for evaluating formal verification techniques, including bounded model checking, induction, and property-directed reachability methods.

4.2.1 HWMCC Benchmark Collections

The Hardware Model Checking Competition was established to provide a common evaluation platform for formal verification tools and algorithms. Each competition benchmark consists of a hardware design together with one or more verification properties.

The HWMCC benchmark suites were selected for this study for several reasons:

- They represent realistic industrial verification problems.
- They contain designs of varying complexity.
- They are widely used in formal verification research.
- They provide a standardized basis for performance comparison.

Using established benchmark suites improves the reproducibility of experimental results and facilitates comparison with existing verification approaches reported in the literature.

4.2.2 Circuit Size Metric

Since the benchmark collections contain designs with significantly different levels of complexity, a quantitative measure of circuit size was required for benchmark selection.

For a design D, the circuit size metric is defined as

$$CktSize(D) = N_{PI} + N_{PO} + N_{AND} + N_{Latch}, \tag{4.1}$$

where

- N_{PI} denotes the number of primary inputs,
- N_{PO} denotes the number of primary outputs / properties (in the AIG context, each property corresponds to an output),
- N_{AND} denotes the number of AND gates, and
- N_{Latch} denotes the number of latches.

This metric provides a simple estimate of design complexity by accounting for both combinational and sequential components of the circuit.

4.2.3 Benchmark Selection Methodology

The combined HWMCC benchmark suites contain a large number of designs spanning a wide range of circuit sizes. Evaluating every benchmark would require substantial computational resources and would significantly increase the overall experimental cost.

To construct a representative benchmark set while maintaining manageable experimental effort, a systematic benchmark selection methodology was employed.

First, the benchmark collections from HWMCC 2012 and HWMCC 2013 were merged into a single benchmark pool. The circuit size metric defined in Equation 4.1 was then computed for every design.

Next, all benchmark designs were sorted in ascending order according to their circuit size. The sorted benchmark list was subsequently divided into forty groups of approximately equal size.

Finally, one benchmark design was selected uniformly at random from each group using the standard random number generation function `rand()`.

The resulting benchmark set contains forty designs covering the full spectrum of circuit sizes present in the original benchmark collections.

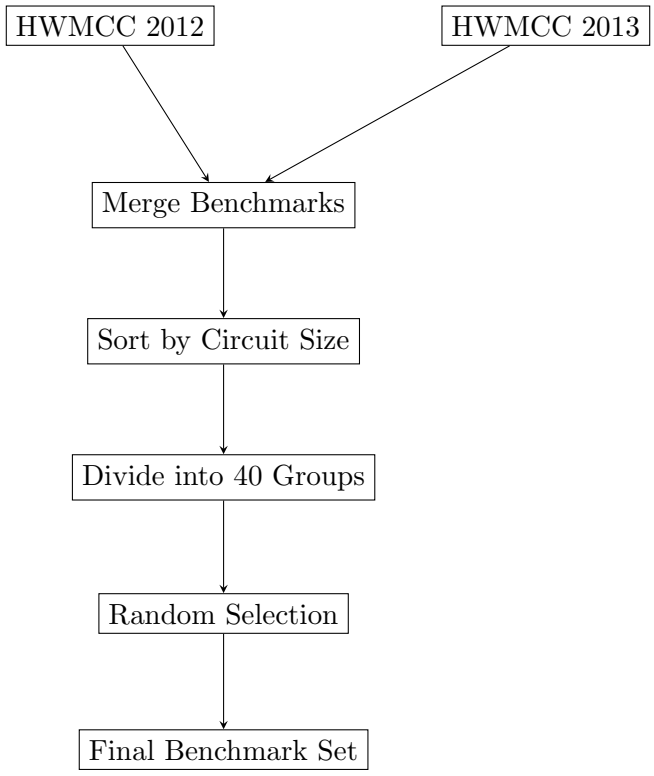

Figure 4.2: Benchmark selection methodology employed in this work.

4.2.4 Rationale for Benchmark Selection

A purely random selection from the complete benchmark pool may inadvertently overrepresent designs belonging to a particular size range while underrepresenting others. Such bias could affect the validity of experimental conclusions.

The grouping-based selection methodology adopted in this work addresses this issue by ensuring that

benchmarks are selected from across the entire circuit-size spectrum. Consequently, both small and large designs are represented within the final benchmark suite.

This approach improves the diversity of the benchmark set and increases confidence that the reported experimental results are not dominated by a narrow subset of benchmark characteristics.

The final benchmark suite therefore provides a balanced collection of verification problems suitable for evaluating the effectiveness of dynamic property ordering strategies in multi-property bounded model checking.

4.2.5 Benchmark Characteristics

The benchmark selection methodology described above resulted in a final benchmark suite consisting of forty designs. The selected benchmarks span a wide range of circuit complexities, ranging from relatively small designs containing only a few hundred circuit elements to large industrial-scale designs containing several hundred thousand elements.

Table 4.1 summarizes the characteristics of the selected benchmark suite.

Table 4.1: Summary of the selected benchmark suite.

Number of Designs	40
Minimum Circuit Size	320
Maximum Circuit Size	1,405,315

The substantial variation in circuit size ensures that the experimental evaluation includes both relatively simple and highly complex verification problems. Consequently, the benchmark suite provides a suitable environment for assessing the robustness and scalability of the proposed dynamic property ordering algorithms.

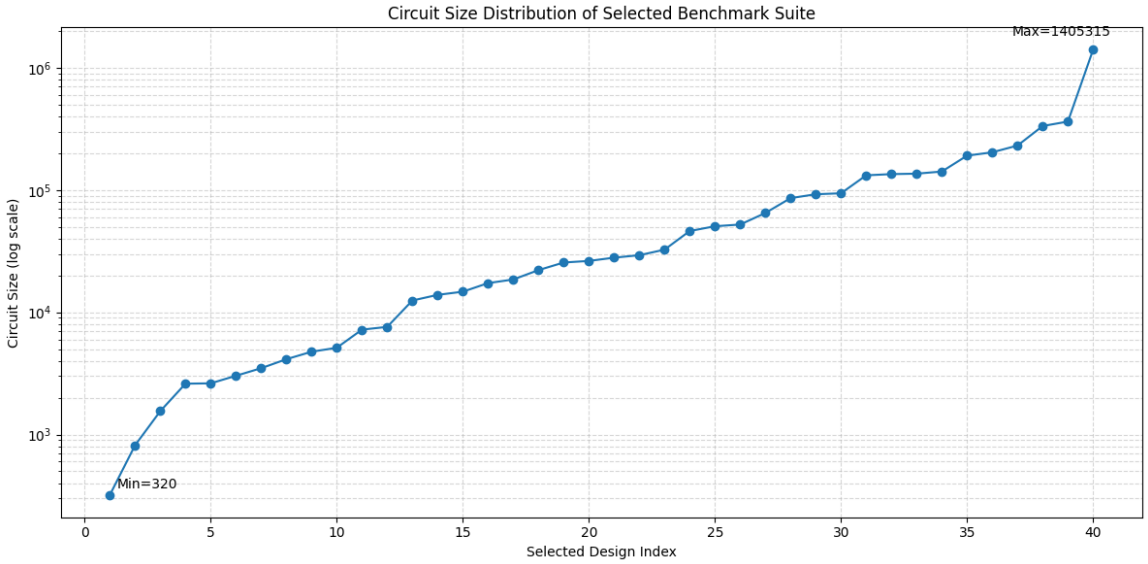

Figure 4.3: Distribution of circuit sizes in the selected benchmark suite. The logarithmic scale highlights the wide range of design complexities represented in the benchmark set, spanning from 320 to 1,405,315 circuit elements.

Figure 4.3 illustrates the distribution of circuit sizes for the selected benchmark suite. The benchmark designs span a wide range of complexities, with circuit sizes ranging from 320 to 333,709 elements.

The logarithmic scale reveals that the selected benchmarks cover more than three orders of magnitude in design size. This diversity ensures that the experimental evaluation includes both relatively small verification instances and large industrial-scale designs, providing a comprehensive assessment of the proposed dynamic property ordering algorithms.

The complete list of benchmark designs selected for experimental evaluation is provided in Appendix A.

4.3 Hardware Environment

All experiments reported in this thesis were conducted on a dedicated workstation to ensure a consistent execution environment across all evaluated verification approaches. Maintaining a fixed hardware and software configuration is important for obtaining reproducible results and enabling a fair comparison between the baseline methods and the proposed dynamic property ordering algorithms.

The hardware platform used for the experimental evaluation is summarized in Table 4.2.

Table 4.2: Experimental hardware environment.

Component	Specification
Processor	AMD Ryzen 5 5600X (6 cores, 12 threads)
Memory	128 GB RAM
Operating System	Ubuntu 22.04 LTS
Compiler	GCC 11.4

The AMD Ryzen 5 5600X processor provides sufficient computational capability for executing SAT-based bounded model checking workloads, while the availability of 128 GB of main memory ensures that large benchmark designs can be analyzed without memory limitations becoming a significant bottleneck.

All verification experiments were executed under Ubuntu 22.04 LTS using applications compiled with GCC 11.4. The use of a modern Linux-based environment provides a stable execution platform and is consistent with common practice in formal verification research and development.

The verification workloads considered in this study are primarily CPU-intensive, as bounded model checking repeatedly invokes SAT solving procedures while incrementally increasing the verification depth. Memory resources are utilized for storing circuit representations, unrolled transition relations, learned clauses, and auxiliary data structures maintained by the verification engine and SAT solver.

To ensure consistency, all benchmark designs and verification methods were evaluated on the same hardware platform using identical software configurations. Consequently, any observed differences in verification performance can be attributed to the scheduling strategies investigated in this thesis rather than variations in the underlying execution environment.

4.4 Tool Configuration

This section describes the verification tools, bounded model checking configuration, and scheduling parameters employed throughout the experimental evaluation. All verification methods were executed using the same underlying verification infrastructure to ensure that performance differences arise solely from the property scheduling strategies under investigation.

4.4.1 Verification Framework

The experiments were conducted using ABC version 1.01, an open-source framework for logic synthesis and formal verification developed at the University of California, Berkeley [8, 16]. ABC provides a collection of verification algorithms, including bounded model checking, induction-based verification, and logic optimization techniques.

The proposed algorithms were implemented as extensions to the ABC framework. Rather than modifying the internal operation of the bounded model checking engine, the implementation introduces an external scheduling layer responsible for property decomposition, runtime statistics collection, and dynamic property ordering, as described in Chapter 3.

4.4.2 Bounded Model Checking Configuration

All verification experiments utilized the `bmc3` bounded model checking engine available within ABC. The `bmc3` procedure performs SAT-based bounded model checking by incrementally unrolling the transition relation and generating satisfiability instances corresponding to increasing verification depths.

The Glucose-enabled variant of the engine, invoked through the `bmc3 -g` command, was used throughout the study. This configuration combines the bounded model checking infrastructure of ABC with the Glucose SAT solver, which is widely recognized for its effectiveness in solving industrial SAT instances [3].

For the proposed algorithms, individual properties extracted from the original multi-property design were verified by invoking the underlying `Abc_NtkDarBmc3()` procedure using the scheduling policies described in Chapter 3.

4.4.3 SAT Solver Configuration

All experiments employed the Glucose SAT solver integrated within ABC. Glucose is a conflict-driven clause learning (CDCL) SAT solver that extends MiniSAT by introducing the Literal Block Distance (LBD) heuristic for learned clause management [3].

The SAT solver configuration remained unchanged throughout the study. No modifications were made to the internal operation of Glucose, ensuring that all verification approaches utilized an identical satisfiability solving infrastructure.

Consequently, any observed performance differences can be attributed to the proposed scheduling strategies rather than variations in SAT solving behavior.

4.4.4 Scheduling Parameters

The experimental parameters used by all evaluated methods are summarized in Table 4.3.

A timeout of 3600 seconds was selected to provide sufficient verification time for large benchmark designs while maintaining a practical overall experimental duration. For both ALG1 and ALG2, the initial verification bound was set to $T = 1$, allowing the scheduling algorithms to gradually increase verification depth based on observed verification behavior.

Table 4.3: Verification tool configuration.

Parameter	Value
ABC Version	1.01
Verification Engine	`bmc3`
SAT Solver	Glucose
Verification Command	`bmc3 -g`
Total Timeout	3600 seconds
Initial Bound (T)	1
Benchmark Format	AIGER

4.4.5 Experimental Consistency

To ensure a fair comparison, all evaluated methods were executed under identical tool configurations. The same benchmark suite, bounded model checking engine, SAT solver, timeout value, and hardware environment were used throughout the study.

The only difference between the evaluated approaches lies in the mechanism used to allocate verification effort among properties. The native ABC implementation relies on its built-in multi-property verification strategy, ETB distributes verification time uniformly across properties, while ALG1 and ALG2 dynamically allocate verification resources according to runtime verification statistics.

By maintaining identical verification settings across all experiments, the study isolates the impact of dynamic property ordering on verification performance and enables meaningful comparison between the proposed algorithms and existing approaches.

4.5 Evaluation Metrics

To evaluate the effectiveness of the proposed dynamic property ordering algorithms, a set of quantitative metrics was employed. These metrics were selected to measure both the ability of a verification method to successfully solve properties and its capability to explore deeper portions of the state space within a fixed verification budget.

The evaluation focuses on three primary metrics: the number of properties solved, the maximum verification depth reached, and the average verification depth achieved across all properties. Together, these metrics provide insight into the bug-finding capability, verification progress, and overall effectiveness of the evaluated approaches.

4.5.1 Number of Properties Solved

The primary objective of bounded model checking is to determine whether a property holds or whether a counterexample exists. Therefore, the most important performance metric considered in this study is the number of solved properties.

A property is considered solved if the verification engine is able to establish either of the following outcomes:

- The property is proven to hold.

- A counterexample is found, indicating that the property is violated.

Let

$$N_{proved}$$

denote the number of properties proven correct and

$$N_{falsified}$$

denote the number of properties for which counterexamples are discovered.

The total number of solved properties is defined as

$$Solved = N_{proved} + N_{falsified}. \tag{4.2}$$

A larger value of *Solved* indicates better verification performance, as more properties are successfully resolved within the available verification budget.

4.5.2 Maximum Verification Depth

In bounded model checking, verification progress is commonly measured in terms of the number of frames explored during state-space traversal [6]. Greater verification depth generally indicates that a larger portion of the reachable state space has been analyzed.

For a property P_i, let

$$FR(P_i)$$

denote the maximum frame reached during verification.

The maximum verification depth achieved by a verification method is defined as

$$MaxFrame = \max_{P_i \in P} FR(P_i). \tag{4.3}$$

This metric reflects the deepest verification bound explored among all properties associated with a benchmark design.

A larger value of *MaxFrame* indicates stronger state-space exploration capability and may increase the likelihood of discovering deep bugs that occur only after many transition steps.

4.5.3 Average Verification Depth

While the maximum frame reached provides information about the deepest exploration achieved by a verification method, it does not describe the overall verification progress across all properties.

To obtain a more comprehensive view of verification behavior, the average verification depth is also considered.

Let

$$n$$

denote the number of properties associated with a benchmark design.

The average verification depth is defined as

$$AvgFrame = \frac{1}{n} \sum_{i=1}^{n} FR(P_i). \tag{4.4}$$

This metric captures the average amount of state-space exploration achieved across all properties. In contrast to the maximum verification depth, the average verification depth reflects the overall effectiveness of resource allocation throughout the verification process.

A higher value of *AvgFrame* indicates that verification effort is being distributed effectively among properties, resulting in deeper exploration on average.

4.5.4 Discussion

The three evaluation metrics considered in this study capture complementary aspects of verification performance. The number of solved properties measures the ability of a verification method to successfully resolve verification objectives. The maximum verification depth reflects the deepest level of state-space exploration achieved, while the average verification depth provides an indication of the overall verification progress across all properties.

Together, these metrics enable a comprehensive assessment of the proposed dynamic property ordering algorithms. The number of solved properties evaluates practical verification effectiveness, whereas the frame-based metrics provide additional insight into how efficiently verification resources are utilized during bounded model checking.

The experimental results presented in Chapter 5 use these metrics to compare the proposed algorithms against the baseline approaches introduced earlier in this thesis.

4.6 Experimental Methodology and Summary

This section describes the experimental procedure followed throughout the evaluation and summarizes the methodology adopted for comparing the proposed dynamic property ordering algorithms against the baseline approaches.

4.6.1 Experimental Procedure

The experimental evaluation was performed using the benchmark suite described in Section 4.2. For each selected benchmark design, four verification approaches were executed independently:

1. Native multi-property verification in ABC.
2. Equal Time Bounding (ETB).
3. ALG1.
4. ALG2.

Each method was executed using the same verification engine, SAT solver, timeout configuration, and hardware environment. For every verification run, the evaluation metrics defined in Section 4.5 were recorded.

The collected results were subsequently analyzed to compare the effectiveness of the different scheduling strategies in terms of property solving capability and verification depth.

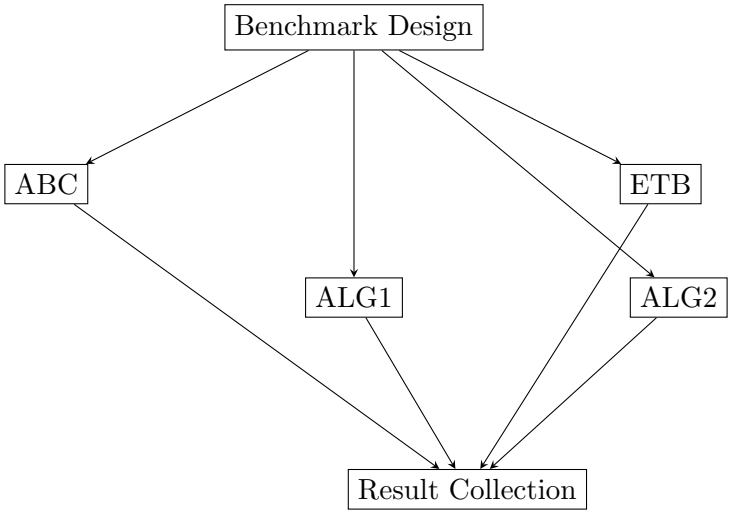

Figure 4.4: Experimental workflow used for evaluating the verification approaches.

4.6.2 Experimental Consistency

To ensure a fair comparison, all verification approaches were evaluated under identical experimental conditions. Every benchmark design was executed on the same hardware platform using the same version of ABC, the same bounded model checking engine, and the same SAT solver configuration.

The total verification timeout was fixed at 3600 seconds for all experiments. Consequently, differences in verification performance can be attributed to the scheduling strategies employed by the evaluated methods rather than variations in the underlying verification infrastructure.

4.6.3 Experimental Limitations

Although the benchmark suite covers a wide range of circuit sizes and verification complexities, several limitations should be noted when interpreting the experimental results.

- The evaluation is restricted to benchmark designs selected from the HWMCC 2012 and HWMCC 2013 benchmark suites.

- Only SAT-based bounded model checking using the `bmc3` engine is considered.
- All experiments utilize the Glucose SAT solver and do not investigate alternative SAT solving engines.
- Verification performance is evaluated using a fixed timeout of 3600 seconds.

Despite these limitations, the selected benchmark suite and evaluation methodology provide a representative environment for assessing the effectiveness of dynamic property ordering in multi-property bounded model checking.

4.6.4 Summary

This chapter presented the experimental setup used throughout the evaluation. The benchmark suite, hardware environment, verification tool configuration, and evaluation metrics were described in detail. A systematic benchmark selection methodology was employed to construct a diverse set of verification problems spanning a broad range of circuit complexities.

The experimental procedure ensures a fair comparison between the proposed algorithms and the baseline approaches by maintaining identical verification conditions across all experiments.

The next chapter presents the experimental results and analyzes the effectiveness of the proposed dynamic property ordering algorithms in improving multi-property bounded model checking performance.

Chapter 5

Experimental Results and Discussion

5.1 Overview

The previous chapter described the benchmark suite, experimental environment, verification tool configuration, and evaluation methodology employed in this study. This chapter presents the experimental results obtained from evaluating the proposed dynamic property ordering algorithms and compares their performance against the baseline verification approaches.

The primary objective of the experimental evaluation is to determine whether dynamic property ordering can improve the effectiveness of multi-property bounded model checking. In particular, the study investigates whether the proposed scheduling algorithms can solve a larger number of properties and achieve greater verification depth within a fixed verification budget.

The evaluation considers four verification approaches:

- Native multi-property verification provided by ABC.
- Equal Time Bounding (ETB).
- ALG1.
- ALG2.

All approaches were evaluated using the benchmark suite described in Chapter 4 under identical experimental conditions. The comparison focuses on the evaluation metrics introduced in Section 4.5, namely:

- Number of properties solved.
- Maximum verification depth reached.
- Average verification depth achieved on unsolved properties.

The first metric evaluates the practical effectiveness of a verification strategy in resolving verification objectives. The frame-based metrics provide additional insight into verification progress and state-space exploration capability, particularly in situations where properties remain unsolved within the allocated timeout.

Figure 5.1 illustrates the organization of the experimental evaluation presented in this chapter.

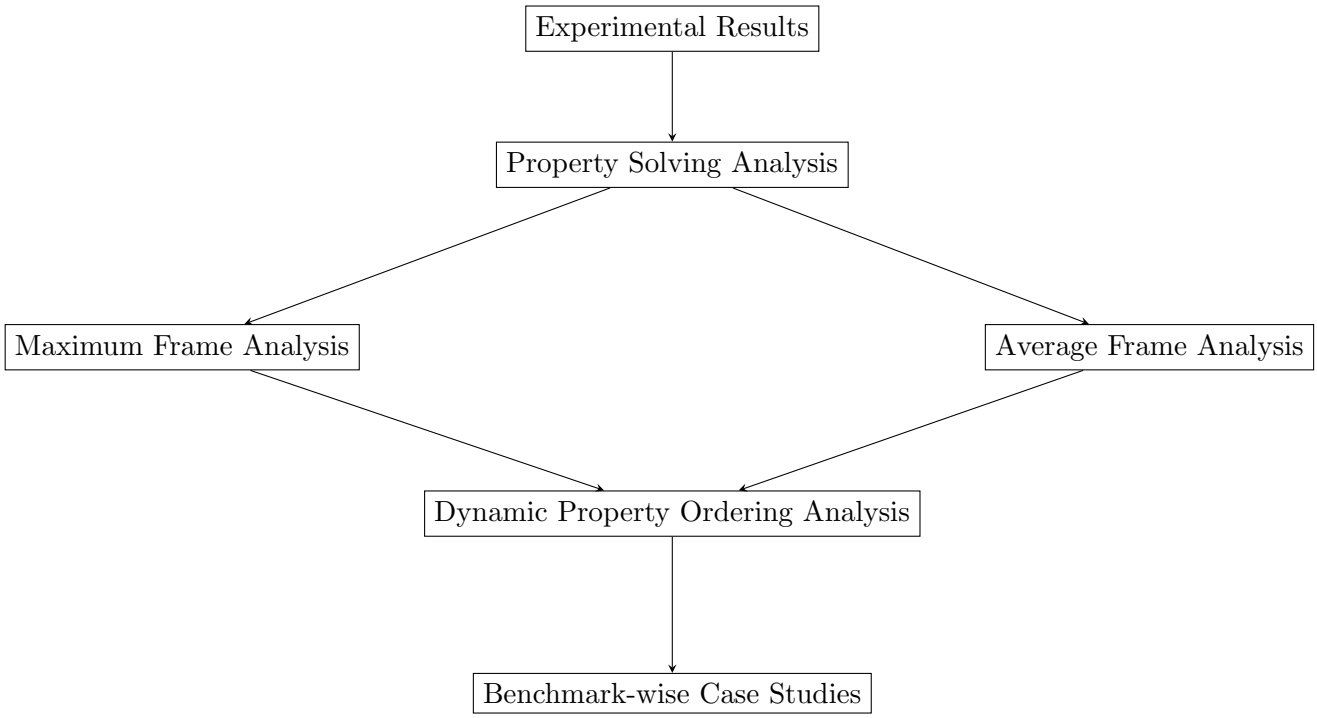

Figure 5.1: Organization of the experimental analysis presented in this chapter.

The chapter begins by analyzing the property solving performance of the evaluated approaches. Subsequently, the verification depth achieved by each method is examined using both maximum and average frame-based metrics. The impact of dynamic property ordering is then investigated in detail through comparative analysis of ALG1 and ALG2. Finally, representative benchmark designs are discussed to provide additional insight into the behavior of the proposed scheduling strategies.

Through these experiments, the chapter aims to assess the effectiveness of dynamic property ordering as a mechanism for improving multi-property bounded model checking and to identify the situations in which the proposed algorithms provide the greatest benefit.

5.2 Property Solving Performance

The primary objective of a verification strategy is to successfully resolve as many verification properties as possible within the available verification budget. Consequently, the number of solved properties serves as the most important performance metric considered in this study.

A property is considered solved when the verification engine is able to either prove the property or identify a counterexample demonstrating its violation. Therefore, a larger number of solved properties directly indicates improved verification effectiveness.

5.2.1 Benchmark-wise Property Solving Results

Figure 5.2 presents the number of solved properties obtained by ABC, ETB, ALG1, and ALG2 across the selected benchmark suite.

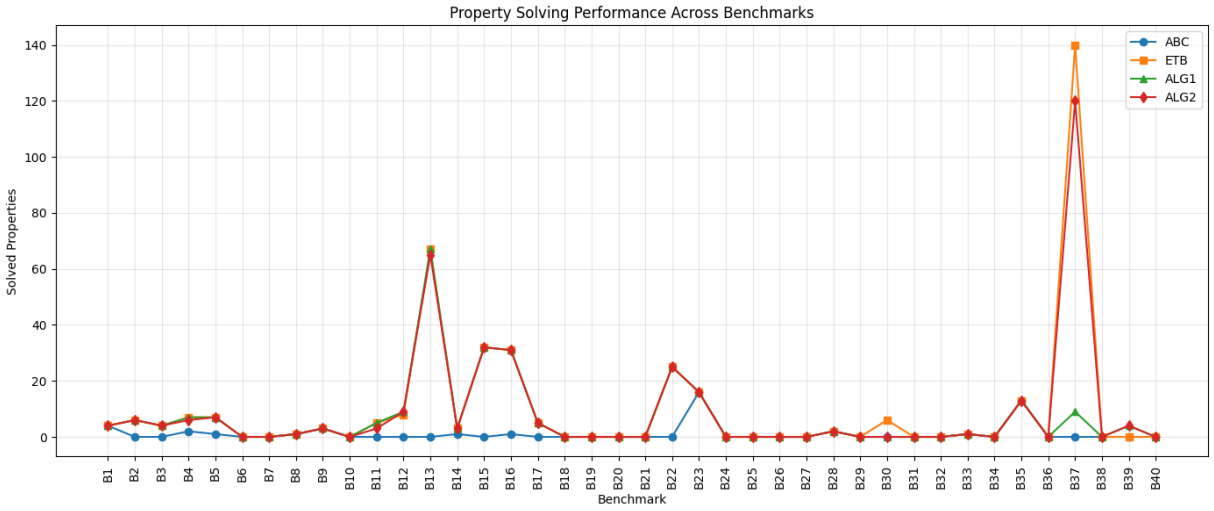

Figure 5.2: Property solving performance across benchmark designs.

The results indicate that the performance of the evaluated approaches varies across benchmark designs. For many benchmarks, all methods achieve comparable results, suggesting that the verification properties are either relatively easy or uniformly difficult. However, several benchmarks exhibit substantial differences in the number of solved properties, highlighting the importance of effective property scheduling.

In particular, benchmarks B17, B18, B24, and B39 exhibit noticeable performance differences among the evaluated approaches. The largest improvement is observed for benchmark B39, where both ALG1 and ALG2 solve substantially more properties than the native ABC implementation.

To better illustrate the benefit of dynamic property ordering, Figure 5.3 shows the additional number of properties solved by ALG1 and ALG2 relative to ABC.

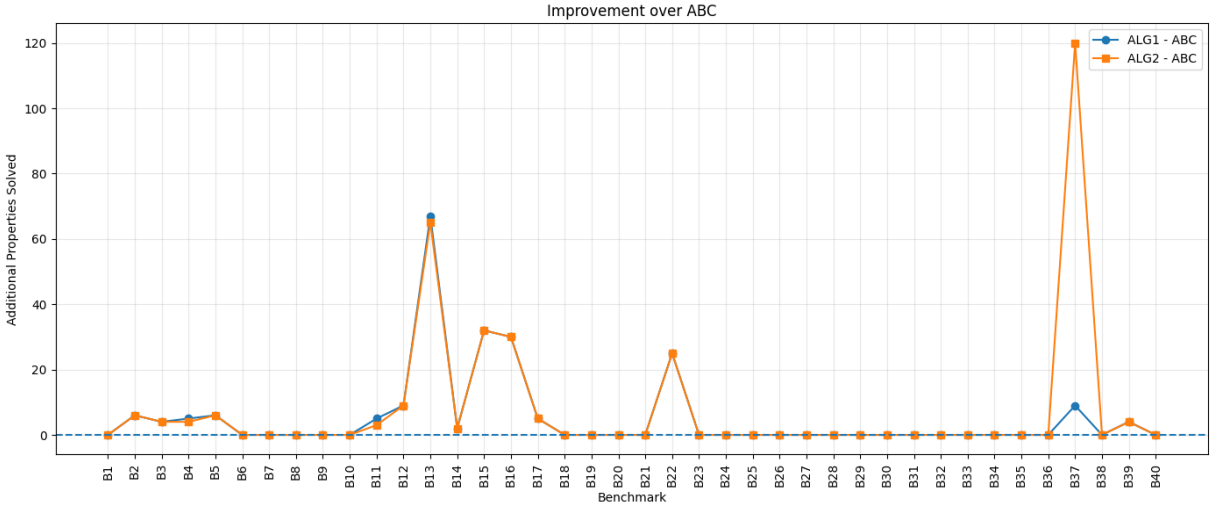

Figure 5.3: Additional properties solved by ALG1 and ALG2 relative to ABC.

The figure shows that the proposed scheduling approaches generally perform similarly to ABC on many benchmarks, while providing substantial improvements on a smaller subset of benchmark designs. This behavior is expected because the effectiveness of dynamic scheduling depends on the diversity of property difficulty within a design. When properties exhibit similar verification characteristics, the potential benefit of scheduling is naturally limited.

5.2.2 Comparative Analysis

To quantify the effectiveness of the proposed scheduling strategies, a benchmark-wise comparison was performed. For each benchmark, the number of solved properties obtained by one approach was compared against that obtained by another approach.

Table 5.1: Benchmark-wise comparison of property solving performance.

Comparison	Wins	Losses	Ties
ALG1 vs ABC	14	0	26
ALG2 vs ABC	14	0	26
ALG1 vs ETB	2	2	36
ALG2 vs ETB	2	5	33
ALG2 vs ALG1	1	3	36

Table 5.1 reveals several important observations. First, both ALG1 and ALG2 outperform the native ABC implementation on more benchmarks than they lose.

Second, the results indicate that ETB is a strong baseline for the selected benchmark suite. ALG2 lose against ETB on more benchmarks than win, although a large number of ties are observed in both. This suggests that allocating verification effort separately to individual properties already provides a significant advantage compared with native multi-property verification.

Finally, comparing the two proposed algorithms directly shows that ALG1 generally achieves better property solving performance than ALG2. ALG1 outperforms ALG2 on three benchmarks while losing on only one benchmark.

Overall, the results demonstrate that dynamic property ordering can improve property solving performance in multi-property bounded model checking. The improvements are particularly pronounced on benchmark designs containing properties with heterogeneous verification characteristics, where intelligent allocation of verification effort becomes increasingly important.

The next section investigates whether the improved property solving performance is accompanied by deeper state-space exploration through analysis of the maximum and average frame-based metrics.

5.3 Verification Depth Analysis

While the number of solved properties provides a direct measure of verification effectiveness, it does not fully capture the amount of verification progress achieved during bounded model checking. In many cases, a property may remain unsolved within the allocated timeout despite substantial exploration of the state space.

To obtain a deeper understanding of the behavior of the evaluated scheduling strategies, verification depth is analyzed using two complementary metrics introduced in Chapter 4:

- Maximum frame reached.
- Average frame reached on unsolved properties.

These metrics provide insight into how effectively each verification approach utilizes its available verification budget to explore deeper portions of the state space.

5.3.1 Maximum Frame Reached

The maximum frame reached represents the deepest verification bound explored by a verification method for a given benchmark. A larger maximum frame indicates that the verification engine was able to investigate behaviors occurring further into the execution of the design.

Figure 5.4 compares the maximum frame reached by ABC, ETB, ALG1, and ALG2 across all benchmark designs.

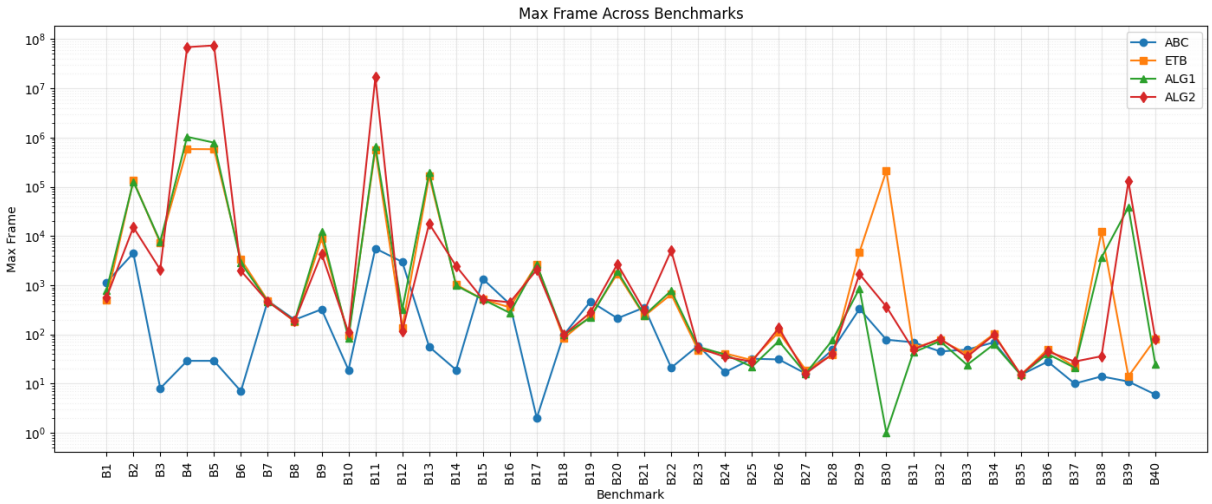

Figure 5.4: Maximum frame reached across benchmark designs. The vertical axis is shown on a logarithmic scale to accommodate the large variation in verification depth observed across benchmarks.

The results demonstrate substantial differences among the evaluated approaches. In particular, ALG2 achieves significantly larger verification depths on several benchmark designs, most notably B4, B5, and B11. These benchmarks exhibit dramatic increases in maximum frame reached compared with both ABC and ETB.

To quantify these observations, Table 5.2 summarizes the benchmark-wise comparison results.

Table 5.2: Benchmark-wise comparison using maximum frame reached.

Comparison	Wins	Losses	Ties
ALG1 vs ABC	23	14	3
ALG2 vs ABC	25	12	3
ALG1 vs ETB	16	22	2
ALG2 vs ETB	20	18	2
ALG2 vs ALG1	22	13	5

Several observations can be drawn from Table 5.2. First, ALG2 achieves deeper verification bounds than ABC on twenty-five benchmarks while losing on only twelve benchmarks. Second, ALG2 also

outperforms ETB on twenty benchmarks, demonstrating that prioritization based on verification efficiency can substantially improve state-space exploration.

Finally, the direct comparison between the proposed algorithms reveals that ALG2 achieves larger maximum verification depths than ALG1 on twenty-two benchmarks while losing on only thirteen benchmarks. This result suggests that the frames-per-second priority metric used by ALG2 provides a more effective scheduling signal than the depth-based ordering strategy employed by ALG1.

5.3.2 Average Frame Reached on Unsolved Properties

Although the maximum frame reached is useful for measuring the deepest verification depth achieved by a method, it does not characterize the overall verification progress across unsolved properties.

For this reason, the average frame reached on unsolved properties is also considered. This metric measures how deeply the verification engine explores properties that remain unresolved at the end of the verification process.

Figure 5.5 presents the average frame reached on unsolved properties for all evaluated approaches.

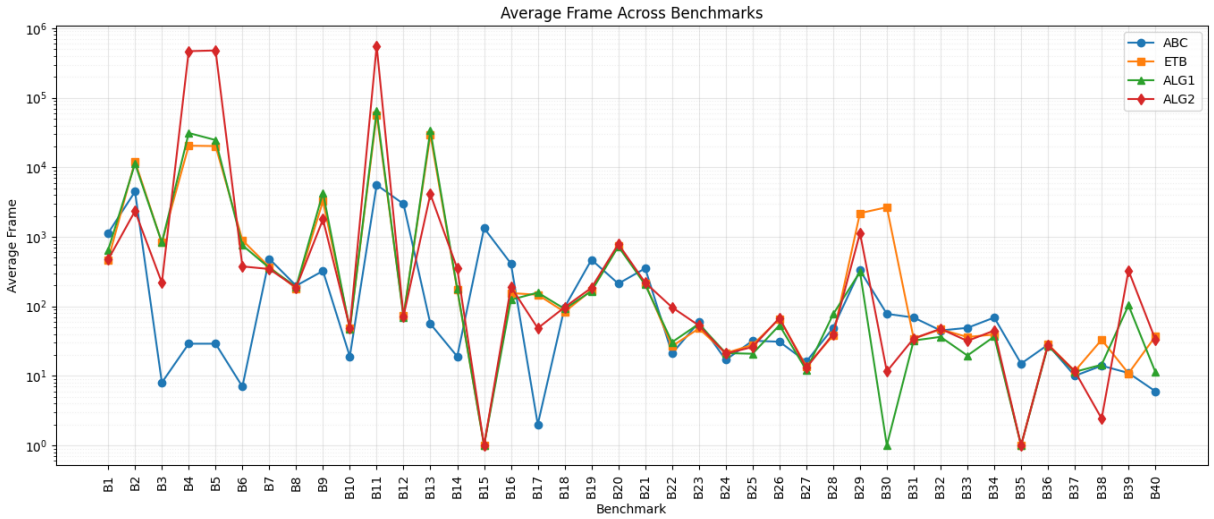

Figure 5.5: Average frame reached on unsolved properties across benchmark designs. The logarithmic scale highlights differences in verification progress spanning several orders of magnitude.

Compared with the maximum frame metric, the results are more mixed. Significant differences can be observed on benchmarks such as B4, B5, B11, and B39. However, no single approach consistently dominates across all benchmark designs.

Table 5.3 summarizes the benchmark-wise comparison results.

Table 5.3: Benchmark-wise comparison using average frame reached on unsolved properties.

Comparison	Wins	Losses	Ties
ALG1 vs ABC	20	20	0
ALG2 vs ABC	19	20	1
ALG1 vs ETB	15	23	2
ALG2 vs ETB	20	18	2
ALG2 vs ALG1	25	13	2

Unlike the maximum frame metric, the average frame results do not reveal a clear overall winner. ETB performs particularly well on several benchmark designs, indicating that uniformly distributing verification effort among properties can sometimes promote deeper exploration of unsolved properties.

Nevertheless, ALG2 maintains a slight advantage over ALG1, outperforming it on twenty-five benchmarks while losing on thirteen benchmarks. This observation is consistent with the property solving and maximum frame analyses presented earlier.

Overall, the frame-based metrics provide additional evidence supporting the effectiveness of dynamic property ordering. While the average frame metric presents a more nuanced picture, the maximum frame results strongly suggest that ALG2 is capable of directing verification effort toward properties that yield greater verification progress.

The next section investigates the underlying reasons for these improvements by examining the impact of dynamic property ordering on verification behavior.

5.4 Impact of Dynamic Property Ordering

The results presented in Sections 5.2 and 5.3 demonstrate that the scheduling strategy employed during multi-property bounded model checking can significantly influence verification performance. The proposed algorithms, ALG1 and ALG2, utilize runtime verification information to dynamically prioritize properties, thereby directing verification effort toward properties that appear more promising during execution.

Unlike the native multi-property verification strategy provided by ABC, which verifies all properties within a common verification process, the proposed approaches treat properties individually and dynamically determine the order in which verification resources are allocated. This allows the scheduling policy to adapt to the observed behavior of individual properties during verification.

5.4.1 Impact of ALG1

ALG1 prioritizes properties according to the maximum verification depth reached during previous verification intervals. The underlying intuition is that a property reaching deeper verification bounds is more likely to be approaching either a proof or a counterexample.

By allocating additional verification effort to such properties, ALG1 attempts to exploit previously observed verification progress. Consequently, verification resources are concentrated on properties that have already demonstrated the ability to advance deeper into the state space.

The experimental results support this intuition. As shown in Section 5.2, ALG1 outperforms the native ABC implementation on a greater number of benchmarks than it loses. Furthermore, the maximum frame analysis presented in Section 5.3 indicates that ALG1 frequently achieves deeper verification bounds than ABC.

These observations suggest that verification depth provides a useful indicator of future verification progress and can therefore serve as an effective scheduling signal.

5.4.2 Impact of ALG2

While ALG1 considers only verification depth, ALG2 incorporates both verification depth and verification efficiency through the frames-per-second priority metric introduced in Chapter 3.

The motivation behind this approach is that verification depth alone may not accurately reflect the cost required to achieve that progress. Some properties may reach large verification depths but require substantial computational effort, whereas other properties may advance rapidly and provide greater verification progress per unit time.

By prioritizing properties according to their observed verification efficiency, ALG2 attempts to maximize the amount of verification progress obtained from the available verification budget.

The experimental results consistently indicate that ALG2 is the strongest of the evaluated approaches. In the property solving analysis, ALG2 outperforms ABC on fourteen benchmark designs while losing on none benchmark. Similarly, in the maximum frame analysis, ALG2 achieves deeper verification bounds than both ABC and ETB on a majority of benchmarks.

Moreover, the direct comparison between ALG1 and ALG2 shows that ALG2 generally achieves superior performance in terms of maximum verification depth. These results suggest that incorporating verification efficiency into the scheduling decision provides a more informative prioritization signal than verification depth alone.

5.4.3 Comparison with Equal Time Bounding

The experimental evaluation also reveals that Equal Time Bounding (ETB) constitutes a strong baseline for the selected benchmark suite. In several experiments, ETB achieves competitive performance and occasionally outperforms the proposed approaches.

This behavior can be attributed to the fact that ETB eliminates resource contention by allocating verification effort separately to individual properties. Consequently, difficult properties are less likely to interfere with the verification progress of easier properties.

However, unlike ALG1 and ALG2, ETB does not exploit runtime verification information. Every property receives an equal share of the available verification budget regardless of its observed verification behavior. As a result, verification resources may be allocated inefficiently when properties exhibit significantly different levels of verification difficulty.

The superior performance of ALG2 on many benchmark designs suggests that incorporating runtime statistics can further improve resource allocation beyond what is achievable through uniform time distribution alone.

5.4.4 Relationship Between Property Solving and Verification Depth

An important observation emerging from the experimental results is the strong relationship between property solving performance and verification depth. Benchmarks exhibiting large improvements in the number of solved properties often also exhibit substantial increases in maximum verification depth.

This relationship is consistent with the fundamental principle of bounded model checking. Exploring deeper verification bounds increases the likelihood of discovering counterexamples and may also provide additional evidence required for proving properties.

Nevertheless, the average frame analysis indicates that deeper exploration alone does not always translate directly into additional solved properties. In several benchmark designs, ETB achieves competitive average frame values despite solving fewer properties. This observation suggests that verification depth and property solving effectiveness capture different aspects of verification performance and should therefore be considered together when evaluating scheduling strategies.

5.4.5 Summary

The experimental results demonstrate that dynamic property ordering has a measurable impact on multi-property bounded model checking. By exploiting runtime verification statistics, the proposed algorithms are able to allocate verification effort more selectively than the baseline approaches.

Among the evaluated methods, ALG2 consistently exhibits the strongest overall performance. The results suggest that combining verification progress with verification efficiency provides an effective mechanism for guiding property scheduling decisions and improving verification effectiveness within a fixed verification budget.

5.5 Benchmark-wise Analysis and Case Studies

While the aggregate comparisons presented in the previous sections provide an useful overview of verification performance, they do not fully reveal how the evaluated approaches behave on individual benchmark designs. To gain additional insight into the effectiveness of dynamic property ordering, a benchmark-wise analysis was performed.

The experimental results indicate that the benefits of dynamic scheduling are not uniformly distributed across all benchmark designs. Instead, a relatively small number of benchmarks account for a substantial fraction of the observed improvements. This behavior suggests that the effectiveness of property scheduling depends strongly on the characteristics of the underlying verification problem.

5.5.1 Case Study I: Large Improvement through Dynamic Scheduling

The most significant improvement observed during the experimental evaluation occurs for benchmark B37 (`6s401.aig`). Table 5.4 summarizes the property solving results obtained by the evaluated approaches.

Table 5.4: Property solving results for benchmark B37 (`6s401.aig`).

Method	Solved Properties
ABC	0
ETB	140
ALG1	9
ALG2	120

The results reveal a dramatic difference between the native ABC implementation and the approaches that allocate verification effort separately to individual properties. While ABC fails to solve any property within the allocated timeout, ETB, ALG1, and ALG2 all solve a substantial number of properties.

5.5.2 Case Study II: Benchmarks with Limited Scheduling Impact

The experimental evaluation also contains numerous benchmarks for which all approaches achieve identical results. Examples include B1 (`sm98tcasmulti.aig`), B8 (`6s417.aig`), B23 (`6s116.aig`), and B35 (`6s386.aig`).

For these benchmarks, the choice of scheduling strategy has little influence on the final verification outcome. This behavior may occur for several reasons.

- The benchmark may contain only a small number of properties.
- The properties may exhibit similar verification difficulty.
- The properties may be either trivially easy or uniformly difficult.

In such situations, dynamic scheduling provides limited opportunity for improving verification performance because there is little distinction among competing verification tasks.

5.5.3 Case Study III: When ALG1 Outperforms ALG2

Although ALG2 generally exhibits the strongest overall performance, several benchmarks demonstrate situations in which ALG1 performs better.

Table 5.5 summarizes the benchmark designs for which ALG1 solves more properties than ALG2.

Table 5.5: Benchmarks where ALG1 solves more properties than ALG2.

Benchmark	ALG1	ALG2
B4 (`bob12m17m.aig`)	7	6
B11 (`pdtvsar8multip.aig`)	5	3
B13 (`6s325.aig`)	67	65

These results suggest that verification depth alone can sometimes provide a stronger scheduling signal than verification efficiency. Since ALG1 prioritizes properties according to previously achieved verification depth, it may continue allocating resources to properties that are progressing steadily toward a proof or counterexample.

In contrast, ALG2 emphasizes verification efficiency through the frames-per-second metric. While this strategy is generally beneficial, it may occasionally deprioritize properties that require substantial computational effort before producing a verification result.

Consequently, the experimental results indicate that neither scheduling strategy is universally optimal. Instead, their relative effectiveness depends on the verification characteristics of the benchmark under consideration.

5.5.4 Impact of Circuit Size

The benchmark suite employed in this study spans a broad range of circuit sizes, ranging from a few hundred logic elements to more than three hundred thousand logic elements. This diversity was intentionally introduced through the benchmark selection methodology described in Chapter 4.

Inspection of the benchmark-wise results suggests that the largest performance differences tend to occur on medium and large benchmark designs. Smaller benchmarks frequently exhibit identical results across all evaluated approaches, indicating that verification effort is often sufficient to resolve the available properties regardless of the scheduling strategy employed.

As circuit complexity increases, however, verification resources become increasingly constrained. Under such circumstances, the manner in which verification effort is distributed among properties becomes more important, thereby increasing the potential benefits of dynamic property ordering.

5.5.5 Summary

The benchmark-wise analysis provides additional insight into the behavior of the evaluated approaches. While many benchmark designs exhibit little sensitivity to scheduling decisions, a small number of benchmarks account for substantial improvements in verification performance.

The results further demonstrate that ALG2 generally provides the strongest overall performance, although ALG1 remains advantageous on several benchmark designs. These observations reinforce the conclusion that runtime verification statistics can serve as an effective basis for guiding verification effort in multi-property bounded model checking.

5.6 Discussion and Summary

This chapter presented a comprehensive experimental evaluation of the proposed dynamic property ordering algorithms using benchmark designs selected from the HWMCC 2012 and HWMCC 2013 benchmark suites. The performance of ALG1 and ALG2 was compared against the native multi-property verification strategy implemented in ABC as well as the Equal Time Bounding (ETB) baseline.

The property solving results demonstrate that dynamic scheduling can significantly influence verification effectiveness. Both ALG1 and ALG2 outperform the native ABC implementation on a greater number of benchmarks than they lose. In particular, both exhibits the strongest overall property solving performance, outperforming ABC on fourteen benchmark designs while losing on none benchmark.

The verification depth analysis further supports the effectiveness of dynamic scheduling. The maximum frame results indicate that ALG2 consistently explores deeper verification bounds than both ABC and ALG1 across a majority of benchmark designs. This observation suggests that incorporating verification efficiency into the scheduling decision enables verification effort to be directed toward properties that provide greater verification progress.

The average frame analysis presents a more nuanced picture. While ALG2 maintains a slight advantage over ALG1, the results indicate that ETB remains a competitive baseline on several benchmark designs. This finding suggests that allocating verification effort separately to individual properties already provides a substantial benefit relative to native multi-property verification.

The benchmark-wise analysis reveals that the benefits of dynamic scheduling are not uniformly distributed across all designs. Many benchmarks exhibit identical behavior across all approaches, indicating limited opportunity for scheduling optimization. In contrast, several benchmarks demonstrate substantial performance improvements, with benchmark B37 (`6s401.aig`) representing the most prominent example. Such results indicate that dynamic scheduling is particularly beneficial

when properties exhibit heterogeneous verification characteristics.

A direct comparison of the proposed algorithms shows that ALG2 generally achieves superior performance. By combining verification depth and verification efficiency through the frames-per-second priority metric, ALG2 is able to make more informed scheduling decisions than ALG1, which relies solely on verification depth. Nevertheless, the existence of benchmarks where ALG1 outperforms ALG2 demonstrates that no single scheduling strategy is universally optimal.

Overall, the experimental results support the central hypothesis of this thesis: runtime verification statistics can be effectively utilized to guide property scheduling in multi-property bounded model checking. The proposed dynamic scheduling approaches improve the allocation of verification resources and frequently achieve better verification outcomes than traditional scheduling strategies.

Among the evaluated approaches, ALG2 provides the strongest overall performance and demonstrates that verification efficiency is a valuable indicator for prioritizing properties during bounded model checking. These findings suggest that dynamic property ordering represents a promising direction for improving the scalability and effectiveness of multi-property formal verification.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Formal verification has become an essential component of modern hardware design flows due to the increasing complexity of digital systems and the limitations of simulation-based verification techniques. Among the various formal verification approaches, bounded model checking (BMC) has proven to be an effective technique for detecting design errors by reducing the verification problem to a sequence of satisfiability instances.

In multi-property verification, multiple verification properties are evaluated simultaneously. While this approach can improve verification throughput, it also introduces challenges related to resource allocation and scheduling. In particular, difficult properties may consume a disproportionate amount of verification effort, potentially delaying the verification of properties that could otherwise be solved quickly. Consequently, the order in which verification resources are allocated to properties can have a significant impact on overall verification effectiveness.

This thesis investigated the problem of dynamic property ordering for efficient multi-property bounded model checking. The central hypothesis of this work was that runtime verification statistics can be utilized to guide scheduling decisions and improve the allocation of verification resources among competing properties.

To address this problem, two dynamic scheduling algorithms were proposed and implemented within the ABC verification framework. The first approach, ALG1, prioritizes properties according to the maximum verification depth previously achieved by each property. The underlying intuition is that properties reaching deeper verification bounds are more likely to be approaching a proof or counterexample and should therefore receive additional verification effort.

The second approach, ALG2, extends this idea by incorporating verification efficiency into the scheduling process. Rather than relying solely on verification depth, ALG2 prioritizes properties according to a frames-per-second metric that captures both verification progress and computational cost. This allows verification effort to be directed toward properties that provide the greatest verification progress per unit time.

The proposed algorithms were implemented directly within the ABC framework using the `bmc3` bounded model checking engine and the Glucose SAT solver. To evaluate their effectiveness, ex-

periments were conducted on a benchmark suite constructed from the HWMCC 2012 and HWMCC 2013 benchmark collections. The benchmark selection methodology ensured that the evaluation covered a broad range of circuit sizes and verification complexities.

Experimental results demonstrate that dynamic property ordering can significantly influence multi-property verification performance. Both ALG1 and ALG2 outperform the native multi-property verification strategy provided by ABC on a greater number of benchmark designs than they lose. Furthermore, the frame-based analyses indicate that dynamic scheduling frequently enables deeper state-space exploration, thereby increasing verification progress within the available verification budget.

Among the evaluated approaches, ALG2 consistently exhibits the strongest overall performance. The experimental results suggest that combining verification depth with verification efficiency provides a more informative scheduling signal than verification depth alone. Although ALG1 remains competitive and performs better on certain benchmark designs, ALG2 generally achieves superior property solving performance and larger verification depths across the benchmark suite.

The benchmark-wise analyses further reveal that the benefits of dynamic scheduling are highly dependent on the characteristics of the verification problem. While some benchmark designs exhibit little sensitivity to scheduling decisions, others demonstrate substantial performance improvements when runtime verification statistics are incorporated into the scheduling process. These observations highlight the importance of adaptive verification strategies in complex multi-property verification environments.

In summary, this thesis demonstrates that dynamic property ordering is an effective mechanism for improving the efficiency of multi-property bounded model checking. By utilizing runtime verification information to guide scheduling decisions, the proposed approaches improve the allocation of verification resources and frequently achieve better verification outcomes than traditional scheduling strategies. The results indicate that adaptive property scheduling represents a promising direction for enhancing the scalability and effectiveness of formal verification tools.

6.2 Limitations and Future Work

Although the proposed dynamic property ordering approaches demonstrate promising results, several limitations of the present study should be acknowledged.

First, the experimental evaluation is restricted to SAT-based bounded model checking using the `bmc3` verification engine within the ABC framework. While bounded model checking is highly effective for bug detection and shallow proofs, other formal verification techniques may exhibit different verification characteristics and scheduling requirements.

Second, all experiments were performed using the Glucose SAT solver. Different SAT solvers employ distinct heuristics and learning strategies, which may influence the effectiveness of runtime-based scheduling decisions. Consequently, the observations presented in this thesis may not directly generalize to all SAT-solving environments.

Third, the proposed scheduling algorithms utilize only a limited set of runtime statistics. ALG1 relies primarily on verification depth, while ALG2 incorporates verification depth and verification efficiency. Although these metrics provide useful scheduling signals, additional runtime information may further improve scheduling quality.

Finally, the experimental evaluation was conducted using benchmark designs selected from the HWMCC 2012 and HWMCC 2013 benchmark suites under a fixed timeout configuration. While the selected benchmarks cover a broad range of circuit sizes and verification complexities, additional benchmark suites and experimental settings may provide further insight into the generality of the proposed approaches.

Despite these limitations, the results obtained in this thesis suggest several promising directions for future research.

Extension to Other Verification Engines

The current work focuses exclusively on bounded model checking. An important direction for future research is the application of dynamic property ordering techniques to other formal verification engines, particularly Property Directed Reachability (PDR) and IC3-based verification methods.

Unlike bounded model checking, PDR incrementally constructs inductive invariants and often exhibits substantially different runtime behavior. Investigating how runtime statistics can be exploited within PDR-style verification frameworks may lead to new scheduling strategies capable of improving proof generation efficiency in multi-property verification environments.

Hybrid Scheduling Strategies

The experimental results indicate that neither ALG1 nor ALG2 is universally superior on all benchmark designs. While ALG2 generally provides the strongest overall performance, several benchmarks demonstrate situations in which ALG1 achieves better results.

This observation motivates the development of hybrid scheduling approaches that dynamically combine multiple prioritization strategies. Such approaches may adaptively switch between depth-oriented and efficiency-oriented scheduling policies according to the observed verification behavior of the benchmark under consideration.

Utilization of Richer Runtime Statistics

The implementation developed in this thesis already collects several runtime statistics, including verification depth, learned clauses, decisions, conflicts, and propagations. Future work may investigate how these statistics can be incorporated into more sophisticated scheduling policies.

For example, scheduling decisions could be guided by measures of SAT solver activity, conflict generation rates, or clause learning effectiveness. Such information may provide a more detailed characterization of property difficulty and verification progress than frame-based metrics alone.

Parallel and Multi-Core Scheduling

Modern verification environments increasingly rely on multi-core computing platforms. The scheduling approaches proposed in this thesis operate in a sequential setting in which a single property receives verification effort at a given time.

Future research may explore parallel scheduling strategies that distribute properties across multiple

processing cores while dynamically adjusting resource allocation according to runtime verification statistics. Such approaches could potentially combine the benefits of dynamic scheduling with the computational advantages of parallel verification.

Machine Learning Based Scheduling

Another promising research direction involves the use of machine learning techniques to predict property verification difficulty and guide scheduling decisions automatically. Rather than relying on manually designed priority functions, learning-based approaches could exploit historical verification data to identify scheduling policies that maximize verification effectiveness.

Combining machine learning models with runtime verification statistics may enable adaptive scheduling strategies capable of responding to a wide variety of verification scenarios.

Closing Remarks

The work presented in this thesis demonstrates that runtime verification information can be effectively utilized to guide property scheduling in multi-property bounded model checking. The proposed algorithms provide evidence that adaptive scheduling can improve both property solving performance and verification depth compared with traditional scheduling approaches.

It is hoped that the ideas explored in this work will contribute toward the development of more intelligent and scalable formal verification methodologies capable of addressing the growing complexity of modern digital systems.

Appendix A

Selected Benchmark Designs

Table A.1 lists the benchmark designs selected from the HWMCC 2012 and HWMCC 2013 benchmark suites. The designs are ordered according to increasing circuit size.

Table A.1: Selected benchmark designs used in the experimental evaluation.

Benchmark	Design	Inputs	Outputs	Latches	AND Gates	Circuit Size
B1	sm98tcasmulti.aig	142	7	1	170	320
B2	bob12m16m.aig	13	62	115	626	816
B3	6s165.aig	35	16	65	1436	1552
B4	bob12m17m.aig	57	164	260	2123	2604
B5	bob12m18m.aig	57	163	261	2140	2621
B6	6s169.aig	58	16	145	2802	3021
B7	6s291.aig	3	88	839	2555	3485
B8	6s417.aig	799	13	11	3305	4128
B9	mentorbm1.aig	224	83	70	4376	4753
B10	6s317.aig	42	32	198	4849	5121
B11	pdtvsar8multip.aig	23	33	195	6956	7207
B12	6s124.aig	199	636	6	6748	7589
B13	6s325.aig	634	301	1756	9782	12473
B14	oski1.aig	11445	50	39	2338	13872
B15	6s155.aig	201	32	513	13976	14722
B16	6s292.aig	125	247	3190	13748	17310
B17	6s154.aig	193	32	128	18183	18536
B18	6s284.aig	192	2	2352	19561	22107
B19	6s327.aig	6285	33	3290	15906	25514
B20	6s275.aig	844	673	3196	21571	26284
B21	6s429.aig	13335	1448	1445	11726	27954
B22	bob12m08m.aig	79	132	1994	27069	29274
B23	6s116.aig	4912	17	4922	22649	32500
B24	6s340.aig	480	76	3337	42291	46184
B25	bob12m05m.aig	437	10	3956	46101	50504
B26	6s138.aig	439	16	4008	47702	52165
B27	bob12m07m.aig	25	53	1258	63503	64839
B28	6s175.aig	11684	3	7415	66693	85795
B29	6s273.aig	983	42	15544	75718	92287
B30	6s381.aig	1210	932	12321	79513	93976
B31	6s409.aig	205	184	10523	120917	131829
B32	6s406.aig	208	122	10746	123785	134861
B33	6s404.aig	202	6	9801	126011	136020
B34	6s407.aig	201	371	11379	129624	141575
B35	6s386.aig	18612	13	14094	158198	190917
B36	6s329.aig	1560	33	31947	169790	203330
B37	6s401.aig	229	304	12309	218057	230899
B38	6s137.aig	647	589	32922	299551	333709
B39	6s387.aig	1121	409	29494	331746	362770
B40	6s332.aig	83041	157	83717	1238400	1405315

Appendix B

Detailed Experimental Results

B.1 Property Solving Results

Table B.1 presents the number of solved properties obtained by each verification approach for all benchmark designs considered in this study.

Table B.1: Property solving results for all benchmark designs.

Benchmark	ABC	ETB	ALG1	ALG2
B1	4	4	4	4
B2	0	6	6	6
B3	0	4	4	4
B4	2	7	7	6
B5	1	7	7	7
B6	0	0	0	0
B7	0	0	0	0
B8	1	1	1	1
B9	3	3	3	3
B10	0	0	0	0
B11	0	5	5	3
B12	0	8	9	9
B13	0	67	67	65
B14	1	3	3	3
B15	0	32	32	32
B16	1	31	31	31
B17	0	5	5	5
B18	0	0	0	0
B19	0	0	0	0
B20	0	0	0	0
B21	0	0	0	0
B22	0	25	25	25
B23	16	16	16	16
B24	0	0	0	0
B25	0	0	0	0
B26	0	0	0	0
B27	0	0	0	0

Continued on next page

Benchmark	ABC	ETB	ALG1	ALG2
B28	2	2	2	2
B29	0	0	0	0
B30	0	6	0	0
B31	0	0	0	0
B32	0	0	0	0
B33	1	1	1	1
B34	0	0	0	0
B35	13	13	13	13
B36	0	0	0	0
B37	0	140	9	120
B38	0	0	0	0
B39	0	0	4	4
B40	0	0	0	0

B.2 Max Frame Results

Table B.2: Max Frame results for all benchmark designs.

Benchmark	ABC	ETB	ALG1	ALG2
B1	1131	504	783	557
B2	4467	133341	127251	15047
B3	8	7223	7707	2072
B4	29	583332	1034489	68745332
B5	29	579970	788017	74483286
B6	7	3363	2843	1960
B7	483	484	464	464
B8	198	181	191	187
B9	323	8832	12325	4378
B10	19	96	84	109
B11	5582	566617	669577	16701508
B12	2966	138	319	117
B13	56	165585	196909	17747
B14	19	1032	982	2456
B15	1333	513	513	513
B16	413	354	273	447
B17	2	2671	2629	2049
B18	97	83	97	97
B19	468	236	220	280
B20	212	1717	1913	2693
B21	352	236	244	302
B22	21	652	773	5051
B23	59	48	56	53
B24	17	41	39	35
B25	32	30	22	28
B26	31	111	73	135
B27	16	19	16	16
B28	49	38	77	40
B29	329	4679	857	1684

Continued on next page

Benchmark	ABC	ETB	ALG1	ALG2
B30	78	210213	0	366
B31	69	53	43	49
B32	45	77	73	82
B33	49	40	24	35
B34	69	102	63	98
B35	15	15	15	15
B36	28	50	40	45
B37	10	22	21	28
B38	14	12307	3646	36
B39	11	14	38286	128313
B40	6	81	25	82

B.3 Average Frame Results

Table B.3: Average Frame results for all benchmark designs.

Benchmark	ABC	ETB	ALG1	ALG2
B1	1131	467	649	470
B2	4467	12223	11272	2346
B3	8	825	826	218
B4	29	20449	31142	468329
B5	29	20209	24659	479512
B6	7	892	755	375
B7	483	368	357	344
B8	198	181	191	187
B9	323	3265	4329	1782
B10	19	48	46	49
B11	5582	56937	65412	557285
B12	2966	73	70	72
B13	56	29161	33922	4104
B14	19	174	178	351
B15	1333	0	0	0
B16	413	155	124	190
B17	2	146	157	48
B18	97	83	92	97
B19	468	170	165	184
B20	212	736	727	787
B21	352	214	204	219
B22	21	26	30	96
B23	59	48	56	53
B24	17	21	21	20
B25	32	27	20	26
B26	31	66	53	67
B27	16	13	12	13
B28	49	38	77	40
B29	329	2191	312	1132
B30	78	2663	0	11
B31	69	34	32	34

Continued on next page

Benchmark	ABC	ETB	ALG1	ALG2
B32	45	46	36	47
B33	49	36	19	31
B34	69	39	36	44
B35	15	0	0	0
B36	28	28	27	27
B37	10	11	11	11
B38	14	33	14	2
B39	11	10	103	327
B40	6	36	11	33

Bibliography

[1] Hardware model checking competition 2012 benchmarks, 2012.

[2] Hardware model checking competition 2013 benchmarks, 2013.

[3] Audemard, G., and Simon, L. Predicting learnt clauses quality in modern sat solvers. In *IJCAI* (2009).

[4] Biere, A., Cimatti, A., Clarke, E., and Zhu, Y. Symbolic model checking without bdds. In *TACAS* (1999).

[5] Biere, A., Cimatti, A., Clarke, E., and Zhu, Y. Symbolic model checking without bdds. In *TACAS* (1999).

[6] Biere, A., Cimatti, A., Clarke, E., and Zhu, Y. Symbolic model checking without bdds. In *TACAS* (1999).

[7] Biere, A., et al. *Handbook of Satisfiability*. 2009.

[8] Brayton, R., and Mishchenko, A. Abc: An academic industrial-strength verification tool. In *Computer Aided Verification (CAV)* (2010), vol. 6174 of *LNCS*, Springer, pp. 24–40.

[9] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L.-J. Symbolic model checking: 10^{20} states and beyond. In *LICS* (1992).

[10] Clarke, E. M., Grumberg, O., and Peled, D. *Model Checking*. MIT Press, 1999.

[11] Clarke, E. M., Grumberg, O., and Peled, D. *Model Checking*. MIT Press, 2018.

[12] Cook, S. A. The complexity of theorem-proving procedures. *Proceedings of STOC* (1971).

[13] Das, S., Hazra, A., and Dasgupta, P. Purse: Property ordering using runtime statistics for efficient multi-property verification. In *Design, Automation and Test in Europe Conference (DATE)* (2024).

[14] Davis, M., Logemann, G., and Loveland, D. A machine program for theorem proving. *Communications of the ACM* (1962).

[15] Marques-Silva, J., and Sakallah, K. Grasp: A search algorithm for propositional satisfiability. In *IEEE Transactions on Computers* (1999).

[16] Mishchenko, A., and Brayton, R. Abc: A system for sequential synthesis and verification, 2007. Berkeley Logic Synthesis and Verification Group.

[17] Roorda, J., and Claessen, K. Multi-property verification.

[18] Tseitin, G. S. On the complexity of derivation in propositional calculus.