JOHANNES KEPLER
UNIVERSITÄT LINZ

Submitted by
Emily Zhengqi Yu

Submitted at
**Institute for
Formal Models
and Verification**

First Supervisor
**Univ.-Prof. Dr.
Armin Biere**

Second Supervisor
**Univ.-Prof. Dr.
Martina Seidl**

July 2023

Certifying Hardware Model Checking

Doctoral Thesis
to obtain the academic degree of
Doktorin the Technischen Wissenschaften
in the Doctoral Program
Technischen Wissenschaften

**JOHANNES KEPLER
UNIVERSITÄT LINZ**
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Abstract

Ensuring the correctness of safety-critical systems is of paramount importance, and it is becoming increasingly vital to utilize formal verification techniques to achieve this. In automated reasoning for formal verification, verifiers generate certificates in the form of machine checkable proofs, in addition to providing simply a yes or no verification result. These certificates are then validated independently by formally verified tools, and can be be subject to an additional layer of verification by trusted theorem provers. The research endeavors in this thesis are focused on the domain of hardware model checking.

This thesis introduces a novel certificate format that is highly adaptable for various model checking techniques. The novel witness circuit certificate contains an inductive invariant and can be readily verified via simple SAT checks. The framework initially involved a one-alternation QBF check for the witness circuit, but through further study and experimentation, we were able to eliminate the quantifiers and simplify the certification process to pure SAT checks. The certificate format was further refined to enable more advanced preprocessing model checking techniques, such as phase abstraction and cone-of-influence reduction, to be incorporated.

The certification framework proposed covers a range of model checking techniques, including the well-established temporal induction (also known as k-induction). This research also contributes by providing a concrete construction of a certificate specifically for this technique, effectively utilizing the certificate format. Additionally, to demonstrate the versatility of our approach, the scope is expanded from bit-level to word-level model checking by leveraging quantifier-free fixed-size bit-vectors.

Furthermore, the framework extends to certifying preprocessing techniques for model checking. The certificate format is effectively applied to temporal decomposition, enabling the generation of a single witness circuit for the entire model checking process, including both preprocessing and base model checking. In addition, this thesis addresses another popular and important preprocessing technique namely phase abstraction. Notably, the proposed certification approach is complete, such that whenever model checking is successful, a valid certificate can be produced.

The certification framework has been implemented in the tool suite, CERTIFAIGER, along with its extensions, rigorously evaluated through empirical experiments, demonstrating the effectiveness and practicality of the methodology. While not a primary focus of the research, it is worth noting that our work has yielded valuable benchmarks that can be of use to both the QBF and SAT communities.

Zusammenfassung

Die Korrektheit von sicherheitskritischen Systemen ist von höchster Bedeutung. Um dies zu gewährleisten, werden formale Verifikationstechniken eingesetzt. Dabei werden Zertifikate in Form von maschinenüberprüfbaren Beweisen generiert. Unsere Forschung konzentriert sich auf den Bereich der Hardware-Modellprüfung.

Diese Arbeit präsentiert ein neues Zertifikatsformat, das für eine Vielzahl von Modellprüftechniken eingesetzt werden kann. Unsere Zertifikate sind eine Kombination aus einem boolesche Schaltkreise und einer induktiven Invariante. Die erste Version unseres Zertifikats benötigte einen QBF-Test mit einer Alternierung, um verifiziert zu werden. Später gelang es uns, den Quantor zu eliminieren und eine vollständige Verifizierung nur durch SAT-Tests zu ermöglichen. Wir haben unser Zertifikatsformat weiter verfeinert, um fortgeschrittenere Vorverarbeitungstechniken wie Lokalisierung und Cone of Influence Analyse zu ermöglichen.

Unsere Forschung umfasst eine Reihe von Modellprüftechniken, einschließlich der etablierten temporalen Induktion (auch bekannt als k-Induktion). Darüber hinaus zeigen wir wie unser Ansatz erweitert werden kann, um Modellprüfung auf Wort-Ebene via quantorenfreier Bit-Vektor Logik zu zertifizieren.

Des Weiteren behandeln wir Vorverarbeitungstechniken, die effizientere Modellprüfung ermöglichen. Wir wenden unser Zertifikatsformat erfolgreich auf die temporale Dekomposition eines Modells an und zeigen wie ein einziges Zertifikat für den gesamten Prozess der Vorverarbeitung und Modellprüfung erzeugt werden kann. Das Gleiche gelingt uns für die verwandte Technik der Phasenabstraktion.

Wir haben unseren Ansatz in der Anwendung CERTIFAIGER und deren Erweiterungen implementiert. Empirische Experimente zeigen die Wirksamkeit und Praktikabilität unserer Methodik. Wir sind zuversichtlich, dass unser Ansatz auch auf verwandte Bereiche angewendet werden kann. Obwohl dies nicht der Schwerpunkt unserer Forschung war, haben wir wertvolle Benchmarks entwickelt, die für die QBF- und SAT-Gemeinschaft von Nutzen sein können.

Acknolwedgement

First and foremost, I would like to express my deepest gratitude to my supervisor, Armin Biere. His guidance and support throughout my PhD study have been invaluable. Without his expertise and encouragement, this thesis would not have been possible. I am truly grateful to have been introduced to the world of hardware verification and explore the fascinating research problems.

I am also immensely thankful to my second supervisor, Martina Seidl, for her invaluable support, particularly during the early stages of the PhD journey.

I extend my heartfelt appreciation to my co-authors as well. The insightful discussions and collaboration with them have contributed significantly to my growth and knowledge in this scientific exploration. Their wisdom and guidance have been enlightening.

Furthermore, I wish to express my gratitude to the DK LogiCS program, which provided me with the opportunity to meet exceptional people in the doctoral college who also became my dear friends.

I would like to extend my sincere appreciation to the reviewers for their time and effort in evaluating this thesis.

Last but certainly not least, I want to convey my deepest appreciation to my family, especially my husband and my son. They have been the light in my life, offering constant support and love. Despite the numerous relocations across continents that I have experienced throughout my life thus far, their presence have become my anchor.

Contents

Chapter 1

Introduction

In the context of safety-critical systems, ensuring their correctness requires thorough examination of the correctness of both software and hardware components. A flawed processor design, for example, could have catastrophic consequences on the system's overall functionality. As a result, it is essential to verify and validate both software and hardware to guarantee their reliability and safety.

To mitigate the risks, the study of formal verification using automated reasoning has become increasingly crucial. Through the application of rigorous machine-checkable formal proofs, we can provide assurance of the safety and correctness of complex, scaled systems. Nonetheless, given the scope and intricacy of these systems, the quest for trustworthy formal verification remains a challenging and ever-evolving pursuit. In the domain of automated reasoning, it is common to utilize verifiers to inspect formal specifications. These verifiers also generate certificates in the form of proofs, which are subsequently checked by independent proof checkers. To further enhance the credibility of the verification process, the proof checkers themselves can be subjected to formal verification. For example, the certificate checkers can be verified within a theorem proving environment, thereby ensuring the correctness of the verification outcomes.

Symbolic model checking [4, 43, 44] is a widely acclaimed formal technique that has proven to be highly effective, especially in the domain of hardware verification and synthesis. At its core, this technique involves using a symbolic model to represent a hardware design under verification, while a specification outlines a temporal logic property that the system must satisfy. The model checker then plays a crucial role in verifying whether the specified property holds true or not in the model. This process ensures the correctness of the system design. Leveraging the power of model checking, industrial engineers are able to build correct and robust systems with specific requirements.

A model checker typically provides a true or false answer to the verification problem, and obtaining certificates from the model checking process has become increasingly necessary to build trust due to the complex nature of model checkers themselves. Similarly in the SAT community, SAT solvers provide certificates that are proofs of unsatisfiability [80], which greatly adds trusts to the verification process. Currently, it is mandatory for solvers to generate certificates in SAT competitions [64]. However, the state-of-the-art research for certifying hardware model checking is limited, highlighting the need for the development of techniques that can generate machine checkable proofs in a generic manner. This remains an important area of focus.

The central objective of this thesis is to establish a standardized certificate format that can be utilized across all model checkers. In essence, this involves designing a certificate format that can be easily understood and implemented by all model checking tools. For example, in hardware model checking competitions [120], it would be beneficial to require all model checkers to provide certificates based on a universal format. This can largely improve the quality of the solvers. The resulting certificates will then be verified by independent proof checkers for assessing their validity.

Modern model checkers are typically multi-engined, and uses a series of preprocessing techniques for performance optimization. Given the substantial differences in the underlying model checking algorithms, while following the same certificate format, the constructions of the certificates for different techniques are expected to vary significantly. As such, an equally important research question is how to effectively generate certificates from model checkers, taking into account the unique characteristics of the algorithms employed. An illustration of this is demonstrated by IC3/PDR [29] and interpolation-based model checking [105], which offer inductive invariants that can be utilized directly as certificates due to their algorithmic properties. Conversely, other techniques like k-induction [124] do not possess such feature. More specifically, temporal induction, also referred to as k-induction, is a powerful model checking technique that operates through two primary stages. Firstly, a bounded model checking check is carried out to verify whether the property holds from initialization for a set number of k time steps. Subsequently, the inductive step is executed to confirm that if the property holds consecutively for k steps, it will continue to hold for the successor states.

This thesis introduces a novel certification framework, and as an example, it is first applied to k-induction. The certificate is in the form of a circuit that features an inductive invariant [137]. Such circuit is called a witness circuit. Initially, the witness circuit defined can be validated through five SAT checks and a one-alternation QBF check. Later on, we demonstrate how to eliminate quantifiers in the certificate, enhancing the certification process to comprise solely of simple SAT formulas. Subsequently, this methodology is applied to word-level model checking using k-induction [25].

Next, this thesis also includes certifying model checking using preprocessing, under the same standardized certificate format. Specifically, we start with temporal decomposition [36], which is among the most commonly used preprocessing techniques. We revisit this technique from a formal standpoint as an improvement over previous work and propose a novel compositional framework for certifying temporal decomposition. The framework utilizes compositionality that results in producing elegant proofs, but still produces a single witness circuit as the certificate for the entire model checking procedure. All certification techniques mentioned above have been implemented in the tool-suite CERTIFAIGER along with its extensions to demonstrate their practicality.

Lastly, more results are presented for certifying phase abstraction [26], another commonly used preprocessing technique, within the scope of this thesis. This line of research involves a thorough reassessment to refine the witness circuit definition, aiming to enable the certification of additional techniques. As a result, a more refined and versatile certification framework is developed, capable of accommodating a broader range of model checking methodologies. Specifically, we revisit generalized phase

abstraction and apply the proposed framework to it. It is also worth mentioning that the work on certification has additionally resulted in interesting benchmarks for both the SAT and QBF communities. Pursuing a similar goal, the thesis includes exploratory work on a certification framework within the context of a temporal epistemic logic.

1.1 Outline and Contributions

This thesis is organized into two distinct parts. In this opening chapter, we provide a concise introduction to the thesis topics and motivations, followed by a clear outline of the thesis structure and contributions. Moving forward to Chapter 2, we delve into a comprehensive overview of the background relevant to the thesis. Chapter 3 is dedicated to discussing and expanding upon the work presented in the second part of the thesis.

The second part of the thesis (Chapter 4-8) consists of four peer-reviewed papers and one unsubmitted paper. Chapter 5 is an extended version. Of all these papers, the author of the thesis is the main author, published under the author name Emily Yu. The papers have been slightly revised to adhere to a standardized and consistent format. In Chapter 6, the content related to optimized ternary simulation is excluded from this thesis. However, they should not have any impact on the main content of the papers. Lastly, we conclude in Chapter 9.

For the rest of this section, we summarize the thesis contributions.

Chapter 4 [137] *Progress in Certifying Hardware Model Checking Results* with Armin Biere and Keijo Heljanko. In Part II of the Proceedings of the 33rd International Conference on Computer Aided Verification (CAV 2021), pages 363–386, Virtual Event.

In this paper we present a novel certification framework for k-induction, and implement it as a complete tool suite. We also perform empirical evaluation on our tool to demonstrate its practicality. The incentive for a generic certificate format was given by A. Biere. The formalization of the certification framework was a result of discussions between E. Yu and A. Biere and was described by E. Yu. The main proofs presented were constructed by E. Yu and described with contributions from A. Biere and K. Heljanko. Furthermore, all techniques in the paper were described and implemented in the tool suite CERTIFAIGER by E. Yu, as well as extending the existing model checker MCAIGER. The experimental evaluation was performed and described by E. Yu. The co-authors further contributed with discussions and proofreading.

Chapter 5 *Stratified Certification for k-Induction* with Nils Froleyks, Armin Biere and Keijo Heljanko. Extended version to [25] published in Proceedings of the 22nd International Conference on Formal Methods in Computer-Aided Design (FMCAD 2022), pages 59–64, Trento, Italy, October 17-21, 2022.

This paper is an incremental contribution extending our work in Chapter 4. We formally present a certification framework for k-induction that eliminates quantifiers thus allowing certification via pure SAT checks. We show our framework can be also lifted to word level. The concept of acyclic reset functions was the results of

a discussion between E. Yu and A. Biere. The formalization of the improved and simplified framework was developed by E. Yu. The techniques and proofs in Paper 5 were developed and described by E. Yu. Furthermore, E. Yu implemented and evaluated the bit-level certification tool CERTIFAIGER++. The word-level tool CERTIFAIGER-WL was implemented by N. Froleyks. The experimental analysis was performed by E. Yu. The co-authors further contributed with discussions and proofreading.

Chapter 6 *Towards Compositional Hardware Model Checking Certification* with Nils Froleyks, Armin Biere and Keijo Heljanko. Submitted.

In this paper we revisit temporal decomposition as an effective preprocessing technique by defining its precise formalism. Based on that, we present a novel certification approach for temporal decomposition. The formalization was the contribution of all authors and was described by E. Yu. All proofs in the paper were developed and constructed by E. Yu. All techniques in the paper were described by E. Yu. E. Yu further contributed to extending both model checkers MCAIGER and AIGTRAV to support reset functions and generate inductive invariants, as well as extending CERTIFAIGER++ for the new certificate. The experimental evaluation was run by N. Froleyks as well as implementing CHMC. The results were analyzed and described by E. Yu. the content related to optimized ternary simulation was the contribution of N. Froleyks thus excluded from this thesis. The co-authors further contributed with discussions and proofreading.

Chapter 7 [139] *A Framework for Model Checking against CTLK Using Quantified Boolean Formulas* with Martina Seidl and Armin Biere. In Proceedings of the 7th International Workshop on on Formal Techniques for Safety-Critical Systems (FTSCS 2019), pages 127–132, Shenzhen, China, November, 9, 2019.

In Paper 7, we present a novel Bounded Model Checking (BMC) tool chain for translating a temporal epistemic logic to quantified boolean formulas. The incentive for implementing the tool was given by E. Yu. The framework was described and implemented by E. Yu with contributions from M. Seidl. The QBF tool for prenexing and cleansing was implemented by M. Seidl. The experimental evaluation was performed by E. Yu. The co-authors further contributed with discussions and proofreading.

Chapter 8. *Certifying Phase Abstraction.*

This chapter contains a proof-of-concept version of unsubmitted work. We present a novel certification framework for phase abstraction where we revisit the simulation relation and show how to generate a certificate for this technique. The results presented were written and constructed by E. Yu. The idea of restricted simulation relation was the result of a discussion between A. Biere and E. Yu. The revisited definitions and semantics of phase abstraction and the certificate framework proposed were developed by E. Yu. Furthermore, N. Froleyks contributed with proofreading the formal proofs.

As mentioned above, the certification work has additionally enabled us to provide SAT and QBF benchmarks. We submitted benchmarks to SAT competition 2022 [138], which was described and generated by E. Yu.

A further journal paper where E. Yu is a co-author is [10]. The topic is related to the master thesis of E. Yu which resulted in a peer-reviewed paper published at the AAAI conference [11], however, the journal paper is on a different logic therefore the contribution is new. In [10] we presented a novel three-valued semantics for the logic ATL* with bounded recall together with an experimental implementation. The tool presented was implemented and evaluated by E. Yu. Results were analyzed and described by E. Yu. However, even though in the same scope of formal verification, the topic is not directly related to our discussions here, thus it is not included in this thesis.

Chapter 2

Background

This chapter serves as a comprehensive introduction to the background of this thesis. The efficacy of modern model checkers heavily depends on the utilization of underlying SAT/SMT solvers for efficient verification. In our certification work, we also leverage these solvers. Hence, we commence this chapter by providing a high-level introduction to SAT and SMT, as well as QBF. Additionally, we present a selection of standard model checking techniques, and highlight the industrial relevance of hardware model checking. Understanding these techniques is crucial for comprehending the context in which our certification work operates. Then to wrap up this chapter, we conclude with a discussion on related work on certification.

2.1 SAT

We start by introducing some basic terminology in SAT.

Propositional logic is the input language of SAT solvers. It is defined over Boolean variables, the Boolean constants (true and false), and the Boolean operators $\{\neg, \wedge, \vee, \Rightarrow, \equiv\}$ expressing negation, conjunction, disjunction, implication, and equivalence respectively. All Boolean formulas can be translated to their conjunctive normal form (CNF) that is equivalent to themselves. A literal is either a positive Boolean variable x or its negation $\neg x$. A clause is a finite disjunction of literals, and a formula in CNF is a conjunction of clauses, which is the most common representation used by modern SAT solvers. We write $vars(\varphi)$ to denote the set of variables occurring in the formula φ. An assignment is a function mapping variables to their truth values, i.e., true ($\top$) or false ($\bot$). An assignment over a formula φ is said to be *total* iff it assigns truth values to all variables in $vars(\varphi)$, otherwise it is called a *partial* assignment (or cube).

Given a Boolean formula φ defined over a set of variables V, the Boolean satisfiability (SAT) problem determines whether there is an assignment to the variables V such that φ evaluates to $\top$. Such an assignment is referred to as a *satisfying assignment*. In the scenario where there exists no satisfying assignment for φ, it is considered *unsatisfiable*; otherwise it is satisfiable. The SAT problem is NP-Complete [68]. Most state-of-the-art SAT solvers are based on Conflict-Driven-Clause-Learning(CDCL) [103], which is the advancement of the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [48, 49].

Nowadays there has been considerable interest in certifying SAT solving to increase trusts in SAT solvers. The SAT solver provides a proof (i.e., the certificate) via some

book-keeping during the analysis for a formula to be unsatisfiable, which is then checked independently by a proof checker [71, 142]. In the case of a satisfiable formula, the certificate is simply the variable assignment. For a more detailed introduction to proofs of SAT solving, refer to Chapter 15 of *the Handbook of Satisfiability* [23].

2.2 QBF

The logic of Quantified Boolean Formulae (QBF) extends propositional logic by introducing quantification over Boolean variables. It is known that the QBF decision problem is PSPACE-complete, therefore harder than the SAT problem.

Let V be a set of Boolean variables. A QBF formula φ in its prenex conjunctive normal form (PCNF) is defined in the form of $Q_1X_1...Q_nX_n.\varphi(V)$ such that $Q_i \in \{\exists, \forall\}$, $X_i \subseteq V$ and $\varphi(V)$ is a propositional formula in CNF over V. As an arbitrary propositional formula can be translated to its CNF using the Tseitin transformation [128], this also applies to QBFs. The format QCIR [67] is a popular prenex non-CNF format in practice. The propositional formula $\varphi(V)$ is called the matrix. Here we assume all variables to be quantified. Note that a propositional formula used in SAT can be seen as a QBF formula implicitly quantified with $\exists$ over all variables.

We write $\varphi_{\{x\to\bot\}}$ to denote the formula obtained after replacing all occurrences of x in φ with $\bot$, similarly $\varphi_{\{x\to\top\}}$ for replacing all occurrences of x with $\top$. The QBF formula $\forall Q.\varphi$ is true iff $\forall Q.\varphi_{\{x\to\top\}}$ is true and $\forall Q.\varphi_{\{x\to\bot\}}$ is also true. The formula $\exists Q.\varphi$ is true iff $\forall Q.\varphi_{\{x\to\top\}}$ is true or $\forall Q.\varphi_{\{x\to\bot\}}$ is true. A formula containing only $\bot$ is false, and $\top$ is true.

2.3 SMT

First-order logic (FOL) extends propositional logic by introducing quantifiers, first-order variables, and predicates used for representing relations and functions. Variables are defined based on certain domains. Satisfiability Modulo Theories (SMT) extends FOL with theories, that focuses on arrays, arithmetic and bit-vectors. In the following we briefly introduce the standard notions following [6, 23, 31]:

- A signature Σ is a set of predicate symbols and function symbols, as well as propositional symbols. Each symbol is associated with a sort and an arity.
- The set of axioms $\mathcal{A}$ are constructed based on Σ. They define the interpretation for all symbols in Σ that are not constant.
- A first-order theory T is defined over Σ and $\mathcal{A}$.
- A Σ-formula is a formula constructed from Σ.

Given a quantifier-free Σ-formula, it is satisfiable in T (or T-satisfiable) if it is satisfied by some assignment of domain variables to their constant, predicate and function symbols, under the intepretation of T.

A fixed-size bit-vector is a sequence of bits with length n. The length n is typically called the bit-width which defines the sort of the bit-vector. In the context of this thesis, we are interested in the theory of quantifier-free fixed-size bit-vectors.

2.4 Model Checking

Model checking [4, 43, 44] concerns the problem of determining whether a model representing a system satisfies a given specification. It is a key technique widely used in the verification of safety-critical systems. In the context of hardware verification, the specification languages of interests are typically temporal logics such as Linear Temporal Logic (LTL) which includes safety properties and liveness properties.

Safety properties are those whose counterexamples are finite traces originating from initial states to some bad state where the property does not hold. Currently, safety properties are the most prominent part in hardware model checking competitions, which is also the main focus of this thesis.

Example 2.1. Consider an example of a 2-bit counter counting modulo three, starting at zero. It increments by one at each step. There is an additional enabler bit which is a user input. The counter resets to zero when the enabler bit is set to one. The bad state is when the counter reaches three.

Fig. 2.1 depicts a state transition diagram of 3 bits consisting of 2 bits for counting and 1 bit of input. Each state can be expressed by an assignment to the Boolean variables $b_0b_1b_2$ where we use b_0b_1 as the counting bits and b_2 is the input. In this example, the property states that the counter does not reach three, expressed as $\neg b_0 \vee \neg b_1$. The initial state is $\neg b_0 \wedge \neg b_1$. A state is considered reachable if there is a trace from the initial state to it following the transition relation. The transitions of the system are specified by the directed arrows. It is encoded by introducing a set of auxiliary variables $\bar{b_0}\bar{b_1}\bar{b_2}$ for representing successor states. A system is considered safe if no bad state is reachable from the initial states. The model checking problem concerns checking whether there is a trace from the initial state (00) to the bad state (11). As we can see from the diagram, the bad state is not reachable, therefore the system is safe. We refer to the handbook of model checking [41] for a general introduction to model checking.

2.4.1 Bounded Model Checking

Bounded model checking, as described in the works of Biere et al. [17, 18, 41], is a powerful technique that utilizes SAT technology. Interestingly, it continues to gain significant popularity in industrial settings compared to other unbounded model checking approaches. The objective of bounded model checking is to determine whether a bad state can be reached within a specified number of system unrollings from reset states.

First of all, we fix some notations:

- V represents the set of state variables.

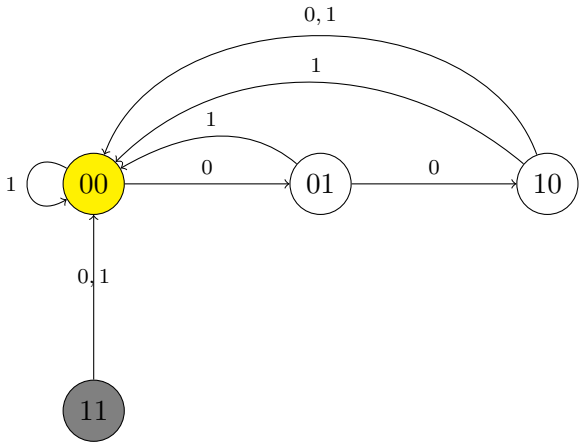

Figure 2.1: State transition diagram of a 2-bit counter. The state (00) coloured yellow is the initial state. The state (11) coloured grey is the bad state. Edges are marked with the values of the input bit.

- $P(V)$ is a temporal logic property that encodes the set of good states.
- $R(V)$ is a Boolean formula that encodes the initial states.
- $T(V, \bar{V})$ represents the transition relation, where $\bar{V}$ is a copy of V representing the next-state variables.
- V_i denotes a copy of V.

The bounded model checking (BMC) problem is essentially the satisfiability problem of the SAT formula:

$$R(V_0) \wedge \bigwedge_{i \in [0,k)} T(V_i, V_{i+1}) \wedge \neg P(V_k).$$

2.4.2 Interpolation

In [105], McMillan proposed a SAT-based model checking technique using Craig interpolation. The core concept is to compute a Craig interpolant, which can be derived from the proof of an unsatisfiable Boolean formula. The interpolant captures the difference between two given formulas I, F, where I is the set of initial states and F the set of final states after a number of transitions. The model checking algorithm constructs a proof of unsatisfiability ϕ of $I \cup F$ such that ϕ can be obtained from a proof system such as resolution or sequent calculus. Bounded model checking is efficiently combined with Craig interpolation. For instance, a bounded checking problem with depth k can be determined to be unsatisfiable, such that no counterexample exists within k steps of unrolling from reset. An interpolant can be then derived from the proof of unsatisfiability representing an over-approximation of the states that are one step from

the initial states. It can be further refined by iterative computation until fixpoint to obtain an over-approximation of reachable states. The depth k is increased each time whenever there is a violation of the property until a true counterexample is found (a path from I to F), or the over-approximation converges.

2.4.3 Temporal Induction

Temporal induction, commonly referred to as k-induction, is a widely used technique in unbounded model checking. It can be formulated into two steps:

1. $R(V_0) \wedge \bigwedge_{i\in[0,k)} T(V_i, V_{i+1}) \wedge \bigvee_{i\in[0,k]} \neg P(V_i)$;
2. $\bigwedge_{i\in[0,k)} T(V_i, V_{i+1}) \wedge \bigwedge_{i\in[0,k]} P(V_i) \wedge \neg P(V_{k+1})$.

For a property P to be considered k-inductive, it is necessary for both formulas above to be unsatisfiable. The first formula is essentially a BMC check with a bound of k, serving to determine if the property is violated within the initial k time frames of the system. The second formula ensures that if the property holds for k consecutive steps, it will continue to hold in the subsequent step. To achieve completeness for this method, an additional uniqueness constraint can be introduced to the second formula, guaranteeing that the k consecutive states are distinct from one another.

2.4.4 IC3/PDR

IC3/PDR [30, 56] is one of the most powerful and robust SAT-based model checking algorithms. The algorithm constructs a sequence of frames $\mathcal{F}_0, ..., \mathcal{F}_n$, where each $\mathcal{F}_i$ is a Boolean formula representing an over-approximation of the set of states reachable in i steps from the initial states. The algorithm starts with an empty formula $\top$ and incrementally extends the frame sequence through iterative computations of relative reachability. During the extension step, it computes backwards from the last frame searching for a trace where a bad state is reachable. It refines the frames by blocking the bad states at the biggest index i possible. The computation continues until either a counterexample is found or a fixpoint is reached such that $\mathcal{F}_n = \mathcal{F}_{n+1}$. In the end, the frame F_n is also an inductive invariant in the system therefore a certificate.

2.4.5 Preprocessing

In addition to the core model checking techniques, preprocessing plays a crucial role in enhancing the overall performance of the verification process. Preprocessing is commonly employed in conjunction with standard model checking approaches, further optimizing their efficiency and effectiveness.

Retiming [90] encompasses the process of shifting latches both forward and backward along the AND-gates in order to minimize the total number of latches. The goal is to optimize the number of registers used for obtaining a smaller circuit. When a latch is

moved backward, it introduces a delay, referred to as the gate's lag, which represents the number of timeframes it is delayed. In peripheral retiming, latches can also be relocated across inputs and properties. An additional variation is normalized retiming, where gates are assigned negative lags.

Temporal decomposition [36] is one of the most fundamental preprocessing techniques, and commonly used in combination with other preprocessing techniques too, such as retiming. The approach focuses on optimizing the model checking problem by finding and removing transient signals that stabilize to constant values after a number of unrollings from reset. Those transient signals can occur due to an initialization sequence in the circuit design and become unused afterwards. Removing them simplifies the circuit, which results in a smaller thus simpler circuit under verification. Typically, ternary simulation is used to quickly find a set of transient signals, which can be lossy, but efficient. Under temporal decomposition, the model checking problem is broken down into two stages: (i) doing a bounded model checking for the initial states of the first n time frames; (ii) checking the safety property of the time-shifted circuit. For the second part, a standard model checker using IC3/PDR or k-induction is used. A more detailed introduction to temporal decomposition can be found in Chapter 6.

Phase abstraction is another popular technique. It was first introduced in [7, 8, 77], where the authors introduced two-phase abstraction in terms of a strict two-phase design style. This can be seen as a more restricted and syntactic version compared to generalized phase abstraction [26], which is more commonly implemented in modern model checkers. The main idea is to simplify the model checking process by identifying and removing oscillating signals that have clock-like behaviours. Generalized phase abstraction operates as the following:

1. Ternary simulation is used to find oscillating latches that have clock-like behaviours, i.e., their transition function results in a periodic pattern.

2. An optimal phase number n can then be computed based on the periodic patterns of the oscillating signals that are found.

3. The circuit gets unfolded n times by making n copies of it. The copies are connected to each other such that the values of the latches of a current copy are inputs for the transition functions of the adjacent (or next) copy.

4. The values of the oscillating signals as well as their copies are fixed to constants with those in the first copy setting to zero.

5. The transformed circuit gets simplified through rewriting as well as cone-of-influence reduction.

In addition to the preprocessing techniques discussed here, modern model checkers employ a wide range of other preprocessing techniques too [65, 90, 91, 93, 109, 110, 144].

2.4.6 Hardware Model Checking Competition

Hardware model checking competitions (HWMCCs) [120] have gained significant popularity in the model checking community over the years. These competitions have played a crucial role in driving advancements and improving the performance of hardware model checkers. The history of these competitions can be traced back to 2006, where they became a central theme at the Computer Aided Verification (CAV) conference. Since 2011, the competitions have been primarily affiliated with FMCAD, which is the main conference for formal verification in the hardware domain. One of the notable contributions of these competitions is the collection of a large number of realistic industrial benchmarks, which are made available to the research community. This has become an important motivation for improving state-of-the-art model checkers to tackle these difficult problems. As a result, the competition has been serving as a great platform facilitating research and development for both industry and academia.

The AIGER format, introduced by Biere et al [22], is a standardized format for representing bit-level circuits in hardware model checking competitions. It is used by all participating model checkers in the competitions. Since 2019, a word-level track has been introduced, where the BTOR2 format [116] is employed. Aside from this, the competition features various tracks, emphasizing different aspects of model checking. Among these, the single property track is the most important one, which is also the focus of this thesis. Participants in the competitions include state-of-the-art model checkers from research groups worldwide, employing a diverse range of model checking algorithms. For example, the industrial-strength model checker ABC [32] has been a dominant force in the bit-level track for several years. In addition to the non-competitive model checkers submitted by the organizers themselves, there have been a great number of competitive model checkers participating such as Pdtrav [34], IIMC [28], TIP [57], AVY [132], and many others [9, 88, 96, 126, 134, 136]. More recently, HWMCC'20 focused exclusively on the word-level track where participants include AVR [70], NuXmv [37], Pono [101] etc.

2.5 Certification

There are different levels of trust in the correctness of a hardware design, which we summarize as the following.

- Basic tesing: to ensure the design meets the basic functionality requirements, we can perform standard tests.
- Model checking: by modeling the design and its specifications using formal methods, we achieve a higher level of trust in the verification results.
- Certificate checking: model checkers themselves can be complex software tools, therefore establishing a certificate format is crucial. Based on this, we can require model checkers to produce certificates for the verified properties. These certificates are then independently checked to validate the verification results.

- Proof checking: the certificate checker utilizes SAT solvers which also produces certificates of unsatisfiability, such as DRUP [130, 131], DRAT [82] and LRAT [47] proofs. Therefore we can use an independent proof checker to validate the certificates produced by the SAT solver.
- Using a trusted certificate checker: to achieve the highest level of trust, a verified certificate checker can be employed, which entails formally verifying the checker itself in a theorem proving environment using an interactive proof assistant such as Isabelle/HOL [117].

The last step of building a verified certificate checker using theorem proving is part of future work. The focus of this thesis is on the third step, which involves the certification of model checking results using an independent certificate checker. The aim is to establish a standardized format for certificates that serve as proofs of the model checking results. Another method for ensuring correctness is to develop certified model checkers that are directly verified by theorem provers [3, 60, 125]. An example of this is [60] where the authors present a verified LTL model checker. However, developing a fully verified model checker is a very heavy task, which can get extremely complicated with optimization techniques. An update in the technique may require the entire construction to be certified again. Furthermore, fully verified tools often exhibit slower performance compared to their non-verified counterparts. Alternatively, by developing a generic certificate format, existing state-of-the-art model checkers can still be utilized to become *certifying*, required to produce a certificate as proof evidence. This redirects the problem of implementing a certified model checker to a relatively simpler tool, a verified certificate checker.

2.6 Related Work

Even though certification of model checking traces back to decades ago, the current state-of-the-art still faces some limitations. The work in [111] includes a deductive proof system for generating proofs for μ-calculus, where they also present experimental results using a BDD-based model checker. More recently, similar approaches have also been extended to SAT-based model checking [72, 73]. In [72, 73], the authors show how to produce deductive proofs for Linear Temporal Logic (LTL). More relevant to the topic of this thesis, in [73] they also incorporate temporal decomposition into their deductive framework. However, their approach mainly relies on the assumption that the underlying model checker produces an inductive invariant as a certificate for the safety property, and builds a proof for liveness properties on top of that. This could only apply to very specific model checking algorithms, and differs significantly from our work. We focus on designing a generic certificate format that is compatible with all model checking techniques and show how to construct certificates based on it. Moreover, the approaches mentioned above use a deductive proof system that requires multiple indpendent parts to be checked, whereas the goal of the thesis is to design a format such that only one witness circuit is checked, instead of a multi-stage proof as done by [72, 73, 111].

Certification has also been discussed in the context of parameterized model checking [46], where the authors propose a framework for certifying an SMT-based parameterized model checker by generating an inductive invariant from backward reachability analysis. Their focus is on a different domain within model checking, which differs from the approach presented here.

As regarding k-induction, several approaches have been proposed to generate proofs for k-induction, leveraging interpolation. In [27] the authors proposed an inductive invariant for k-induction by simply unfolding the circuit k times, meaning encoding explicitly the unrolling of the cicuit. This certificate, however, is exponential to the size of the original model checking problem. Based on this, under the same complexity, the authors in [75] proposed a new model checking algorithm combining k-induction with IC3 where their algorithm also produces a certificate that is an inductive invariant. Another line of work also concerns generating certificates for the k-induction algorithm [107, 133]. In [107], the authors present a method for certifying an SMT-based model checker by extracting and simplifying the k-inductive invariants. Their work differs from the work in this thesis mainly because they focus on generating inductive invariants that are specific for k-induction, while we focus on a generic certificate format. In [107] the authors also show how to simplify the certificates for k-induction through trimming, while not directly applicable to our approach, this draws inspiration for future work on improving the certificates presented in Chapter 5.

In comparison, our k-induction certificate presented in Chapter 4 and 5 is linear to the original circuit size and the inductive depth. Moreover, in contrast to other approaches that break down the certification process into separate steps using a deductive proof system, the work presented in this thesis takes a different approach by focusing on establishing a standardized certificate format in the form of a witness circuit that certifies the entire model checking process.

Chapter 3

Dicussion on Published Work

Chapter 2 includes a summary of a selection of techniques utilized in model checking, some of which can be effectively combined while others exhibit profound disparities. Indeed, modern model checkers employ an extensive array of techniques, often in tandem, to achieve state-of-the-art performance. Because of the intricate and distinctive nature of these algorithms and search heuristics, the certificates can markedly differ in their structure and composition. More importantly, it remains an open research question how to certify some of these techniques.

In this chapter, we engage in a reflective discussion on the work presented in Chapter 4-8, placing it into perspective and analyzing its significance. Furthermore, in addition to reviewing our published work, we introduce some novel extensions too.

3.1 Progress in Certifying Hardware Model Checking Results

In Chapter 4, we made our important first steps towards establishing a standardized certificate format for hardware model checking. Our proposed certificate format employs five simple SAT checks and one QBF check with one alternation. To formally define all components, we use a purely symbolic representation of models, based on circuits.

Our approach allows the extension of a given circuit under verification to a larger circuit, referred to as the extended circuit, which effectively simulates the original circuit while preserving its safety property. A critical concept of this extension is the simulation relation (namely combinational simulation), which involves two SAT checks and one QBF one-alternation check. This contribution is a key highlight of the paper. Moreover, we prove a technical theorem demonstrating that if the extended circuit is verified as safe, then the original circuit is also safe.

By redirecting the problem of finding a direct inductive invariant (as a certificate) in a given circuit, our approach enables the generation of an inductive invariant in a larger circuit. We then need another three SAT checks for verifying the inductive invariant.

The second challenge is determining how to generate such an inductive invariant, which can vary significantly for different model checking techniques. In Chapter 4, we specifically addressed the k-induction algorithm, which is a commonly used method in model checking. We introduce a formal definition of the k-witness circuit, which extends the original circuit and provides an inductive invariant. This contribution constitutes

another critical aspect of our research. We then establish a theorem demonstrating that the k-witness circuit is one-inductive if and only if the original circuit is k-inductive.

We have developed a tool-suite called CERTIFAIGER that implements the theoretical framework. This suite includes several components. First, CERTIFAIGER generates a k-witness circuit based on a given circuit and an inductive depth specified by the model checker MCAIGER. With the generated k-witness circuit, we have implemented our own simulation checker, which calls an underlying SAT solver and a QBF solver. Additionally, we use our inductive invariant checker to verify the inductiveness of the k-witness circuit, which also calls a SAT solver.

Based on the experimental results, our tool has demonstrated its effectiveness in certifying model checking results and has the potential to be adopted in practice.

3.1.1 Combinational Simulation

In Chapter 4, as one of the important novel contributions, we introduce the notion of combinational simulation that includes two SAT checks and one QBF check. This definition allows us to embed a given circuit in a larger circuit.

First of all, we define a purely syntactic definition called combinational extension. Essentially, the extended circuit has the same set of inputs and more latches. In practice, when using the AIGER format, we simply assume the first n latches as the common set of latches. Based on this syntactic definition, we introduce combinational simulation which involves three checks. This definition guarantees that the simulating circuit contains all reachable paths in the original circuit.

The reset check $R(L) \Rightarrow \exists(L' \setminus L)R'(L')$ contains a quantifier because we need to ensure $R'(L')$ is satisfiable whenever there is a satisfying assignment for $R(L)$. As in Chapter 5, with the stratification assumption, we were able to eliminate this quantifier. In fact, the reset check in Chapter 5 ($R(L) \equiv R'(L)$) can be further relaxed to be $R(L) \Rightarrow R'(L)$. Interestingly, this is similar for the transition check. If we want to allow the witness circuit to have more transitions on the common latches, we can replace $F(I, L) \equiv F'(I, L)$ with a one-alternation QBF check $F(I, L) \Rightarrow \exists(L' \setminus L)F'(I, L')$.

Figure 3.1 provides an illustration of the complete certification flow we propose. Throughout Chapter 4-8, we have revised the simulation relation a few times and addressed different model checking techniques. Our aim is to establish a universal certification framework that can be applied across these diverse techniques. In order to achieve this, we require the model checkers to implement the proposed certificate format during the model checking process. While model checkers commonly employ preprocessing techniques alongside base model checking algorithms, our approach, as demonstrated in Chapter 6 and 8, focuses on generating a single certificate in the form of a witness circuit. The generated certificate is then passed on to an independent certificate checker, which plays a crucial role in the certification flow. The certificate checker generates a number of simple SAT checks and makes calls to an underlying SAT solver, such as Kissat, to validate the certificate.

To further enhance the trustworthiness of the verification results, we can introduce

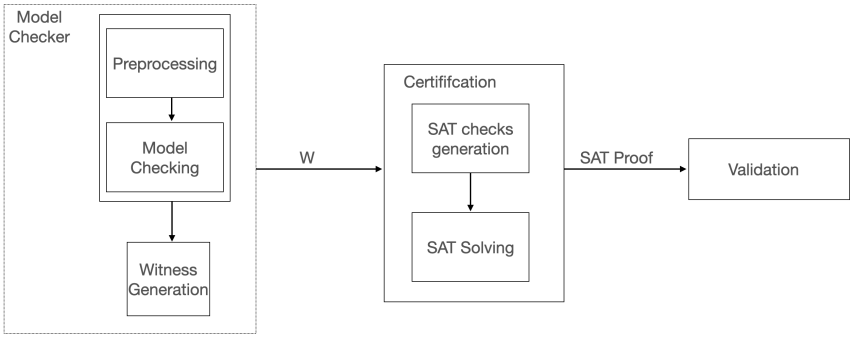

Figure 3.1: The certification flow. The model checker produces a witness circuit (W) which is passed on for validation.

an additional layer of proof generation. By requesting SAT solvers to provide proofs alongside the certificates, we can validate these proofs using independent proof checkers.

3.1.2 Witness circuit

With hindsight, we explored different ways to simplify the k-witness circuit as we later did in Chapter 5. Throughout the process of formulating its formal construction, we devoted considerable time and effort to refining our definition, iterating through numerous versions. Our very first attempt involved constructing a direct inductive invariant from the original circuit. However, we soon realized that this approach necessitated encoding the transition of k successor states in the property, prompting us to explore an alternative solution. The idea of saving multiple states into a single state emerged as an insightful inspiration, which potentially led to simplification of the problem. However, this required extending the original circuit to incorporate state recording functionality.

Along this journey, we encountered a notable challenge that is to obtain the new property as an inductive invariant while satisfying the simulation relation. Our definition of combinational simulation posed the constraint to preserve the same functional definitions for resets and transitions on the common latches. Adhering to this restriction substantially limited the behaviors that the witness circuit could exhibit. This was especially evident later during our attempts to include the uniqueness constraint in our framework. In the end, the reason for the complicated definitions of resets in the k-witness circuit was Theorem 4.14. The inductiveness proof typically involves two parts: the BMC check and the consecution check. When considering Theorem 4.14, with the goal of making manageable proofs, we decided to consider the two checks separately for the circuits. To make the claim as strong as possible, our witness circuit construction satisfies Theorem 4.22. This required us to encode the BMC check of k

steps in C as a one-step BMC check in C', and under the simulation relation, it was not a trivial task.

Both certificates in Chapter 4 and 5 are valid in our framework. We believe that there is a possibility to further simplify these certificates like proof trimming in SAT, which would greatly enhance the efficiency of the certification process.

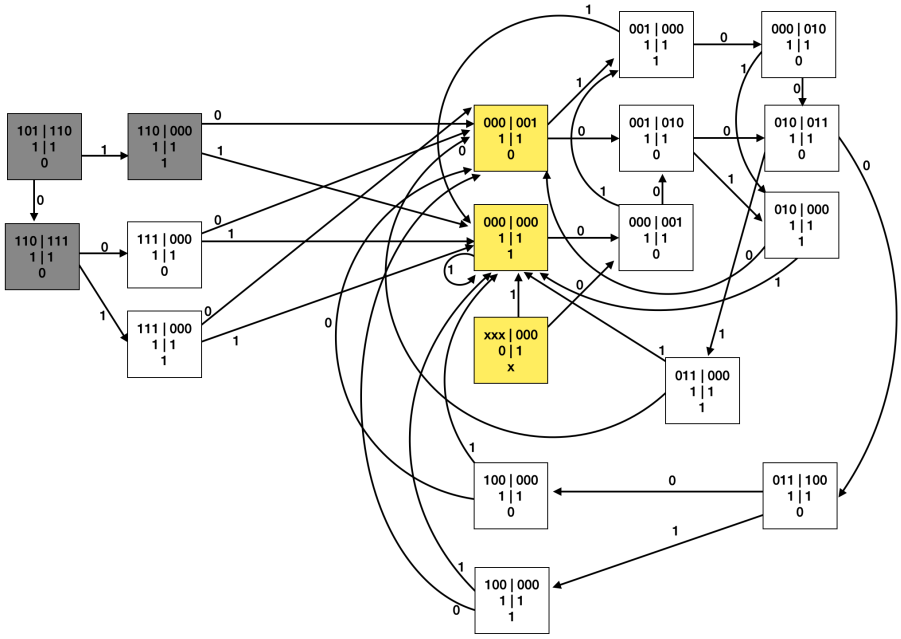

Figure 3.2: The transition diagram of the k-witness circuit of the Counter example. The initial states are coloured yellow. Bad states are marked gray. For simplicity we write “x" to represent either 0 or 1. The edges are marked with input values. Note that only a subset of the state space is depicted.

In Chapter 4, we provide a running example that demonstrates the behavior of a 3-bit counter performing an exact modulo check. The example contains a 2-inductive property. We present the corresponding k-witness circuit in Figure 3.2. This circuit illustrates how the counter progresses and transitions between different states. In this particular example, each state consists of two copies of the original state, accompanied by two initialization bits. This example is 1-inductive, indicating that no bad state is reachable from a good state within one step. Regarding the initialization step, there are two possibilities: full initialization and partial initialization. In the case of full initialization, both initialization bits are set to one, allowing for a complete reset of the counter’s state. In the case of partial initialization, only one copy is initialized.

3.1.3 Simple Paths Constraints

In addition to the standard k-induction technique discussed in Chapter 4, it is common practice to enhance its power with the use of simple path constraints (also known as loop-free constraints or unique state constraints). These constraints limit the paths that are considered to be unique states, and they are known to make k-induction complete. However, this could potentially lead to the inductive depth being equal to the recurrence diameter of the state space [18]. We give a formal definition of k-induction with simple paths constriants below.

Definition 3.1 (k-inductive under SPC)**.** Given a circuit C with a property P, define the formula $S_k = \bigwedge_{i\in[0,k)} P(I_i, L_i)$ and $\text{Simple\_paths}_k = \bigwedge_{i,j\in[0,k),i\neq j} ((I_i \not\simeq I_j) \vee (L_i \not\simeq L_j))$. Then P is called *k-inductive* under simple paths constraints (SPC) in C if and only if the following two conditions hold:

1. $U_{k-1} \wedge R(L_0) \Rightarrow S_k$ "initiation"
2. $U_k \wedge S_k \wedge \text{Simple\_paths}_k \Rightarrow P(I_k, L_k)$. "consecution"

The encoding of simple path constraints involves a quadratic encoding of state comparisons for the inductive depth, which can potentially lead to the generation of large certificates. We have dedicated considerable effort to investigating approaches for dealing with simple path constraints in order to certify it, but unfortunately we have not yet been able to find a satisfactory solution.

One of the main challenges we face is that our certificate format relies on a combinational simulation relation, which necessitates that the simulating circuit has the ability to simulate any permissible path found in the original circuit. However, under simple paths constraints, we only consider a subset of the state space with fewer transition relations. We cannot reconcile the two features with the k-witness circuit construction that involves copying the predecessor states.

Our initial approach was to explore the possibility of restricting the behavior of the k-witness circuit by only saving unique states, rather than all states. However, we found that this resulted in a breakage of the transition property in the inductive invariant. Additionally, directly adding a simple unique state constraint in P' would not work, as it would disrupt the property check in the simulation relation.

3.2 Stratified Certification for k-Induction

In Chapter 5, we build upon the work presented in Chapter 4 where we introduced a robust certification framework that leverages a combination of simple SAT and QBF checks to verify the safety of circuits. Specifically, we focused on the k-induction algorithm and demonstrated how to generate an effective certificate using our framework.

Through the experimental evaluation, however, we observed that the QBF check served as a potential performance bottleneck, which could hinder the overall certification

performance. To address this issue, we embarked on a mission to explore new ways to eliminate the quantifiers and optimize the certification process.

In Chapter 5, we revisited the definition of circuits and their reset functions. We were inspired by the concept of stratification in first-order logic and applied it to the reset functions of circuits. Specifically, we assume that the reset functions are stratified, meaning they are acyclic. In fact, this is a highly reasonable assumption, particularly given the present state of the industry. Cyclic definitions of reset often present significant challenges and can be difficult to manage, hence there has been a significant ongoing effort to address these issues.

In light of the stratification assumption, we undertook a reassessment of the k-witness circuit construction presented in Chapter 4. In the prior study, the reset functions of the k-witness circuit were characterized by a high level of intricacy, given their inherent design complexity to accommodate multiple initialisation ways. However, through our analysis aimed at optimising the certification process, we discovered a viable path to simplify the witness circuit by replacing the QBF check on reset functions with a SAT check and a polynomial-time check.

We integrated the enhanced certification framework into the extended version of CERTIFAIGER, namely CERTIFAIGER++, and conducted a comprehensive performance evaluation in comparison to its predecessor. The experimental results demonstrate a substantial improvement in the certification efficiency, thereby validating the effectiveness of the proposed approach. It is worth noting that while our improved certification framework in CERTIFAIGER++ has enhanced the certification performance, the consecution check still remains a significant bottleneck, consuming a considerable amount of the certification time. As such, it remains an open research question to explore further optimization avenues to improve the certification process.

So far, our focus has been on bit-level model checking. However, the elimination of quantifiers has enabled us to transcend to the context of word-level model checking, where we focus on quantifier-free fixed-size bit-vectors. To this end, we adapted our framework to incorporate finite domain variables. We implemented the word-level certification tool-suite, called CERTIFAIGER-WL, which utilizes the BTOR2 format. Our experiments demonstrate that the framework is very efficient and effective for word-level model checking.

3.2.1 Witness Circuit

As previously discussed, in the context of constructing the witness circuit for k-induction, we made adjustments to the reset definition outlined in Chapter 5. To illustrate this modification, we provide the construction of the witness circuit for the 3-bit Counter example in Figure 3.3. The primary difference compared to Figure 3.2 lies in the initial states. The main difference compared with Fig. 3.2 is the initial states. Definition 6.16 in Chapter 5 only allows partial initialization.

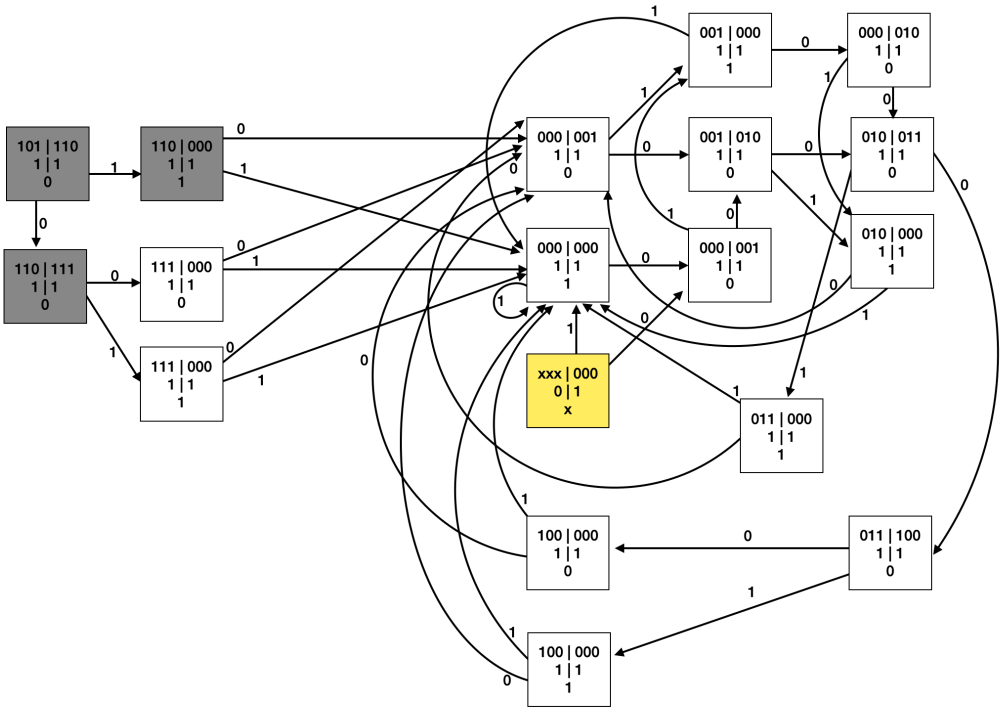

Figure 3.3: The transition diagram of witness circuit of the Counter example. The initial states are coloured yellow. Bad states are marked gray. For simplicity we write “x" to represent either 0 or 1. The edges are marked with input values. Note that only a subset of the state space is depicted.

3.2.2 Circuit Representation

In our early attempt, we considered modelling systems using Kripke structures but ultimately opted for a circuit representation that closely mirrors the AIGER format used in hardware model checking competitions. However, we have always believed that using transition relations instead of transition functions would potentially lead to more QBF checks, but it is still worth investigating. In the following, as an extension to Chapter 5, we present an alternative representation of symbolic transition systems, which involves an increased number of QBF checks.

Definition 3.2. A symbolic transition system $M = (S, I, T, P)$ is defined as follows:

- S is a finite set of Boolean state variables. A state is a total assignment to S.
- I is a formula representing initial states.
- T is a formula over $S \cup S'$ representing transition relations, where S' is a primed copy of S for next states.
- P is a formula over S encoding good states.

One key distinction between the transition system described above and the circuit definition in Chapter 5 lies in the representation of state transitions. In the transition system, the transitions are defined using a transition relation. On the other hand, the circuit definition uses transition functions to capture the behavior of the system.

Definition 3.3. Given $M = (S, I, T, P)$ and $M' = (S', I', T', P')$, where $S \subseteq S'$. M' is said to simulate M iff,

1. $I \Rightarrow \exists(S' \setminus S)I'$.
2. $T \Rightarrow \exists(S' \setminus S)T'$.
3. $P' \Rightarrow P$.

The above definition suggests that verifying the simulation relation using the representation of transition systems requires one SAT check and two one-alternation QBF checks. Compared to the combinational simulation definition in Chapter 4, this requires an additional QBF check instead of a SAT check for the transition relation.

Definition 3.4. The unrolling of a transition system $M = (S, I, T, P)$ of length $m \in \mathbb{N}$ is defined as: $U_m = \bigwedge_{i \in [0,m)} T(S_i, S_{i+1})$.

Same as in Chapter 4-8, in the above definition, we use subscripts to denote temporal copies of state variables.

Definition 3.5. The initialised unrolling of a transition system $M = (S, I, T, P)$ of length $m \in \mathbb{N}$ is defined as: $U_m = I(S_0) \wedge \bigwedge_{i \in [0,m)} T(S_i, S_{i+1})$.

Theorem 3.6. *Given $M = (S, I, T, P)$ and $M' = (S', I', T', P')$ such that M' simulates M. Then if M' is safe, M is also safe.*

Proof. Let us begin by assuming that M' is a safe transition system. Using proof by contradiction, we can assume that there exists a bad state in M that is reachable from the initial states. In other words, there exists an $m \in \mathbb{N}$ such that $I(S_0) \wedge \bigwedge_{i\in[0,m)} T(S_i, S_{i+1}) \wedge \neg P(S_m)$ has a satisfying assignment s.

Since $I(S_0)$ is satisfied, according to Definition 3.3.1, there is an extension of the assignment s satisfies $I'(S'_0)$, where S_0 is a subset of S'_0. Additionally, by Definition 3.3.2, this assignment can be further extended to satisfy $\bigwedge_{i\in[0,m)} T'(S'_i, S'_{i+1})$.

Based on our assumption that M' is a safe transition system, we have $I'(S'_0) \wedge \bigwedge_{i\in[0,m)} T'(S'_i, S'_{i+1})$ satisfied, which implies that $\bigwedge_{i\in[0,m]} P'(S'_i)$. By Definition 3.3.3, we have $P(S_m)$, which contradicts our initial assumption. □

3.2.3 Additional Experiments

In Chapter 4-6, we conducted experiments using benchmarks from HWMCC 2010, which already included a substantial number of industrial benchmarks. To further evaluate the toolkits, we ran more extensive experiments on benchmarks from HWMCC 2019, comprising a total of 317 instances, including both SAT and UNSAT instances. For certification, we are only interested in UNSAT instances.

All experiments were executed in parallel on a cluster of 32 nodes. Each node was equipped with two 8-core Intel Xeon E5-2620 v4 CPUs running at 2.10 GHz (with turbo-mode disabled) and 128 GB of main memory. Furthermore, we allocated 8 instances to each node, with a timeout of 2 hours and a memory limit of 16 GB per instance.

Table 3.1 presents the results comparing the certification performance of the BDD and k-induction configurations. The columns provide the following information:

- Benchmark name;
- the inductive depth k;
- the size of the model (number of gates in thousands, i.e., total number of inputs, latches, and and-gates);
- for the k-induction configuration: the model checking time and certification time (in seconds), and size of the certificate (number of gates in thousands);
- for the BDD configuration: the model checking time and certification time (in seconds), and size of the certificate (number of gates in thousands);

In total, the BDD-based model checker AIGTRAV successfully solved 48 UNSAT instances, of which 44 were certified by CERTIFAIGER++. Similarly, the k-induction based model checker MCAIGER solved 56 instances, with 52 being certified. Interestingly, the two model checkers showed complementary behavior, as their solved instances

had only 14 in common. It is worth noting that while these model checkers may not be as competitive as state-of-the-art model checkers, their simplicity allows for easier certification. Modern model checkers employ complex preprocessing and optimization techniques that make certification more challenging, as it would require certifying the additional techniques used, which is still ongoing research.

The table includes a selection of the most interesting benchmarks, while we do display the average values for each component. A full comparison can be found in the next section where we also include the results for temporal decomposition. In the BDD-based setting, the average ratio of certification time to model checking time is 3.94, while for the k-induction case, it is 1.90. This suggests that the certification overhead for k-induction is relatively smaller compared to the HWMCC'10 benchmarks in Chapter 5. One possible explanation is that the HWMCC 2019 instances were more difficult for MCAIGER, resulting in a smaller certification overhead. At the bottom of the table, we also report the outliers where certificate checking experienced timeouts: 3 for BDD-based certification and 4 for the k-induction-based. Optimizing certificates to speed up the certification process is indeed an interesting future direction.

3.3 Towards Compositional Hardware Model Checking Certification

In our pursuit to further enhance the effectiveness of the generic certificate format proposed in Chapter 4 and 5, we tackle preprocessing techniques in Chapter 6. These preprocessing techniques are commonly utilized in conjunction with base model checking algorithms, such as k-induction, to improve the overall model checking performance. However, with the goal of designing a generic certificate format as proposed previously, we are faced with the challenging task of generating a single certificate for both the preprocessing and base model checking. This is a non-trivial problem.

In Chapter 6, we focus on exploring the powerful technique of temporal decomposition, which is commonly utilized by state-of-the-art model checkers such as ABC [32]. As one of the contributions, we revisit temporal decomposition by defining a precise formalism, which is an improvement over existing theories. In addition, we provide a concrete example of a shift counter in Chapter 6 that showcases the significant benefits of temporal decomposition. Specifically, our example demonstrates that utilizing temporal decomposition can potentially lead to an exponential reduction in computational complexity when compared to using k-induction alone.

To tackle the challenge of constructing a witness circuit for temporal decomposition, our initial approach involved directly building the circuit. Unfortunately, we quickly encountered difficulties as the formal definition became unwieldy and required a multitude of auxiliary definitions for proper construction and especially formal proofs. We therefore opted for a more elegant compositional approach, which breaks the proofs into manageable parts. Given that temporal decomposition unrolls the circuit multiple times from resets at the beginning, we introduce the concept of a forward circuit to define the step-wise unrolling of a circuit. For certification purposes, we demonstrate how to

Table 3.1: Experimental results obtained running CERTIFAIGER++.

	Model		kind			BDD		
	k	#	t_M	t_C	#	t_M	t_C	#
Mean_BDD(44)	-	85	-	-	-	236.08	824.92	387
Mean_kind(52)	19.2	365	158.09	300.98	365	-	-	-
Mean$_\cap$(14)	7.9	85	0.7	37.8	85	25.23	185.74	266
qspiflash_qflexpress_divfive-p059	49	580	8.46	389.49	580	39.98	69.3	312
qspiflash_dualflexpress_divthree-p086	19	245	0.8	75.25	245	301	1646.08	1486
zipversa_composecrc_prf-p20	9	73	0.12	16.34	73	1.75	223.94	577
qspiflash_dualflexpress_divthree-p071	5	60	0.06	9.66	60	3.58	1.29	17
zipversa_composecrc_prf-p21	7	56	0.08	12.94	56	2.48	386.7	700
zipversa_composecrc_prf-p15	7	56	0.08	11.14	56	1.84	268.06	595
qspiflash_dualflexpress_divfive-p018	3	34	0.04	3.56	34	0.02	0.51	4
qspiflash_dualflexpress_divthree-p077	3	34	0.04	3.76	34	1.38	0.83	9
qspiflash_qflexpress_divfive-p023	3	31	0.03	3.14	31	0.94	0.5	4
zipversa_composecrc_prf-p04	2	14	0.03	1.72	14	0.01	0.74	3
qspiflash_dualflexpress_divfive-p142	1	4	0.02	0.51	4	0.01	0.5	4
qspiflash_dualflexpress_divfive-p017	1	4	0.02	0.52	4	0.16	0.58	6
zipversa_composecrc_prf-p11	1	3	0.02	0.72	3	0.01	0.75	3
qspiflash_qflexpress_divfive-p057	1	3	0.02	0.48	3	0.04	0.49	3
zipcpu-pfcache-p16	93	6927	114.33	to	6927	to	to	to
zipcpu-pfcache-p19	35	2588	13.02	to	2588	to	to	to
zipcpu-pfcache-p05	35	2588	13.49	to	2588	to	to	to
dspfilters_fastfir_second-p45	17	853	5985.17	to	853	to	to	to
qspiflash_dualflexpress_divfive-p139	to	to	to	to	to	794.76	to	3984
qspiflash_dualflexpress_divfive-p060	34	443	4.42	269.33	443	682.63	to	8859
qspiflash_qflexpress_divfive-p056	5	55	0.06	6.15	55	290.38	to	4537

construct the witness circuit in a step-wise backward manner. In the end, we obtain a single unified witness circuit for the entire model checking process.

Our approach yields a certificate that conforms to the format proposed in Chapter 5, enabling it to be verified with six simple SAT checks and a polynomial time stratification check. We formally prove that our approach is complete, such that if temporal decomposition and base model checking are successful, a valid certificate is guaranteed be generated. Through experimental evaluation, we have demonstrated the efficacy of our approach, showcasing its ability to efficiently verify certificates for both configurations. Our results show the potential of our approach in providing an effective solution for certifying model checking problems.

3.3.1 Compositional Certification

In Chapter 6, we adopt the same certificate format as used in Chapter 5, which is a critical aspect of our work. Although the construction process itself is compositional, the actual generated certificate is represented as a single circuit. This circuit is then verified using the same six SAT checks stated as in Chapter 5.

Aside from that, the construction of the certificate in Chapter 6 differs significantly from Chapter 4 and 5. This difference arises due to the nature of temporal decomposition, which introduces additional considerations and complexities. As preprocessing techniques are commonly employed alongside base model checking algorithms, it is natural to construct the final witness circuit based on a certificate generated from the base model checking process. This is also confirmed later in Chapter 8 where we tackle a different preprocessing technique phase abstraction.

Temporal decomposition breaks down the model checking problem into two parts. The first part involves a time-shifting of n steps. While revisiting the formal definitions of this technique, it was necessary to introduce auxiliary variables to unroll the circuit, which are referred to as stable variables in Chapter 6. These stable variables play a crucial role in the definition of the forward circuit. The forward circuit represents the resulting circuit after shifting the original circuit by one time step. Consequently, the initial states of the forward circuit are the next states of the initial states in the original circuit. With our circuit definition, in particular reset functions, it is essential to introduce the stable variables as auxiliary variables used to encode the transition relation.

An alternative approach to the use of stable variables is to leverage cone-of-influence analysis. Similar to the methodology employed in Chapter 8, we can identify variables that lie outside the cone-of-influence. These variables do not appear in the transition functions of other latches. By identifying and excluding them, we can reduce the number of latches that need to be copied during circuit forwarding. This alternative definition provides a more efficient way to handle circuit forwarding.

Definition 3.7 (Cone of influence)**.** Given a circuit $C = (I, L, R, F, P)$, the cone-of-influence of P, denoted $coi(P)$ is defined as the smallest set of latches such that:

- $vars(P) \subseteq coi(P)$.

- If $l \in L$, for $l \in coi(P), vars(f_l) \subseteq coi(P)$.

Definition 3.8 (Stable variable)**.** Given a circuit $C = (I, L, R, F, P)$, the set of stable variables is defined as $\Lambda = L \setminus coi(P)$.

3.3.2 Additional Experiments

In the following, we present the experimental results of our approach on the HWMCC'19 benchmarks. Fig. 3.4,3.5 and 3.6 illustrates the comparison of model checking performance across different configurations. Surprisingly, on this particular set of benchmarks, we observe that utilizing temporal decomposition does not yield substantial improvements in overall performance. This observation holds true for both the BDD-based and k-induction configurations. An explanation for this is that in the implementation of CHMC, the transient latches are not removed from the circuit. The decision to retain transient latches in the implementation of CHMC and set their reset and transition functions to constant values (0 or 1) is motivated by the need to maintain latch indices and simplify circuit handling. Doing so simplifies the circuit, however, the circuit size would be much smaller if they are removed. This is left for future work.

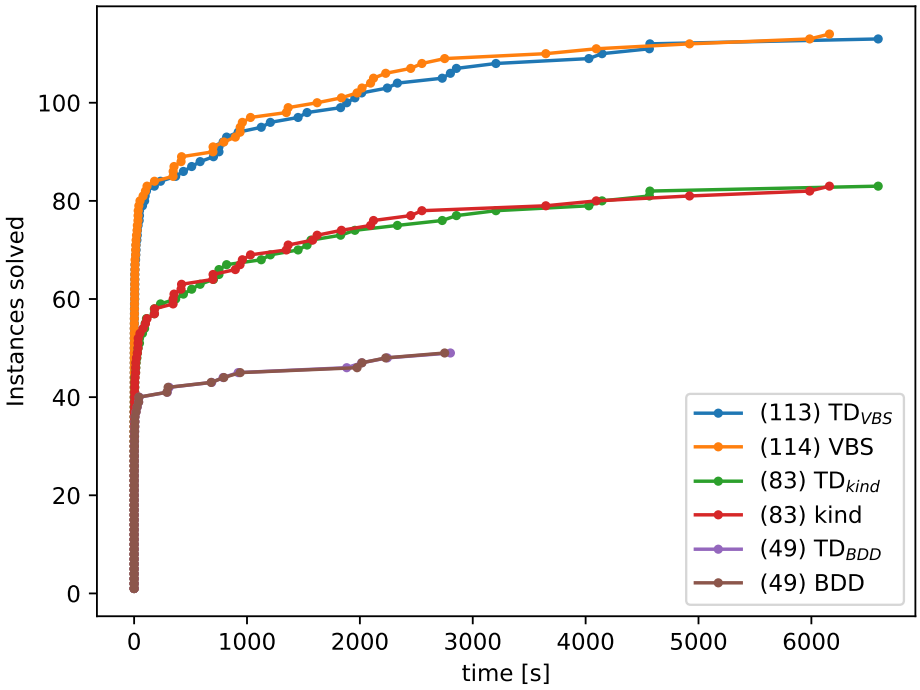

Figure 3.4: Comparison of different configurations on instances from HWMCC'19 including both SAT and UNSAT.

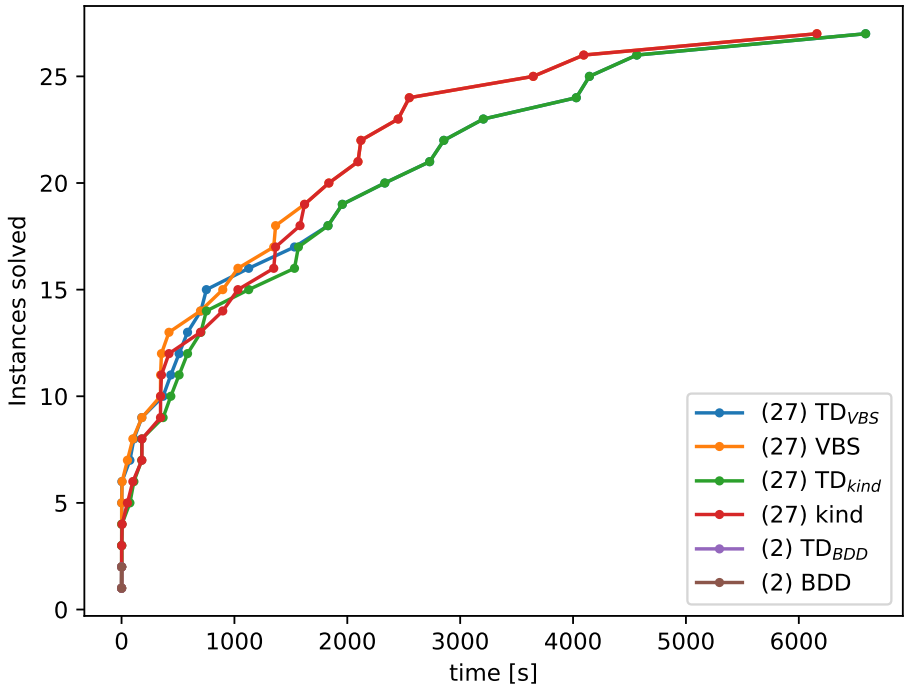

Figure 3.5: Comparison of different configurations on SAT instances from HWMCC'19.

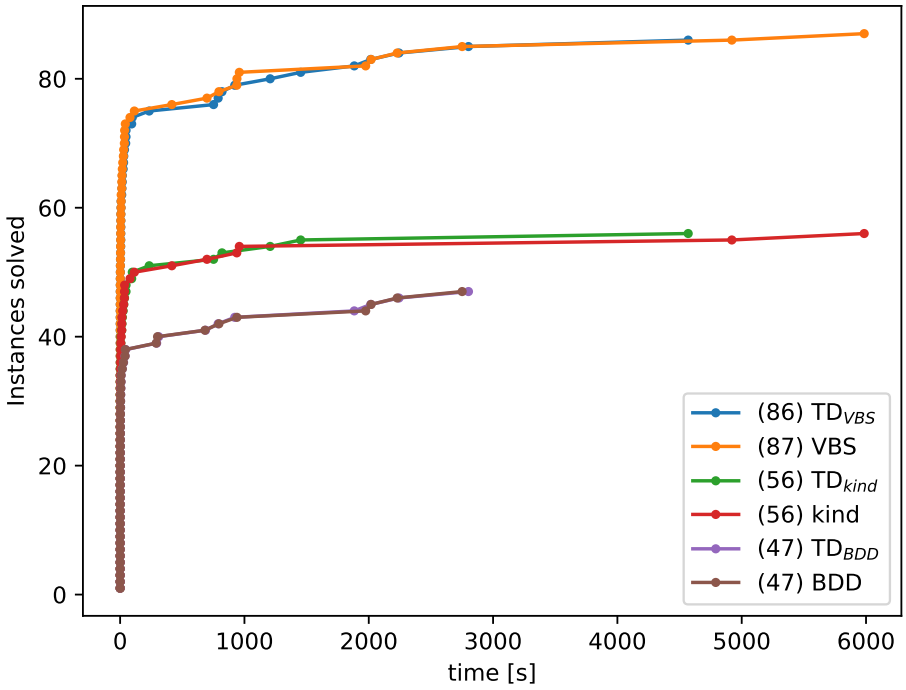

Figure 3.6: Comparison of different configurations on UNSAT instances from HWMCC'19.

Fig. 3.7 presents a comparison of certification and model checking performance on the same set of benchmarks. In the TD_{KIND} configuration, a total of 56 UNSAT instances were successfully solved, out of which 52 instances were certified by CERTIFAIGER++. The average ratio for this configuration is 6.21. Similarly, in the TD_{BDD} configuration, 48 UNSAT instances were solved, with 44 instances certified by CERTIFAIGER++ and an average ratio of 3.32. With a few outliers, these results demonstrate the effectiveness of the certification process in both configurations. This also motivates us to continue to submit more benchmarks to SAT competitions.

Table 3.2 displays more detailed results, including the benchmarks that both configurations solved and the benchmarks that experienced timeouts. The values of unstable duration remain relatively small, which is similar to the results in Chapter 6 as well as the original paper on temporal decomposition [36]. Even though at each circuit-forwarding the active latches and inputs are copied once, this suggests that the cost of time-shifting the circuit in practice is not so high.

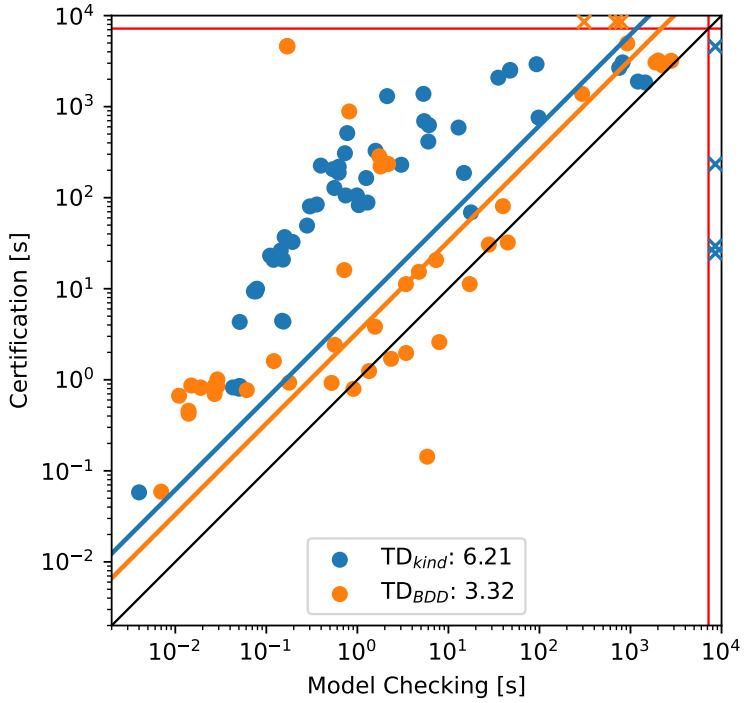

Figure 3.7: Model checking versus certification on instances from HWMCC'19.

Table 3.2: Experimental results obtained running CHMC on HWMCC'19 benchmarks.

	Decomposition				Model		TD_{kind}			TD_{BDD}		
	ω	δ	d	τ	k	#	t_M	t_C	#	t_M	t_C	#
Mean $TD_{BDD}(44)$	1	16.1	1.4	4.4	7.4	7	-	-	-	234.95	780.46	401
Mean $TD_{kind}(52)$	1261.3	17.1	1.2	15.4	18.8	10	89.21	554.28	899	-	-	-
Mean $TD_{\cap}(15)$	1.1	24.1	1.5	6.5	7.4	3	1.02	60.48	218	23.01	147.51	258
qflexpress_divfive-p059	1	25	2	6	49	3	12.92	589.35	1523	39.69	80.7	324
dualflexpress_divthree-p086	1	41	2	9	19	4	1.25	164.28	664	293.76	1383.1	1500
zipversa_composecrc_prf-p20	1	8	1	1	9	3	0.19	32.63	212	1.74	285.32	582
dualflexpress_divthree-p071	1	41	2	9	5	4	0.12	20.78	168	3.43	1.97	32
zipversa_composecrc_prf-p21	1	8	1	1	7	3	0.16	36.92	163	2.18	234.37	705
zipversa_composecrc_prf-p15	1	8	1	1	7	3	0.14	26.27	162	1.8	219.99	600
dualflexpress_divfive-p018	1	41	2	9	3	4	0.08	9.95	97	0.03	0.85	18
dualflexpress_divthree-p077	1	41	2	9	3	4	0.08	9.99	97	1.34	1.25	24
qflexpress_divfive-p023	1	25	2	6	3	3	0.07	9.36	85	0.91	0.8	16
zipversa_composecrc_prf-p04	1	8	1	1	2	3	0.05	4.32	38	0.02	0.82	9
dualflexpress_divfive-p142	1	41	2	9	1	4	0.05	0.86	18	0.03	0.86	18
dualflexpress_divfive-p017	1	41	2	9	1	4	0.05	0.84	18	0.18	0.93	21
qflexpress_divfive-p057	1	25	2	6	1	3	0.05	0.8	15	0.06	0.77	16
zipversa_composecrc_prf-p11	1	8	1	1	1	3	0.04	0.82	9	0.02	0.82	9
vcegar_QF_BV_itc99_b13_p06	2	0	0	21	0	0	0	0.06	0	0.01	0.06	0
zipcpu-pfcache-p16	1	6	1	39	93	23	233.85	to	17401	to	to	to
zipcpu-pfcache-p19	1	6	1	39	35	23	29.42	to	6493	to	to	to
zipcpu-pfcache-p05	1	6	1	39	35	23	24.52	to	6493	to	to	to
VexRiscv-regch0-20-p1	1	8	1	1	22	20	4569.49	to	3241	to	to	to
dualflexpress_divfive-p139	1	41	2	9	to	4	to	to	to	788.52	to	3999
dualflexpress_divfive-p060	1	41	2	9	34	4	6.01	413.77	1199	686.85	to	8873
qflexpress_divfive-p056	1	25	2	6	5	3	0.11	23.09	147	310.3	to	4549

3.4 A Framework for Model Checking against CTLK Using Quantified Boolean Formulas

In Chapter 4-8, we presented techniques for certifying safety properties in hardware model checking. Our work in these papers represented a significant contribution to the field of certification, particularly with regard to establishing a generic certificate format for hardware model checking. In contrast, Chapter 7 was more of an exploratory study, aimed at generating additional benchmarks for certification work.

In Chapter 7, we propose a BMC tool chain for a complete certification flow in the context of QBFs, which includes an interesting translation from a temporal epistemic logic CTLK (Computation Tree Logic with Knowledge) to QBFs based on a previous theoretical work [143]. We used the Interpreted Systems Programming Language (ISPL) to describe a multi-agent system and its specification in CTLK, which was then translated to the QCIR format as input for the QBF solver. Our evaluation of the tool involved conducting a case study on a Train-Gate-Controller, in which we additionally demonstrated the process of encoding an ISPL model to QBFs. In fact, in the field of multi-agent systems, the availability of benchmarks remains quite limited.

To verify the correctness of the verification results, we used the QBF solver to generate a certificate in AIGER format. This certificate contained concrete solutions with assignments to the variables, essentially serving as a witness for the truth value of the formula. The resulting certificate in AIGER is verified by an independent SAT solver. Upon reflection, we recognize that our proposed certification flow can be extended to include the certification of SAT solving as well. Specifically, we can augment our tool chain to generate certificates for SAT solving in addition to QBF solving. This would further enhance the trustworthiness of our verification results.

We presented our tool in Chapter 7 that translates the bounded model checking problem of CTLK specifications to QBFs, which are then solved by a QBF solver. As a promising avenue for future research, we are curious to investigate whether the certification framework proposed in Chapter 4-8 can be extended to QBF-based model checking beyond bounded model checking. This would further enhance the trustworthiness and reliability of QBF-based model checking, paving the way for even more efficient and effective verification of complex systems.

In Chapter 7 we presented our tool $\mathsf{MCMAS}_{\mathsf{qbf}}$ which translates a CTLK formula together with a system description in ISPL into QBF formulas, based on the bounded semantics of CTLK defined in [143]. The models used are interpreted systems, which are compatible with ISPL.

Looking back at Chapter 7, it would be interesting to have a direct comparison of QBF solving with a bounded model checker. This would however require encoding both ISPL models and CTLK formulas into SAT formulas, which can be an interesting direction for future work. Regarding related work, the authors of [119] presented a BMC encoding for the existential fragment of CTLK and later published experimental results for it in [97]. Our tool covers the full fragment of CTLK.

3.5 Certifying Phase Abstraction

In Chapter 8, we continued the work on hardware model checking certification by focusing on generalized phase abstraction. Given that generalized phase abstraction involves a series of transformations on the circuit under verification, we have chosen to embrace a compositional approach, as outlined in Chapter 6.

Regarding the certificate format, we noticed that the previously defined stratified simulation relation was not sufficient to accommodate cone-of-influence reduction used in phase abstraction, therefore had to refine it further. As a result, we define the notion of restricted simulation relation which allows a smaller witness circuit. This new certificate format allows us to certify more techniques.

As stated in Proposition 8.5, the witness circuit constructed in Chapter 5is still a valid witness circuit according to our new definition of restricted simulation. However, as for temporal decomposition, we proposed a compositional framework in Chapter 6 that builds a certificate iteratively, this requires us to revisit the constructions and proofs. We leave this to future work.

We also revisited phase abstraction using cube semantics. Similar to temporal decomposition, phase abstraction also utilizes ternary simulation, however, for a different purpose. For this technique, ternary simulation is used to quickly identify oscillating signals. A third popular usage of ternary simulation is to identify equivalent gates, which is implicitly included in the cube simulation definition and we plan to incorporate in CHMC too. Based on this, we present the certificate construction for phase abstraction including based model checking.

During the design process of the restricted simulation relation, our initial approach involved introducing a new attribute in the circuit definition called oracles resulting in a new model $C = (I, L, R, F, P, O)$. These oracles were intended to have arbitrary next-state values, meaning they did not possess a specific transition function. This presented a challenge as it required distinguishing oracles from normal latches that do have transition functions. Additionally, we need to extend the AIGER format, as currently it does not support latches without transition functions. Furthermore, from a practical implementation standpoint, incorporating oracles in the simulation checker posed additional difficulties.

Chapter 4

Progress in Certifying Hardware Model Checking Results

Published In Part II of the Proceedings of the 33rd International Conference on Computer Aided Verification (CAV 2021), pages 363–386, Virtual Event, 2021.

Authors Emily Yu, Armin Biere and Keijo Heljanko.

Acknowledgement This work is supported by the Austrian Science Fund (FWF) under the project W1255-N23, the LIT AI Lab funded by the State of Upper Austria, and Academy of Finland under the project 336092.

Abstract We present a formal framework to certify k-induction-based model checking results. The key idea is the notion of a k-witness circuit which simulates the given circuit and has a simple inductive invariant serving as proof certificate. Our approach allows to check proofs with an independent proof checker by reducing the certification problem to pure SAT checks and checking a simple QBF with one quantifier alternation. We also present CERTIFAIGER, the resulting certification toolkit, and evaluate it on instances from the hardware model checking competition. Our experiments show the practical use of our certification method.

4.1 Introduction

In many verification applications, k-induction [124] (also known as temporal induction) is used as a powerful technique that reduces model checking to a series of SAT problems. It has been extensively investigated as an effective approach for unbounded model checking [58, 74]. As a generalisation of simple induction, for a given safety property, the k-induction method concerns a base case and an inductive case: the base case is a bounded model checking problem with a depth of k; the inductive case assumes the property holds for k consecutive steps, then checks it also holds for $k + 1$ steps. The safety property is said to be k-inductive if both conditions are satisfied. The nature of the k-induction algorithm allows it to be integrated with modern SAT/SMT solvers. For example, reduction techniques such as preprocessing have been investigated with k-induction in an incremental setting [55]. The present state-of-the-art also concerns combining k-induction with existing SAT-based model checking (SMC) techniques including interpolation and property directed reachability [76, 89]. Furthermore, k-induction has also been extended to the context of infinite-state systems [39, 50, 66, 86], as well as software verification [53]. Another variant of this line of research is the use of k-induction in sequential equivalence checking [108].

Model checking has been an effective technique for the verification of safety-critical systems. In particular, applications deployed in industrial settings such as nuclear facilities, increasingly utilise model checking to gain trust in the correctness of their designs [69, 95, 133]. In such ultra safety-critical applications the certification that the model checking results are in fact correct is crucial. We argue that in model checking generic machine checkable certification is still in its infancy in contrast to related fields. For instance in SAT competitions [5, 79], certifiable proofs are mandatory. This has helped to improve the trust we have in SAT solving results as well as the quality of SAT solvers tremendously.

Even though counterexample validation is commonly used in model checking to certify negative verification results through simulation, producing a generic machine checkable proof on success is less straight-forward. To mitigate this problem, certification of model checking has been suggested earlier in [46, 72, 76, 92, 111, 133, 140], but the methods presented in these works are either not directly applicable to k-induction (in its vanilla form), produce k-induction specific certificates (fail to provide an inductive invariant), or are considered to have exponential certificates. This apparently made it hard to, e.g., require all model checkers to produce proofs in the hardware model checking competitions.

As symbolic model checking of bit-level properties for hardware circuits is PSPACE-complete, we introduce in this paper a novel certification framework for k-induction-based model checking. Our proposed approach generates a fixed number of SAT problems together with a one-alternation only QBF, which are verified by an independent certifier, thereby enabling the certification of k-induction proofs at lower complexity. Our method efficiently extends the given model checking problem to finding a simple inductive invariant of a larger circuit as a proof of k-induction of the original circuit. In particular, the certificate size (as a circuit) is shown to be linear in size of the given

model, and the inductive depth. We present CERTIFAIGER, which works as a complete tool suite for certification, independent of any model checker. Experimental results show that our technique works efficiently and can be adapted for practical use.

The rest of the paper is organised as follows: In Section 4.2 we introduce the notion of combinational simulation in the context of circuits. In Section 4.3, we study the formal property of combinational simulation and define k-induction-based model checking with an example. In Section 6.5, we present our proposed certification approach followed by theoretical results in terms of k-induction. We describe the implementation of our tool suite in Section 4.5, and report on experimental results in Section 4.6. Finally, we conclude in Section 6.7.

4.2 Circuits

In this section, we present a slightly non-standard notation to formalize systems. It allows us to represent systems and particularly circuits symbolically in a compact way and is crucial to reduce notational clutter in the following.

Let $\mathbb{B}(V)$ be the set of Boolean expressions (propositional formulas) over the Boolean variables V. We also write $\mathbb{B}(I, L)$ to denote the set of Boolean expressions over $I \cup L$, where I and L are two sets of Boolean variables. Given two Boolean expressions $f(V)$, $g(V) \in \mathbb{B}(V)$ we call them *equivalent*, written $f(V) \equiv g(V)$, if they have the same models. This notation is also applied to Boolean expressions over different sets of variables by simply interpreting them over the union of their variables. We use " $\simeq$ " for syntactic equivalence [51], " $\rightarrow$ " for syntactic implication, and " $\Rightarrow$ " for semantic implication. To define semantical concepts or abbreviations we stick to equality " $=$ ".

In the context of this paper, models are expressed in the form of finite logical circuits, where states can be seen as truth assignments to latches and inputs. Initial states are defined by the reset values of latches, in our case, represented by their reset functions. For each latch l in L, there is a reset function $r_l(L)$ which is a formula (Boolean expression) over a set of latches L, thus allowing cyclic definitions. Note that a cyclic definition can lead to unsatisfiable reset formulas, in which case there are simply no initial states. Additionally, for some $L'' \subseteq L$, we define $R(L'') = \bigwedge_{l \in L''} l \simeq r_l(L)$ to allow us to analyse reset functions of individual subsets of latches. The transition relation is expressed as a "next state" formula associated with each latch, whereas non-determinism comes from inputs (which act as the environment). The successor value of each latch is defined by applying its transition function on the current values of latches and inputs. Intuitively, a safety property specifies that the system must not violate certain behaviours, i.e., only "good states" are reachable. In this paper we focus on such simple safety properties and leave liveness properties (see e.g., [92]) etc. for future work.

Definition 4.1 (Circuit)**.** A circuit $C = (I, L, R, F, P)$ is defined as follows:

1. I: the set of Boolean *input* variables.
2. L: the set of Boolean *latch* variables.

3. $R = \{r_l(L) \mid l \in L\}$ is a set of *reset function* formulas.

4. $F = \{f_l(I, L) \mid l \in L\}$ is a set of *transition function* formulas, such that for every latch $l \in L$, there is a transition function formula $f_l(I, L) \in \mathbb{B}(I, L)$.

5. $P(I, L) \in \mathbb{B}(I, L)$ is a formula encoding the *(good states) property*.

The reset functions characterise the initialisation of the circuit. Such definition of reset abstracts the way how circuits are reset. As a short-hand we use $L' \simeq F(I, L)$ to denote a conjunction of the corresponding equivalences, i.e., it is interpreted as $\bigwedge_{l \in L'} l' \simeq f_l(I, L)$. For clarity, we use **subscripts** as in L_i to denote a copy of the latch variables L in the **temporal direction** at some timestamp i, where L_0 is the set of latches at timestamp 0 when the circuit is supposed to be initialised. Note that, using such transition *functions* to describe transition relations implies that there will always be a successor state. The temporal evolution of a system is expressed using the notion of *unrolling*, which has a specific length and follows the transition relation at each step.

Definition 4.2 (Unrolling)**.** For an unrolling depth $m \in \mathbb{N}$, the *unrolling* of a circuit C of length m is defined as the formula $U_m = \bigwedge_{i \in [0,m)} (L_{i+1} \simeq F(I_i, L_i))$.

Note that in this definition, we use I_i and L_i as sets of variables, whereas U_m is a formula. For $m = 0$, the conjunction is empty thus the formula is trivial.

Definition 4.3 (Initialised unrolling)**.** An *initialised unrolling* of a circuit C, with $C = (I, L, R, F, P)$, is defined as $U_m \wedge R(L_0)$, where U_m is an unrolling.

We say an unrolling is *safe* if and only if the property holds at every timestamp along the whole length of the unrolling.

Definition 4.4 (Safe unrolling)**.** Unrolling U_m of a circuit $C = (I, L, R, F, P)$ is said to be *safe* if

$$U_m \Rightarrow \bigwedge_{i \in [0,m]} P(I_i, L_i).$$

Definition 4.5 (Safe initialised unrolling)**.** An initialised unrolling $U_m \wedge R(L_0)$ of a circuit $C = (I, L, R, F, P)$ is said to be *safe* if

$$U_m \wedge R(L_0) \Rightarrow \bigwedge_{i \in [0,m]} P(I_i, L_i).$$

We are now ready to introduce the notion of a *combinational extension* between two circuits. It is purely syntactic based on sharing inputs and latches.

Definition 4.6 (Combinational extension)**.** Given circuits $C = (I, L, R, F, P)$ and $C' = (I', L', R', F', P')$, C' *combinationally extends* C if $I = I'$ and $L \subseteq L'$.

As noticed above, this definition allows us to interpret the inputs and latches of a circuit as being part of another circuit. In practice for instance we simply assume that the first $|L|$ latches of the circuit C' are mapped to those of C assuming some ordering of the latches, as it is for instance the case in the AIGER format [22] used in the Hardware Model Checking Competition (HWMCC) [24].

To tackle the problem of generating a proof certificate for k-induction of the safety of a circuit C, as is the main goal of this paper, we extend it to a larger circuit C' with additional "book-keeping" behaviours [1] for which we can show the same property by using standard induction. To ensure that the resulting extended circuit C' preserves the original property, we provide a formalization through a *combinational simulation* relation between two circuits, which needs to be formally verified by a certifier. One important aspect of our design principles is to keep the complexity of the required certification procedure low, in other words, to be done via pure SAT solver checks or by solving a QBF with at most one quantifier alternation. This leads to a more complicated non-standard design of the certification approach, the details of which will be described in Section 6.5.

From a practical perspective, under *combinational simulation* defined below in Def. 4.7, we require that the transition functions on the "common" parts of the two circuits are equivalent. For the new latches, the transition functions are always satisfiable (as they are functions), and thus we need no constraints on them. As second condition we require that if the safety property P' holds in the extended circuit, then the property P holds in the original circuit. The last condition we need to check is that all the new latches of the extended circuit can be initialised with some values whenever the original circuit can be initialised and using the same values for initialising the common latches. In other words, for all initialisations of the original circuit there is at least one initialisation of the extended circuit with the same values for common latches.

Under these conditions Theorem 4.9 in Section 4.3 shows that if the extended circuit (in this sense) combinationally simulates the original one and the extended circuit is safe then the original circuit is safe as well.

With some abuse of notation, we use $\exists L$ in a Quantified Boolean Formula (QBF) to denote existential quantification over variables in L. As usual, free variables are (implicitly) assumed to be quantified universally.

Definition 4.7 (Combinational simulation)**.** Given circuits $C = (I, L, R, F, P)$ and $C' = (I', L', R', F', P')$ where C' combinationally extends C, we say that C' *combinationally simulates* C, if the following holds:

1. $f_l(I, L) \equiv f'_l(I, L')$ for $l \in L$, "transition"
2. $P'(I, L') \Rightarrow P(I, L)$, and "property"
3. $R(L) \Rightarrow \exists (L' \backslash L) R'(L')$. "reset"

In later context when verifying the combinational simulation relation between two circuits, we refer to Def. 4.7.2 as the *transition check*, Def. 4.7.3 as the *property check*, and Def. 4.7.1 as the *reset check*.

4.3 Model Checking

In this section, we consider model checking via k-induction. The model checking problem for safety properties concerns determining whether, given a circuit with a property P, it is the case that P holds in all reachable states, *i.e.,* the initialised unrolling of a circuit of any arbitrary length is safe.

Definition 4.8 (Safe circuit)**.** Let U_m be the unrolling of circuit C, C is safe iff $U_m \wedge R(L_0) \Rightarrow \bigwedge_{i\in[0,m]} P(I_i, L_i)$ holds for all $m \in \mathbb{N}$.

Based on the above definition, we say the property P "holds" in C if the circuit is safe with respect to P.

Theorem 4.9. *Assume that the circuit C' combinationally simulates the circuit C. If C' is safe, then C is safe.*

Proof. We do a proof by contradiction. Let $m \in \mathbb{N}$ be a bound for which the claim does not hold. Thus the unrolling of length m of C' is safe w.r.t. P, and therefore $U'_m \wedge R'(L'_0) \Rightarrow \bigwedge_{i\in[0,m]} P'(I'_i, L'_i)$ holds. To obtain the contradiction we assume there is a satisfying assignment s of $U_m \wedge R(L_0) \wedge \neg \bigwedge_{i\in[0,m]} P(I_i, L_i)$, which would make C not to be safe. Thus $R(L_0)$ needs to be satisfiable. Now the reset check of Def. 4.7.1 implies that $R'(L'_0) \wedge R(L_0)$ is guaranteed to be satisfiable with L_0 being a subset of L'_0. Moreover, by Def. 4.7.2, the unrolling U'_m of C' is also satisfiable with the transition function F applied on the projected ("common") component on both circuits. Also for the new latches the fact that we use a transition function for them, they are also satisfiable (transition functions guarantee that there is always a successor state for all states). Therefore the initialised unrolling $R'(L'_0) \wedge U'_m$ is satisfiable. Furthermore, by our assumption, $\bigwedge_{i\in[0,m]} P'(I'_i, L'_i)$ holds. By Def. 4.7.2 and Def. 4.7.1, the projected latches of C' stay the same as L_i for all $i \in [0, m]$, and thus by Def. 4.7.3 we have that $\bigwedge_{i\in[0,m]} P(I_i, L_i)$ holds. □

As usual, we call a formula ϕ to be an *inductive invariant* ϕ of a circuit C if ϕ satisfies the following conditions: (1) $R(L) \Rightarrow \phi(I, L)$, (2) $\phi(I, L) \Rightarrow P(I, L)$, and (3) $U_1 \wedge \phi(I_0, L_0) \Rightarrow \phi(I_1, L_1)$. As a generalisation, k-induction looks at k steps of evolution rather than 1 step by assuming the property holds in k consecutive timestamps at the induction step.

Definition 4.10 (k-inductive)**.** Given a circuit C with a property P, define the formula $S_k = \bigwedge_{i\in[0,k)} P(I_i, L_i)$. Then P is called *k-inductive* in C if and only if the following two conditions hold:

1. $U_{k-1} \wedge R(L_0) \Rightarrow S_k$, and "initiation"

```
1   MODULE main
2   VAR
3       r : boolean;
4       c0 : boolean;
5       c1 : boolean;
6       c2 : boolean;
7   DEFINE
8       a0 := TRUE;
9       a1 := c0 & a0;
10      a2 := c1 & a1;
11      m0 := (c0 = FALSE);
12      m1 := (c1 = FALSE) & m0;
13      m2 := (c2 = TRUE) & m1;
14      b0 := (c0 = FALSE);
15      b1 := (c1 = TRUE) & b0;
16      b2 := (c2 = TRUE) & b1;
17  ASSIGN
18      init(c1) := FALSE;
19      init(c0) := FALSE;
20      init(c1) := FALSE;
21      next(c0) := !r & !m2 & (c0 != a0);
22      next(c1) := !r & !m2 & (c1 != a1);
23      next(c2) := !r & !m2 & (c2 != a2);
24  SPEC
25      AG !b2
```

Figure 4.1: The SMV code for the *Counter* example.

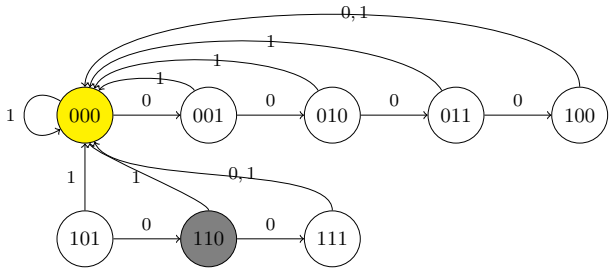

Figure 4.2: The transition diagram of the *Counter* example. The initial state is "000" (colored yellow). In the (gray) "bad" State "110" the property does not hold.

2. $U_k \wedge S_k \Rightarrow P(I_k, L_k)$. "consecution"

The first condition Def. 6.4.1 in this definition is called *initiation check*, also *bounded model checking check* or simply BMC check on the initialised unrolling of length $k-1$, whereas the second condition Def. 6.4.2 is referred to as the *consecution check* for the unrolling of k. Note that a 1-inductive invariant is equivalent to an inductive invariant when $\phi(I, L) \equiv P(I, L)$.

Example 4.11. We consider a simple example of an N-bit counter, where the counter counts up to a *modulo* bound m, then it resets to zero. There is also a *reset* signal which works as an enabler, such that when the signal is set to 1, the counter is forced to reset. The property checks whether the counter value reaches b. Here the exact modulo check makes the model checking problem k-inductive ($k = b - m + 1$). More precisely, for $N = 3$, the formal description of a 3-bit counter is given in the SMV language in Fig. 4.1, where $m = 5, b = 6$. (Note that our example can be easily extended to integers too.) The state diagram of this system is shown in Fig. 5.3. The input values are specified with the transition relations. This model is 2-inductive.

4.4 Certification

In our suggested approach, certifying model checking results concerns finding and checking an inductive invariant which implies the original specification, which in our case, is the safety property P. To tackle the problem of certifying k-induction-based model checking for any given circuit, in this section, we redirect the problem to generating a simple inductive invariant from a *k-witness circuit*, in which the original circuit is combinationally simulated.

We start by defining the formalism of a k-witness circuit. The main idea is to record the previous $k-1$ states and inputs of the circuit observed during the execution, "flattening" the k-induction procedure back to normal induction of a larger circuit. As a result, the size of the circuit increases by a factor of k, where k is the constant used in the k-induction scheme. The k-witness circuit has k local components of inputs and latches. Each component can be seen as representing a state in the original circuit. Whenever a new state is saved, the oldest one is discarded.

One of the key technical challenges is the proper initialisation of the k-witness circuit. We use an additional k initialisation bits for indicating which components of the circuit have been initialised. This helps accomplishing the combinational simulation relation later. We say a component is initialised if its initialisation bit is $\top$. At initialisation, the k-witness circuit can be either *fully* or *partially* initialised. Fig. 4.3 displays three ways of initialising the components. In the case of full initialisation, the circuit pre-computes k steps of the original circuit as the initial state of the k-witness circuit. Thus intuitively in the full initialisation case the initial state of the k-witness circuit encodes the states reachable in the k-step initialised BMC unrolling of the original circuit. In partial initialisation scenarios the circuit instead pre-computes an initialised BMC unrolling for fewer steps, where some components are left uninitialised. In the final case where there are no pre-computed steps, the circuit simply runs from an original initial state, leaving all the other components fully uninitialised.

In the definitions below, we use the **superscript** of i in L^i to denote a copy of latches L in the **spacial direction**, such that we introduce a set of new latch variables for every L^i, where $l^i \in L^i$ is the corresponding copy of $l \in L$, and similarly for inputs. We refer to l^i as some latch in L^i, where i is the index of a latch set L^i. The formal definition of k-witness circuit is given below. We continue to use **subscripts** for the **temporal direction**.

Definition 4.12 (k-witness circuit)**.** Given a circuit $C = (I, L, R, F, P)$, and $k \in \mathbb{N}^+$, the k-witness circuit $\boxed{C' = (I', L', R', F', P')}$ of C is defined as follows:

1. $\boxed{I'} = I$. For simplicity we also refer to I' as X^{k-1}.
2. $\boxed{L'} = X^0 \cup \cdots \cup X^{k-2} \cup L^0 \cup \cdots \cup L^{k-1} \cup B$, such that,
   a) X^i is a copy of the original inputs, for all $i \in [0, k-2]$.
   b) L^i is a copy of the original latches, for all $i \in [0, k-1]$.
   c) $B = \{b^0, \ldots, b^{k-1}\}$ is the set of initialisation bits.

3. The reset function $\boxed{R'} = \{r'_l(L') \mid l \in L'\}$ is defined as follows:
   a) For $x \in X^0 \cup \cdots \cup X^{k-2}, r'_x = x$.
   b) For $i \in [1, k-1)$, $u^i = R(L^i) \vee u^{i+1}$, and $u^{k-1} = R(L^{k-1})$.
   c) For $l \in L^0$, $r'_l = \mathsf{ite}(u^1, l, r_l(L^0))$.
   d) For $i \in [1, k)$, $r'_{l^i} = \mathsf{ite}(u^i, l^i, f_{l^i}(X^{i-1}, L^{i-1}))$.
   e) $r'_{b^{k-1}} = \top$.
   f) $r'_{b^0} = \neg u^1$.
   g) For $i \in [1, k-1)$, $r'_{b^i} = b^{i-1} \vee (R(L^i) \wedge \neg u^{i+1})$.

4. $\boxed{F'} = \{f'_l(I', L') \mid l \in L'\}$ is defined as follows:
   a) For $i \in [0, k-1)$, $f'_{x^i}(I', L') = x^{i+1}$.
   b) For $l \in L^{k-1}$, $f'_l(I', L') = f_l(X^{k-1}, L^{k-1})$.
   c) For $i \in [0, k-1)$, $f'_{l^i}(I', L') = l^{i+1}$.
   d) For $i \in [0, k-1)$, $f'_{b^i}(I', L') = b^{i+1}$, and $f'_{b^{k-1}}(I', L') = b^{k-1}$.

5. The property $\boxed{P'}$ is defined as $P'(I', L') = \bigwedge_{i \in [0,4]} p_i(I', L')$ such that:
   a) For $i \in [0, k-1)$, $h^i = (L^{i+1} \simeq F(X^i, L^i))$.
   b) $p_0(I', L') = \bigwedge_{i \in [0,k-1)} (b^i \to b^{i+1})$.
   c) $p_1(I', L') = \bigwedge_{i \in [0,k-1)} (b^i \to h^i)$.
   d) $p_2(I', L') = \bigwedge_{i \in [0,k)} (b^i \to P(X^i, L^i))$.
   e) $p_3(I', L') = \bigwedge_{i \in [1,k)} ((\neg b^{i-1} \wedge b^i) \to R(L^i))$.
   f) $p_4(I', L') = b^{k-1}$.

In Def. 4.12 we list five parts of the k-witness circuit. For clarity, we explain each part in more details in the following text:

1. The set of **inputs** is identical to that of the original circuit.

2. The set of **latches** consists of the original latches, k initialisation bits, and an additional $k-1$ copies of inputs and latches which are introduced to save observations of previous states.

3. The **reset** function is defined to allow non-deterministic initialisation (see Fig. 4.3), where we use helper variables u^i for a more compact encoding. The formula u^i is satisfied whenever a component younger than the ith has the same reset value as

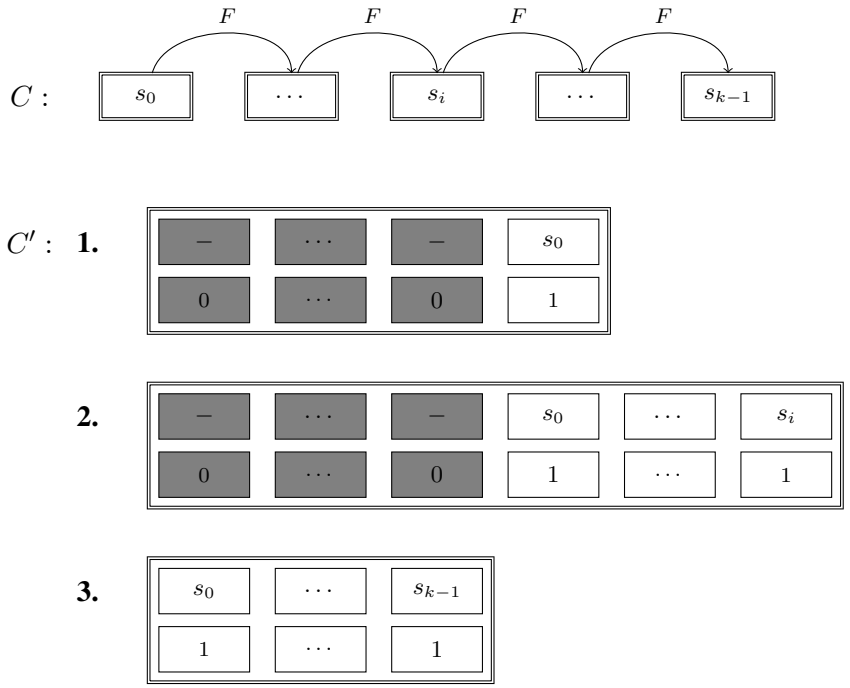

Figure 4.3: The diagram shows three possible initial states of C'. Here (1) illustrates 1-initialisation, (2) is i-initialisation, and (3) full initialisation. The grey area are the uninitialised components (the “don’t care"s).

the original circuit. The reset functions of the X^i latches (for $i < k-1$) ensure they are initialised in a non-deterministic fashion. As for the initialisation bits B, their reset values are deterministic, depending on the initialisation status of the components.

4. The **transition** function of the $(k-1)^{\text{th}}$ copy of latches is identical to the original transition function, while every older component simply saves the value of its one timestep younger component.

5. Finally, the **property** is composed of five sub-properties, where h^i is satisfied whenever the two adjacent components follow the original transition relation.

Fig. 4.4 illustrates a comparison of variable structures of the original circuit and its k-witness (this also suggests their combinational extension relation). The area marked yellow (left box and top right box on the right) consists of the same set of variables. We consider each pair (X^i, L^i) as a *component* in the circuit and refer to (X^{k-1}, L^{k-1}) as the most recent component (youngest copy), and (X^0, L^0) as the oldest component (copy). Additionally we also refer to the inputs I' as X^{k-1} for convenience.

The property P' is comprised of five sub-properties. The *monotonicity* property p_0 expresses the monotonic nature of the initialisation bits. Intuitively, if a component is initialised, all components younger than it should also be initialised. The *transition*

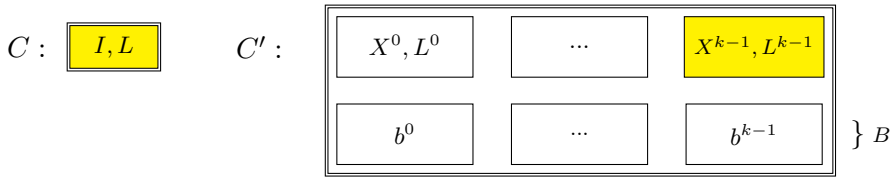

Figure 4.4: The structure of input and latch variables in C and C'.

property p_1 expresses the property that every initialised component has to follow the transition relation in the original circuit. Of particular interest is the k-safety property p_2, which says the original property P needs to be satisfied in every initialised component. The *reset* property p_3 expresses the property that in the case of partial initialisation, the oldest initialised component needs to satisfy the original reset function. Finally, p_4 expresses that at least the youngest component should have the initialisation bit set.

We now show the combinational simulation relation between the original circuit and its k-witness circuit.

Theorem 4.13. *The circuit C is combinationally simulated by its k-witness circuit.*

Proof. By the construction in Def. 4.12, the inputs stay the same in the k-witness circuit C', and the new latches are a superset of the original ones (the youngest component in C'). Thus by Def. 4.6, C' combinationally extends C. Based on Def. 4.12.4, the transition function of L^{k-1} is identical to the original one, which satisfies Def. 4.7.2. In the new property, p_4 and p_2 together imply $P(X^{k-1}, L^{k-1})$. In other words, the original property holds in the most recent component. This then satisfies Def. 4.7.3. By Def. 4.12, for every satisfiable assignment of $R(L)$, the same assignment satisfies $R'(L)$ on the common latches (the youngest component). For all the new latches we observe the following. Because the reset of the newest component is satisfiable with the same assignment as in the original circuit, we can see that u^{k-1} is true in the k-witness circuit and therefore all other u^i are also true. Therefore all the ite-statements of the reset definition become trivially satisfiable. To complete the argument, by Def. 4.12.3, all the initialisation bits can be now set to $\bot$ except b^{k-1} which can be set to $\top$. A satisfying assignment of $R'(L')$ can thus be directly constructed (deterministically in polynomial time) from any satisfying assignment of $R(L)$. This implies the reset condition of Def. 4.7.1 holds. (Sidenote: This implies that the QBF check needed in the combinational simulation relation could potentially be solved easily in practice for these k-witness circuits.) Therefore C' combinationally simulates C.

□

In the following, we present the main result of this paper on the relationship between a circuit C and its k-witness circuit C' in terms of k-induction.

Theorem 4.14. *Given a circuit C, a fixed $k \in \mathbb{N}^+$, and its k-witness circuit C', P is k-inductive in C iff P' is 1-inductive in C'.*

Proof. We consider the two k-inductive checks in Def. 6.4 for both directions. In Theorem 4.21 we show that the BMC check (of the initialised unrolling of length $k-1$) in C passes, if and only if the same check (of the initialised unrolling of length 0) in C' also passes. In Theorem 4.22 we prove that if the consecution check of C' passes, then the consecution check also passes in C. Lastly, Theorem 4.25 shows that if P is k-inductive in C, then the consecution check of P' using the unrolling of length 1 passes in C'. By combining them together, we conclude P is k-inductive in C iff P' is 1-inductive in C'.

□

For the BMC check in the two circuits, we need to analyse three separate cases as shown in Fig. 4.3, which correspond to Lemmas 4.16, 4.17, and 4.18, respectively. But before this we need a technical Lemma 4.15 on the initialisation bits. In the following context, we consider a given circuit C, and its k-witness circuit C' with a fixed k.

Lemma 4.15. *For the initialised unrolling of length* 0 *of the k-witness circuit C', the reset values of the initialisation bits B_0 are deterministic and depend only on the component with the largest index $i \in [0,k)$ for which $R(L_0^i)$ is satisfied.*

Proof. Firstly, we define $S = \{i \mid R(L_0^i)\}$, based on which we consider two cases.

(1). By Def. 4.12.3(c), if $\neg u_0^1$, then $0 \in S$. In this case, $b_0^0 = \top$ by Def. 4.12.3(f), and by Def. 4.12.3(e)(g), $b_0^1, ..., b_0^{k-1}$ are all set to $\top$.

(2). Otherwise we consider u_0^1, where S contains at least some $i \in [1,k)$. Let m be the maximum index in S, and $m \neq 0$. Since $R(L_0^m)$, u_0^m is satisfied, so are $u_0^{m-1}, ..., u_0^1$, while $u_0^{m+1}, ..., u_0^{k-1}$ are not. In Def. 4.12.3(g), for all $i \in S$, $R(L_0^i) \wedge \neg u^{i+1}$ is only satisfied when $i = m$, thus $b_0^m = \top$. Therefore $b_0^i = \top$ for all $i \in [m+1, k)$. By Def. 4.12.3(f), $b_0^0 = \bot$, therefore for all $i \in [1, m), b_0^i = \bot$. □

Initialisation bits are indicators for the initialisation status of the k-witness circuit. We observe that the sub-properties $p_0, ..., p_3$ of the k-witness circuit trivially hold for uninitialised components (*i.e.,* those for which the initialisation bit is 0), while p_4 solely depends on b^{k-1}.

Lemma 4.16. *If the initialised unrolling of length $k-1$ of the original circuit C is safe, the initialised unrolling of length* 0 *of the k-witness circuit C' is also safe, in the case of* 1*-initialisation.*

Proof. Assume $U_{k-1} \wedge R(L_0) \Rightarrow \bigwedge_{i \in [0,k)} P(I_i, L_i)$ such that the initialised unrolling of C is safe. In the case of 1-initialisation, we consider $R'(L'_0) \wedge R(L_0^{k-1})$ as the initialised unrolling of C', as U'_0 is trivial. By Lemma 4.15 and Def. 4.12.3, for the initialisation bits, only b_0^{k-1} is set to $\top$ and the rest remain $\bot$. The values of B_0 then satisfy $p_0(I'_0, L'_0), p_1(I'_0, L'_0), p_4(I'_0, L'_0)$ trivially. Every satisfying assignment of $R'(L'_0) \wedge R(L_0^{k-1})$ satisfies $R(L_0)$ with $L_0 = L_0^{k-1}, I_0 = X_0^{k-1}$. Similar to our argument in Theorem 4.9, $U_{k-1} \wedge R(L_0)$ is then also satisfiable. By our assumption, $P(X_0^{k-1}, L_0^{k-1})$ is thus satisfied. The premise of $p_2(I'_0, L'_0)$ is only satisfied for

b_0^{k-1}, and with the same assignment satisfying $P(X_0^{k-1}, L_0^{k-1})$, $p_2(I'_0, L'_0)$ is also satisfied. Lastly, the premise of $p_3(I'_0, L'_0)$ is only satisfied for $\neg b_0^{k-2} \wedge b_0^{k-1}$, and since $R(L_0^{k-1}), p_3(I'_0, L'_0)$ is satisfied. Therefore we have $P'(I'_0, L'_0)$. □

Lemma 4.17. *If the initialised unrolling of length $k-1$ of the original circuit C is safe, the initialised unrolling of length 0 of the k-witness circuit C' is also safe, in the case of i-initialisation.*

Proof. Firstly, we assume $U_{k-1} \wedge R(L) \Rightarrow \bigwedge_{i\in[0,k)} P(I_i, L_i)$. In the case of i-initialisation, we consider $R'(L'_0) \wedge R(L_0^m) \wedge \neg u^{m-1}$ as the initialised unrolling of C', where $m \in [1, k-1)$ is the largest index for which $R(L_0^m)$ is satisfied. As we showed in Lemma 4.15, $b_0^m, ..., b_0^{k-1}$ are set to $\top$ while $b_0^0, ...b_0^{m-1}$ are $\bot$. Following Def. 4.12.3, $L_0^i \simeq F(X_0^{i-1}, L_0^{i-1})$ for all $i \in (m, k)$, while all components older than m are uninitialised. Every satisfying assignment of $R'(L'_0) \wedge R(L_0^m) \wedge \neg u^{m-1}$ also satisfies $\bigwedge_{i\in[0,k-m-1)} (L_{i+1} \simeq F(I_i, L_i)) \wedge R(L_0)$ with $I_{i-m} = X_0^i, L_{i-m} = L_0^i$ for all $i \in [m, k)$. In the rest of the proof, we fix the assignment satisfying $R'(L'_0) \wedge R(L_0^m) \wedge \neg u^{m-1}$. Similar to our argument in Theorem 4.9, $U_{k-1} \wedge R(L_0)$ is satisfiable with our fixed assignment. By our assumption, $\bigwedge_{i\in[m,k)} P(X_0^i, L_0^i)$ is then satisfied. We now consider $P'(I'_0, L'_0)$. As the premise of $p_2(I'_0, L'_0)$ is only satisfied for $b_0^m, ..., b_0^{k-1}$, $p_2(I'_0, L'_0)$ is satisfied. Similarly for the transition property, with $L_0^i \simeq F(X_0^{i-1}, L_0^{i-1})$ for all $i \in (m, k)$, $p_1(I'_0, L'_0)$ is satisfied. Given the values of B_0, the monotonicity property is satisfied. In addition, $p_4(I'_0, L'_0)$ is also satisfied as $b_0^{k-1} = \top$. Finally, the premise of $p_3(I'_0, L'_0)$ is only satisfied for $\neg b_0^{m-1} \wedge b_0^m$, and as we already have $R(L_0^m)$, p_3 is satisfied. □

Lemma 4.18. *If the initialised unrolling of length $k-1$ of the original circuit C is safe, the initialised unrolling of length 0 of the k-witness circuit C' is also safe, in the case of full initialisation.*

Proof. We assume $U_{k-1} \wedge R(L) \Rightarrow \bigwedge_{i\in[0,k)} P(I_i, L_i)$ for the original circuit. Since we consider full initialisation, $R'(L'_0) \wedge R(L_0^0) \wedge \neg u_0^1$ is the initialised unrolling of C'. Following Def. 4.12.3, $L_0^i \simeq F(X_0^{i-1}, L_0^{i-1})$ for all $i \in [1, k)$. Every satisfying assignment of $R'(L'_0) \wedge R(L_0^0) \wedge \neg u_0^1$ satisfies $U_{k-1} \wedge R(L_0)$ with $I_i = X_0^i, L_i = L_0^i$ for all $i \in [0, k)$. The rest of the proof follows the same logic as in Lemma 4.17. □

Lemma 4.19. *If the BMC check for the unrolling of length $k-1$ of the original circuit C passes, then the BMC check for the unrolling of length 0 of the k-witness circuit C' also passes.*

Proof. Based on Def. 4.12.3, we consider the BMC check for all possible initial states. Lemma 4.16, 4.17 and 4.18 cover the case-split over all initial states of C' based on whether each component satisfies the original reset function $R(L_0^i)$ or not. We show that the BMC check of C' passes under the same assumption for three initialisation

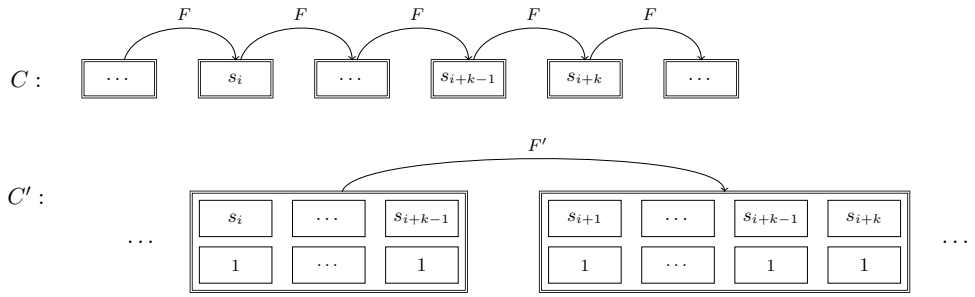

Figure 4.5: The diagram shows the consecution check in C and C'.

cases respectively. In particular, our construction in Def. 4.12.3 does not allow all components to be uninitialised, in which case $R'(L'_0)$ becomes unsatisfiable (more specifically, $R'(L^0_0)$ is unsatisfiable). We conclude the BMC check of the initialised unrolling of length 0 passes in C'.

□

We proceed to prove the opposite direction of the BMC check for C and C' by considering the reset status in the k-witness circuit.

Lemma 4.20. *If the BMC check for the unrolling of length 0 of the k-witness circuit C' passes, then the BMC check for the unrolling of length $k-1$ of the original circuit C also passes.*

Proof. We assume the BMC check passes in the k-witness circuit, $R'(L'_0) \Rightarrow P'(I'_0, L'_0)$. We do a proof by contradiction by assuming the BMC check of length $k-1$ fails for the original circuit. Thus there exists a satisfying assignment s of $U_{k-1} \wedge R(L_0) \wedge \neg \bigwedge_{i \in 0,k)} P(I_i, L_i)$. We can construct a satisfying assignment of $R'(L'_0)$ as follows. Let $a \in [0, k)$ be some index for which $\neg P(I_a, L_a)$ is satisfied. Let $m \in [0, a]$ be the index for which $R(L_m) \wedge \neg \bigvee_{i \in (m,a]} R(L_i)$ is satisfied. Let $X_0^{k-1-i} = I_{a-i}, L_0^{k-1-i} = L_{a-i}, b_0^{k-1-i} = \top$ for all $i \in [0, a-m]$. The rest of initialisation bits of B_0 are set to $\bot$. By Def. 5.2, we have $L_{i+1} \simeq F(I_i, L_i)$ for all $i \in [m, a)$, which satisfies Def. 4.12.3(d). As our construction satisfies $R'(L'_0)$, by our assumption, $P'(I'_0, L'_0)$ is satisfied. By Theorem 5.13, $P(I_a, L_a)$ is satisfied. Since we assume s satisfies $\neg P(I_a, L_a)$, we have reached a contradiction.

□

As an immediate consequence of Lemma 4.19 and 4.20, the BMC check of C passes iff the same check passes in C'. We record the result in the following Theorem.

Theorem 4.21. *The BMC check for the unrolling of length 0 of the k-witness circuit C' passes, if and only if the BMC check for the unrolling of length $k-1$ of the original circuit C passes.*

We show in Fig. 4.5 an illustration of the consecution check in both circuits.

Theorem 4.22. *If the consecution check for the unrolling of length 1 of the k-witness circuit C' passes, then the consecution check for the unrolling of length k of the original circuit C passes too.*

Proof. We assume $U'_1 \wedge P(I'_0, L'_0) \Rightarrow P(I'_1, L'_1)$ holds. We then do a proof by contradiction by assuming that the consecution check for the original circuit fails. Thus there is a satisfying assignment s of the formula $U_k \wedge \bigwedge_{i \in [0,k)} P(I_i, L_i) \wedge \neg P(I_k, L_k)$. Based on s, we have a satisfying assignment for $U'_1 \wedge P'(I'_0, L'_0)$ as follows. Let $X^i_0 = I_i, L^i_0 = L_i$, and $b^i_0 = \top$ for all $i \in [0, k)$. Let $X^{i-1}_1 = I_i, L^{i-1}_1 = L_i, b^{i-1}_1 = \top$ for all $i \in [1, k]$. We now show this satisfies $L'_1 \simeq F'(I'_0, L'_0)$. Since $X^{i-1}_1 = I_i = X^i_0$ and $L^{i-1}_1 = L_i = L^i_0$ for all $i \in [1, k)$, Def. 4.12.4(a) and Def. 4.12.4(c) are satisfied. Since s satisfies U_k, by Def. 5.2, it satisfies $L_k \simeq F(I_{k-1}, L_{k-1})$. With $X^{k-1}_0 = I_{k-1}, L^{k-1}_0 = L_{k-1}$, and $L^{k-1}_1 = L_k$, we have $L^{k-1}_1 \simeq F(X^{k-1}_0, L^{k-1}_0)$, and thus Def. 4.12.4(b). As for the initialisation bits, since all of them are set to $\top$ in both B_0 and B_1, Def. 4.12.4(d) is satisfied. As a result, U'_1 is satisfied, and we continue to show the same assignment satisfies $P'(I'_0, L'_0)$. Similar to our proof in Lemma 4.17, the values of B_0 satisfy $p_0(I'_0, L'_0)$ and $p_4(I'_0, L'_0)$ immediately. As the premiss of $p_3(I'_0, L'_0)$ is unsatisfiable, $p_3(I'_0, L'_0)$ trivially holds. Since U_k is satisfied, by Def. 5.2, we have $L_{i+1} \simeq F(I_i, L_i)$ which satisfies h^i_0 for all $i \in [0, k-1)$, thus also $p_1(I'_0, L'_0)$. Lastly, since $P(I_i, L_i)$ is satisfied for all $i \in [0, k)$, the original property is satisfied in every component $P(X^i_0, L^i_0)$, resulting in the satisfaction of $p_2(I'_0, L'_0)$. By our initial assumption, $P'(I'_1, I'_1)$ is satisfied. By Theorem 5.13, we have $P(X^{k-1}_1, L^{k-1}_1)$, thus $P(I_k, L_k)$. We reach a contradiction here. We can therefore conclude the consecution check of the original circuit passes. □

Lemma 4.23. *If the safety property P is k-inductive in the original circuit C, the consecution check of the unrolling of length 1 passes in the k-witness circuit C', given that L'_0 is partially initialised.*

Proof. Assume P is k-inductive in C. Let U'_1 be the unrolling of C', and $m \in [1, k)$ is some index such that $b^0_0, ..., b^{m-1}_0$ are set to $\bot$, while $b^m_0, ..., b^{k-1}_0$ are set to $\top$ (as we consider partial initialisation here). We do a proof by contradiction, and assume there is a satisfying assignment s of the negation of the consecution check formula $U'_1 \wedge P'(I'_0, L'_0) \wedge \neg P'(I'_1, L'_1)$. Since we assume $P'(I'_0, L'_0)$, it implies $R(L^m_0)$, based on $p_3(I'_0, L'_0)$. We also have $L^{i+1}_0 \simeq F(X^i_0, L^i_0)$ for $i \in [m, k-1)$, based on $p_1(I'_0, L'_0)$. Furthermore, U'_1 implies $L'_1 \simeq F'(I'_0, L'_0)$, and by Def. 4.12.4, $L^{k-1}_1 \simeq F(X^{k-1}_0, L^{k-1}_0)$. Therefore the same assignment satisfies $U_{k-1} \wedge R(L_0)$ where $I_{i-m} = X^i_0, L_{i-m} = L^i_0$ for all $i \in [m, k)$, and $I_{k-m} = I'_1, L_{k-m} = L^{k-1}_1$. By our assumption that the BMC check passes in C, we have $P(X^i_0, L^i_0)$ for all $i \in [m, k)$ and $P(I'_1, L^{k-1}_1)$.

We can then proceed to prove $P'(I'_1, L'_1)$ is indeed satisfied. Similar to our proof in Theorem 4.22, based on Def. 4.12.4, $b^i_1 = \top$ for all $i \in [m, k)$ while $b^i_1 = \bot$ for all $i \in [0, m)$. Additionally, $X^i_1 = X^{i+1}_0, L^i_1 = L^{i+1}_0$ for $i \in [0, m-1)$. The rest of the proof follows the same logic as Theorem 4.22 for showing $P'(I'_1, L'_1)$ is satisfied. We then reach a contradiction here, and thus conclude the consecution check for C' passes in this case. □

Lemma 4.24. *If the consecution check for the unrolling of length k passes in the original circuit C, the consecution check for the unrolling of length 1 passes in the k-witness circuit C', given that L'_0 is fully initialised.*

Proof. Let U'_1 be the unrolling of C' with $b_0^0, ..., b_0^{k-1}$ all set to $\top$. Similar to Lemma 4.23, we do a proof by contradiction, and assume there is a satisfying assignment s of $U'_1 \wedge P'(I'_0, L'_0) \wedge \neg P'(I'_1, L'_1)$. By the transition property $p_1(I'_0, L'_0)$, the components follow the transition function F, such that $L_1^{i+1} \simeq F(X_0^i, L_0^i)$ for all $i \in [0, k-1)$. Similar to our argument in Lemma 4.23, U'_1 implies $L_1^{k-1} \simeq F(I'_0, L_0^{k-1})$. We also have $\bigwedge_{i\in[0,k)} P(X_0^i, L_0^i)$ based on $p_2(I'_0, L'_0)$ and the values of B_0. The same assignment thus satisfies $U_k \wedge \bigwedge_{i\in[0,k)} P(L_i, L_i)$ where $L_i = L_0^i \wedge I_i = X_0^i$ for all $i \in [0, k)$ and $I_k = I'_1, L_k = L_1^{k-1}$. Based on our assumption that the consecution of C passes, we have $P(I'_1, L_1^{k-1})$. Following the same reasoning in Lemma 4.23, after one transition, $b_1^i = \top$ for all $i \in [0, k)$, and $X_1^i = X_0^{i+1}, L_1^i = L_0^{i+1}$ for $i \in [0, k-1)$.

We can now show $P'(I'_1, L'_1)$ is satisfied. The k-safety property $p_2(I'_1, L'_1)$ is satisfied as we have proved $p(X_1^i, L_1^i)$ for all $i \in [0, k)$. The transition property $p_1(I'_0, L'_0)$ is preserved, as U_k is satisfied which implies $L_1^{i+1} \simeq F(X_1^i, L_1^i)$. Based on the values of B_1, $p_0(I'_1, L'_1), p_3(I'_1, L'_1), p_4(I'_1, L'_1)$ are satisfied immediately. We conclude the $P'(I'_1, L'_1)$ is satisfied thus we reach a contradiction. Therefore the consecution check for C' passes in this case. □

Theorem 4.25. *If both k-induction checks pass in the original circuit C, then the consecution check of the unrolling of length 1 in the k-witness circuit C' passes.*

Proof. First of all, we assume both checks pass in C. We then do a proof by contradiction by assuming there is a satisfying assignment s for the negation of the consecution check $U'_1 \wedge P'(I'_0, L'_0) \wedge \neg P'(I'_1, L'_1)$. Since s satisfies $U'_1 \wedge P'(I'_0, L'_0)$, we consider two separate cases where the property $P'(I'_0, L'_0)$ is satisfied: full initialisation or partial initialisation. Note when all $b_0^0, ..., b_0^{k-1}$ are set to $\bot$, $P'(I'_0, L'_0)$ is not satisfied. Therefore applying Lemma 4.24 and Lemma 4.23 together, we conclude if both k-induction checks pass in C, the consecution check of the unrolling of length 1 in the k-witness circuit also passes. □

We briefly discuss why the k-witness circuit is linear in the size of the original circuit, and the value k. If we consider the circuit size in terms of gate numbers, the number of latches and inputs increase by a factor of approximately k. The transition functions are copied $k-1$ times, i.e., $k-2$ times for reset in Def. 4.12.3(d), and once more in 4.12.4(b), while the $k-2$ copies in the property part 4.12.5(a) have the same arguments and can be shared. For the reset predicates, defining $R(L^i)$ is linear in the number of the latches, while u^i is linear in k. We apply the same logic when defining the property, therefore we conclude our construction is linear in the size of the circuit and k.

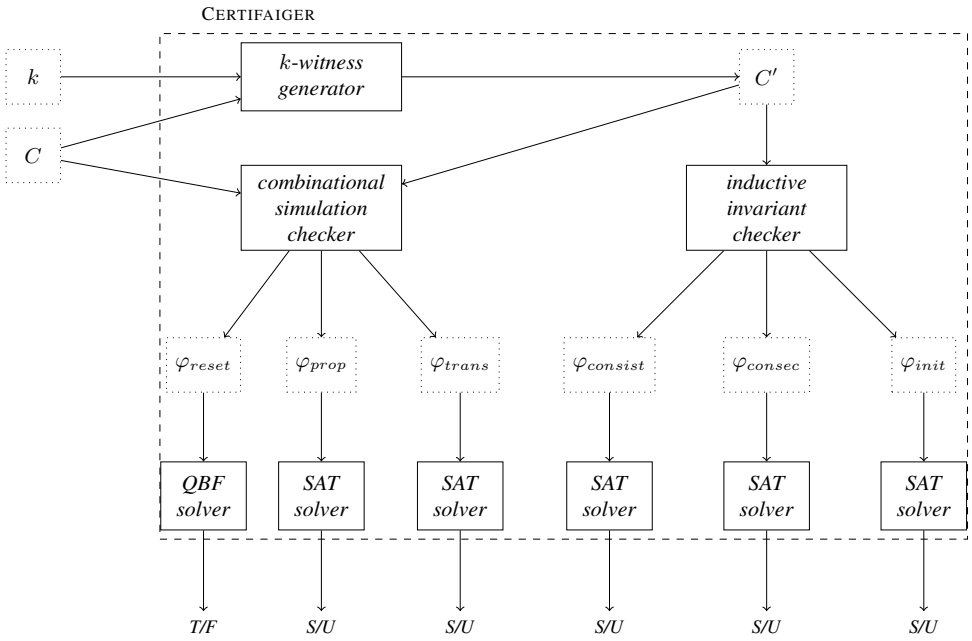

Figure 4.6: The architecture of CERTIFAIGER. C is the input circuit in AIGER format and k is the value given by a k-induction-based model checker. The final outputs of the SAT solvers are given in the form of S/U, for satisfiable or unsatisfiable. The QBF solver outputs true or false (T/F) as the result.

4.5 Implementation

Based on our new construction we implemented CERTIFAIGER [38], which works as a tool suite comprised of multiple components as shown in Figure 4.6. The tool takes as inputs a circuit which contains a safety property given in AIGER format [22] and a value k provided by a k-induction-based model checker which outputs a positive model checking result. Upon invocation, internally the inputs are passed on to the k-witness generator that parses the AIGER file and generates a k-witness circuit as defined in Def. 4.12. The new safety property is a simple inductive invariant (to be verified) for the k-witness circuit. We extended the reset logic definition of the existing AIGER format defined by the authors of [22] to enable reset functions, whereas all previous AIGER versions only allow reset values to be 0, 1, or uninitialised. The k-witness circuit from the k-witness generator is given in this extended AIGER format.

To verify the inductive invariant $\phi(I, L)$, as discussed in Section 4.3, our certifier generates three conditions. (Note that here we are only looking at extended circuits, therefore we use L instead of L'.)

Condition	Formula	The inductive invariant ...
"*initiation*"	$R(L) \Rightarrow \phi(I, L)$	... must hold at all initial states.
"*consistency*"	$\phi(I, L) \Rightarrow P(I, L)$	... must hold at all good states.
"*consecution*"	$U_1 \wedge \phi(I_0, L_0) \Rightarrow \phi(I_1, L_1)$	... is preserved during the transition.

In our implementation, the latch variables used in the inductive invariant are updated with their next state literals after each transition. The consistency condition is rather trivial here, as the inductive invariant is exactly the property in the k-witness circuit, although this is only specific to our case.

Our certifier generates for each of the three conditions a (combinational) AIGER circuit which is then checked by a SAT solver. In our implementation, we used the SAT solver Kissat [21] for checking validity of the formulas after they have been converted to CNF by invoking AIGTOCNF from the AIGER library.

Furthermore, we implemented the combinational simulation checker for verifying the combinational simulation relation described in Def. 4.7. The checker takes as inputs the original circuit and the k-witness circuit. It generates two AIGER files for the transition check and the property check, as well as a QAIGER file for the reset check, as defined in Def. 4.7. Similar to the inductive invariant checker, the AIGER files are then converted to CNFs and verified by Kissat. QAIGER is a standard format used in QBF Competitions. In our experiments the formula is verified with the QBF solver QuAbS [127].

The tool CERTIFAIGER returns "SUCCESS" as a result if all six formulas hold, meaning that the circuit C' combinationally simulates C and C' is safe by the 1-induction proof. Thus by Theorem 4.9 the original circuit C is also safe. Note that this result holds regardless of how C' is constructed.

Given a scenario where we would want to place trust on the correctness of the extended circuit mapping inside the k-witness generator (to trust that the k-witness circuit construction of Def. 4.12 is correct and the program implementing it is also provably correct), all three combinational simulation checks (one QBF and two SAT checks) could be skipped in the certification procedure.

Intuitively, given a faulty generation of the k-witness circuit C', the error would either be caught by the combinational simulation check (due to an erroneous under-approximation of the set of reachable states) or the inductive invariant check (due to an erroneous over-approximation of the set of reachable states). Furthermore, we have also done a sanity check of certification on failure, where the model checking results are falsified by CERTIFAIGER. An incorrect value of k is detected by a negative result of φ_{consec}, whereas φ_{init} does not hold in cases where an initial state is a bad state.

4.6 Experiments

As described in previous sections, the complexity of extending the original circuit into k-witness is linear in the size of the circuit, and the inductive depth. To evaluate the practicality of our tool, we now report the experimental results obtained by evaluating CERTIFAIGER against a number of widely used benchmarks. The benchmarks were first run on the open source k-induction-based model checker McAiger [15], which was modified to give the values of k explicitly. All experiments were carried out on an Intel$^{\text{ő}}$ Core™ i9-9900 CPU 3.60GHz computer with 32GB RAM running Manjaro with kernel version 5.4.72-1.

We start with the TIP suite benchmarks which were originally used in [58]. The

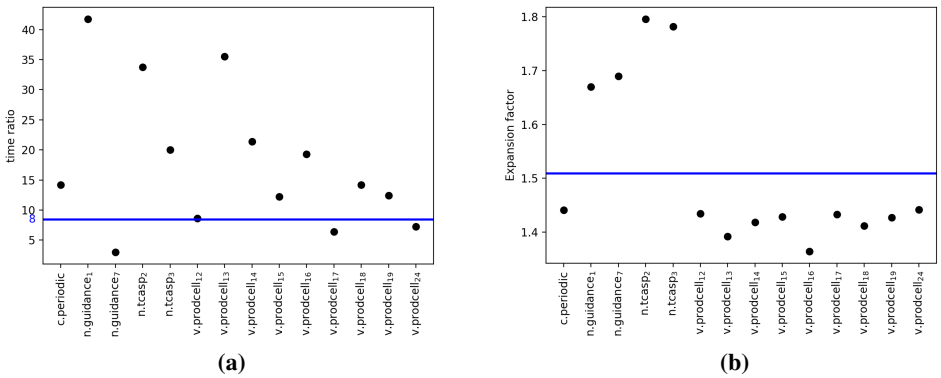

Figure 4.7: The time (a) and file size (b) comparison results for the TIP suite. The benchmark names are shown on the x-axis. Average values are shown as the blue horizontal line in each plot. The y-axis of (a) displays the time ratio of total certification time and model checking time. The y-axis of (b) shows the expansion factor indicating the comparison of circuit sizes (k-witness circuit v.s the original).

benchmarks were converted from .SMV to AIGER by invoking SMVTOAIG from the AIGER library. Table 5.1 reports the certification results obtained, where the file names are associated by the origin of the problems explained in [58]. The table displays the following information in each column:

1. the name of the AIGER file,
2. the verification time on McAiger,
3. the size of the original circuit, in terms of the number of gates (thousands),
4. the k-inductive value k given by the model checker,
5. the size of the k-witness circuit,
6. the time taken on the k-witness generator, and
7. the size and solving time (seconds) of each condition.

Note here we selected benchmarks that gave a positive model checking result, only in which case the original property is k-inductive. Moreover, three instances that require simple paths constraints (also called *loopFree* constraints in [124]) were ruled out. Handling these constraints is an interesting area for future study. We retrieved the inductive depths k from the model checker McAiger, and compared with the results in [58] to ensure the values are identical. As shown in Table 5.1, the values of k vary between 4 and 96. The SAT solver was able to handle the proof checking without experiencing time-outs. We observe that the k-witness circuit generation time is rather

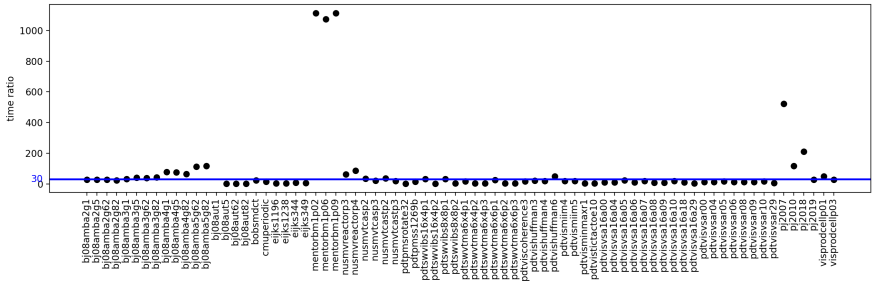

Figure 4.8: Certification time *vs.* model checking time obtained by running HWMCC'10 benchmarks.

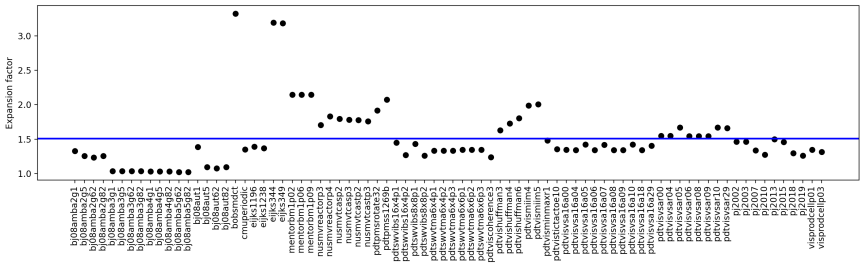

Figure 4.9: The k-witness circuit size *vs.* the original circuit size.

small, compared with the model checking time as well as the proof checking time. In the proof checking stage, Table 5.1 suggests that the SAT-solving time for φ_{consec} is much higher than the rest of the formulas. This is as expected, as the formula φ_{consec} is in general more complicated than the rest, and appears to be the most difficult formula to solve. In addition, QBF solving times are also worth-noting: in a few cases QBF solving time is longer than for other formulas, however, in most cases, it is rather small. To compare certification time with model checking time, we plotted the results in Fig. 4.7a, where the y-axis shows the ratio of certification and model checking. Here certification time is the sum of time taken on each component, assuming the six conditions are computed in parallel. As shown in the diagram, the average time ratio is around 8, which is quite promising. Furthermore, Fig. 4.7b shows a comparison of circuit sizes, where the *expansion factor* ε is computed by $\frac{\#C'}{\#C \times k}$(alternatively, $\#C' = \varepsilon \cdot \#C \cdot k$). The average value observed here is around 1.5. This is consistent with Def. 4.12, as we expected the size of the k-witness circuit to grow linearly with respect to the original circuit and the value of k.

We also used benchmarks from the Hardware Model Checking Competition (HWMCC) 2010 [20]. The benchmarks were pre-filtered by running on McAiger with a time-out of 15 minutes. A total of 513 instances were solved by McAiger, from which we selected from the 216 *unsat* instances with a meaningful k (*i.e.,* $k \geq 2$). We also observed only 7 out of the 216 instances require simple path constraints. The results in Fig. 4.8 are

Table 4.1: Experimental results for the TIP suite.

Name	t	#C	k	#C'	t'	φ_{init} #	t	$\varphi_{consist}$ #	t	φ_{consec} #	t	φ_{trans} #	t	φ_{prop} #	t	φ_{reset} #	t
c.periodic	6.01	1.56	96	215.79	0.06	242.91	5.62	215.79	0.06	424.80	57.54	217.42	0.15	217.28	0.06	216.06	85.25
n.guidance$_1$	0.08	1.91	10	31.89	0.01	38.39	0.21	31.89	0.01	62.16	3.34	33.97	0.12	33.63	0.01	32.58	1.24
n.guidance$_7$	8.57	2.00	27	91.22	0.03	109.35	3.58	91.216	0.02	177.90	18.24	93.39	0.12	93.04	0.02	91.90	25.61
n.tcasp$_2$	0.08	3.02	6	32.54	0.01	39.76	0.16	32.542	0.01	63.28	2.70	35.92	0.26	35.23	0.02	33.92	1.84
n.tcasp$_3$	0.09	2.98	5	24.23	0.01	32.47	0.13	26.56	0.01	51.63	1.80	29.90	0.26	29.21	0.02	27.94	1.04
v.prodcell$_{12}$	7.60	2.91	29	121.07	0.04	133.59	2.66	121.07	0.03	239.01	65.76	124.19	0.12	123.88	0.03	121.69	8.61
v.prodcell$_{13}$	0.10	2.91	8	32.41	0.01	35.78	0.20	32.41	0.01	63.97	3.55	35.52	0.12	35.21	0.01	33.03	0.21
v.prodcell$_{14}$	0.81	2.91	16	66.03	0.02	72.88	0.73	66.03	0.02	130.34	17.30	69.14	0.12	68.83	0.02	66.65	1.48
v.prodcell$_{15}$	2.94	2.91	23	95.60	0.03	105.51	2.05	95.60	0.03	188.74	35.87	98.72	0.12	98.41	0.02	96.23	4.34
v.prodcell$_{16}$	0.07	2.91	5	19.85	0.01	21.91	0.04	19.85	0.01	39.18	1.35	22.97	0.12	22.66	0.01	20.47	0.06
v.prodcell$_{17}$	7.04	2.91	27	112.57	0.03	124220	2.46	112.57	0.03	222.22	44.88	115.68	0.12	115.37	0.03	113.19	6.95
v.prodcell$_{18}$	0.62	2.91	13	53.40	0.02	58.95	0.54	53.40	0.02	105.41	8.79	56.51	0.12	56.20	0.02	54.02	0.81
v.prodcell$_{19}$	2.67	2.91	22	91.37	0.03	100.84	1.99	91.37	0.03	180.37	33.09	94.48	0.12	94.17	0.03	91.99	3.81
v.prodcell$_{24}$	16.38	2.91	37	155.23	0.05	171.24	3.45	155.23	0.04	306.46	118.50	158.35	0.12	158.04	0.04	155.85	17.78

sorted by the benchmark names, which enables us to compare individual benchmarks from the same family. In most cases, similar to our previous observation from the TIP suite, the SAT solving time of φ_{consec} takes much longer than the rest, while in very few cases it is less than the QBF solving time for φ_{reset}. The average time ratio is 30, where we excluded 4 outliers in the plot coming from the $pj20$ family, that give a worse result (total certification time $\geq$ 15min). We observe that this was due to the high format conversion time from QAIGER to QCIR [85] before the QBF solving handled by QuAbS, while the actual QBF solving time was significantly smaller and more feasible. We believe this can be overcome by generating an alternative format directly in practice. Finally, similar to our previous TIP results, Fig. 4.9 shows the values of the expansion factor with an average of 1.5.

In the final experiments, to further inspect the expansion factor, we generalise the *Counter* example in Example 4.11, where we scale the number of bits to 500 with a modulo value 32. To clarify the complexity of our construction for the k-witness circuit, we ran experiments with different values of k. The results are shown in Fig. 4.10, where the x-axis shows the values of b up to 431, meaning the value of k was scaled up to 400. The expansion factor gradually converges to a constant as we increase the value of b, as we expected.

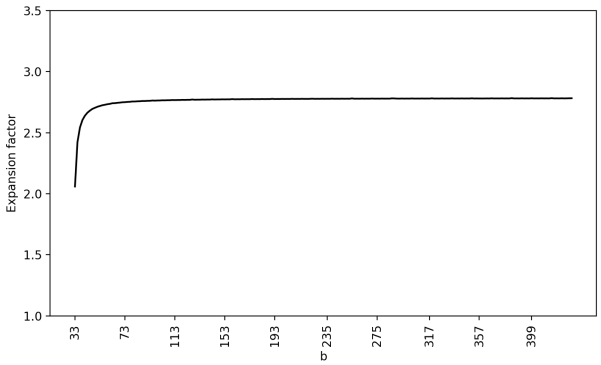

Figure 4.10: The experimental results of the *Counter* example. The values of *b* are shown on the x-axis.

As noticed above, overall our approach works efficiently in the certification stage, in particular, in our implementation we adopted the linear construction of k-witness circuit in Def. 4.12, thus the size of the resulting AIGER circuit is linear in the size of the original circuit, and the value of k. Each component in the tool suite works independently from each other when performing verification, which increases trust in the verification results.

4.7 Conclusion

We propose an approach to certify k-induction-based model checking results, by extending the model to produce an inductive invariant. The resulting tool, CERTIFAIGER, was evaluated experimentally on multiple sets of widely used benchmarks. The analysis showed our approach can be adapted to use in practice.

Our certificates are linear in size of the original problem and k. Validation requires several SAT checks and solving a simple QBF. In related work [27, 76] the worst case is considered to be exponential. It is an interesting open question whether our notion of combinational simulation requiring a QBF check for the reset condition can be changed to use only SAT checks.

Further, we only considered k-induction without simple paths constraints, even though such constraints on executions of the original model can in principle be handled by adding unique state constraints to our k-witness circuit. For simplicity we stick to models without such constraints, a restriction also made for instance in the hardware model checking competition. Thus certifying k-induction with simple path constraints is left to future work as well as handling different types of properties such as liveness properties.

We also want to extend our approach to common preprocessing techniques including temporal decomposition [36] or retiming [90] with the goal to obtain a single certificate (witness circuit). This goal is particularly challenging for complex multi-engine model checkers [32, 35]. Furthermore, we believe our approach can be extended to infinite-state systems, where k-induction is commonly used.

Chapter 5

Stratified Certification for k-Induction

Published This paper is an extended version of [25], published in Proceedings of the 22nd International Conference on Formal Methods in Computer-Aided Design (FMCAD 2022), pages 59–64, Trento, Italy, October 17-21, 2022.

Authors Emily Yu, Nils Froleyks, Armin Biere and Keijo Heljanko.

Acknowledgement This work is supported by the Austrian Science Fund (FWF) under the project W1255-N23, the LIT AI Lab funded by the State of Upper Austria, and Academy of Finland under the project 336092.

Abstract Our recently proposed certification framework for bit-level k-induction-based model checking has been shown to be quite effective in increasing the trust of verification results even though it partially involved quantifier reasoning. In this paper we show how to simplify the approach by assuming reset functions to be stratified. This way it can be lifted to word-level and in principle to other theories where quantifier reasoning is difficult. Our new method requires six simple SAT checks and one polynomial-time check, allowing certification to remain in co-NP while the previous approach required five SAT checks and one QBF check. Experimental results show a substantial performance gain for our new approach. Finally we present and evaluate our new tool CERTIFAIGER-WL which is able to certify k-induction-based word-level model checking.

5.1 Introduction

Over the past several years, there has been growing interest in system verification using word-level reasoning. Satisfiability Modulo Theories (SMT) solvers for the theory of fixed-size bit-vectors are widely used for word-level reasoning [112, 113]. For example, word-level model checking has been an important part of the hardware model checking competitions since 2019. Given the theoretical and practical importance of word-level verification, a generic certification framework for it is necessary. As quantifiers in combination with bit-vectors are challenging for SMT solvers and various works have focused on eliminating quantifiers in SMT [113, 114, 115], a main goal of this paper is to generate certificates without quantification which are thus efficiently machine checkable.

Temporal induction (also known as k-induction) [124] is a well-known model checking technique for verifying software and hardware systems. An attractive feature of k-induction is that it is natural to integrate it with modern SAT/SMT solvers, making it popular in both bit-level model checking and beyond [39, 50, 86], including word-level model checking.

Certification helps gaining confidence in model checking results, which is important for both safety- and business-critical applications. There have been several contributions focusing on generating proofs for SAT-based model checking [46,72,76,92,111,133,140]. For example [73] and [72] proposed an approach to certify LTL properties and a few preprocessing techniques by generating deductive proofs. In this paper, we focus on finding an inductive invariant for k-induction. Unlike other SAT/SMT-based model checking techniques such as IC3 [29] and interpolation [105, 106], k-induction does not automatically generate an inductive invariant that can be used as a certificate [102]. In previous research [137], certification of k-induction can be achieved via five SAT checks together with a one-alternation QBF check, redirecting the certification problem to verifying an inductive invariant in an extended model that simulates the original one.

At the heart of the present contribution is the idea of reducing the certification method of k-induction to pure SAT checks, *i.e.,* eliminating the quantifiers. This enables us to complete the certification procedure at a lower complexity, and to directly apply the framework to word-level certification. We introduce the notion of stratified simulation which allows us to reason about the simulation relation between two systems.

This stratified simulation relation can be verified by three SAT and a polynomial-time check. The latter checks whether the reset function definition is indeed stratified. In addition to that, we present a witness circuit construction which simulates the original circuit under the stratified simulation relation thus creating a simpler and more elegant certification construction for k-induction.

While the previous work only focused on bit-level model checking, we also lift our method to word-level by implementing a complete toolsuite CERTIFAIGER-WL, where the experiments show the practicality and effectiveness of our certification method for word-level models.

The rest of the paper is organised as follows. Section 5.2 includes the preliminaries where we fix the notation and basic concepts. In Section 6.5 we present the formal definition of stratified simulation and the witness circuit construction for k-induction

with an inductive invariant. We continue to show that the witness circuit simulates the given circuit and provide a formal proof that the inductive invariant can be used as a proof certificate. We give details of our implementation of the approach for both bit-level and word-level model checking in Section 7.1, where we also provide experimental results of our toolkits on different sets of benchmarks.

5.2 Background

This paper extends previous work in certification for k-induction-based bit-level model checking [137]. In this section, we present essential concepts and notations.

For the sake of simplicity we work with *functions* represented as interpreted terms and formulas over fixed but arbitrary theories which include an equality predicate. We further assume a finite sorted set of variables L where each variable $l \in L$ is associated with a finite domain of possible values. We also include Boolean variables as variables with a domain of $\{\top, \bot\}$, for which we keep standard notations.

For two sets of variables I and L, we also write I, L to denote their union. Given two functions $f(V)$, $g(V')$ where $V \subseteq V'$ (represented as interpreted terms over our fixed but arbitrary theories) we call them *equivalent*, written $f(V) \equiv g(V')$, if for every assignment to variables in V and V' that matches on the shared set of variables V, the functions $f(V)$, $g(V')$ have the same values. Additionally, we use " $\simeq$ " for syntactic equivalence [51], " $\rightarrow$ " for syntactic implication, and " $\Rightarrow$ " for semantic implication. To define semantical concepts or abbreviations we stick to equality " $=$ ". We use $vars(f)$ to denote the set of variables occurring in the syntactic representation of a function f.

In word-level model checking operations are applied to fixed-size bit-vectors. We introduce the notion of word-level circuits where we model inputs and latches as finite-domain variables. The specification of a circuit is described in terms of a "good states" property (so we focus on safety in this paper).

Definition 5.1 (Circuit)**.** A circuit is a tuple $C = (I, L, R, F, P)$ such that:

- I is a finite set of *input* variables.
- L is a finite set of *latch* variables.
- $R = \{r_l(L) \mid l \in L\}$ is a set of *reset functions*.
- $F = \{f_l(I, L) \mid l \in L\}$ is a set of *transition functions*.
- $P(I, L)$ is a function that evaluates to a Boolean output, encoding the *(good states) property*.

By Def. 6.1 a circuit represents a hardware system in a fully symbolic form. It is very close to bit-level models encoded in AIGER [22] format as well as word-level models in BTOR2 [116] format - both used in hardware model checking competitions. A state in the system is an assignment to the latches, and any state where all latches match their

reset functions is an initial state. In order to talk about the reset functions of a subset of latches $L'' \subseteq L$, we also write

$$R(L'') = \bigwedge_{l \in L''} (l \simeq r_l(L)).$$

The inputs represent the environment, which can be assigned any value thus bringing non-determinism to the circuit behaviour. The following four definitions are adapted from our previous work [137] for completeness of exposition.

Definition 5.2 (Unrolling)**.** For an unrolling depth $m \in \mathbb{N}$, the *unrolling* of a circuit $C = (I, L, R, F, P)$ of length m is defined as $U_m = \bigwedge_{i \in [0,m)} (L_{i+1} \simeq F(I_i, L_i))$.

For clarity, we use subscript such as L_i for the temporal direction (*e.g.,* in the context of unrolling), and later in the paper we use superscript for the spatial direction.

Definition 5.3 (Inductive invariant)**.** Given a circuit C with a property P, $\phi(I, L)$ is an *inductive invariant* in C if and only if the following conditions hold:

1. $R(L) \Rightarrow \phi(I, L)$, "initiation"
2. $\phi(I, L) \Rightarrow P(I, L)$, and "consistency"
3. $U_1 \wedge \phi(I_0, L_0) \Rightarrow \phi(I_1, L_1)$. "consecution"

As a generalisation of the notion of an inductive invariant, k-induction checks k steps of unrolling instead of 1. In the following, to verify that a property is an inductive invariant, we consider it as the special case of k-induction with $k = 1$ and $\phi(I, L) = P(I, L)$.

Definition 5.4 (k-induction)**.** Given a circuit C with a property P, P is called *k-inductive* in C if and only if the following two conditions hold:

1. $U_{k-1} \wedge R(L_0) \Rightarrow \bigwedge_{i \in [0,k)} P(I_i, L_i)$, and "BMC"
2. $U_k \wedge \bigwedge_{i \in [0,k)} P(I_i, L_i) \Rightarrow P(I_k, L_k)$. "consecution"

Definition 5.5 (Combinational extension)**.**
A circuit $C' = (I', L', R', F', P')$ combinationally extends a circuit $C = (I, L, R, F, P)$ if $I = I'$ and $L \subseteq L'$.

5.3 Certification

In this section we introduce and formalise our certification approach which reduces the certification problem to six SAT checks and one polynomial stratification check.

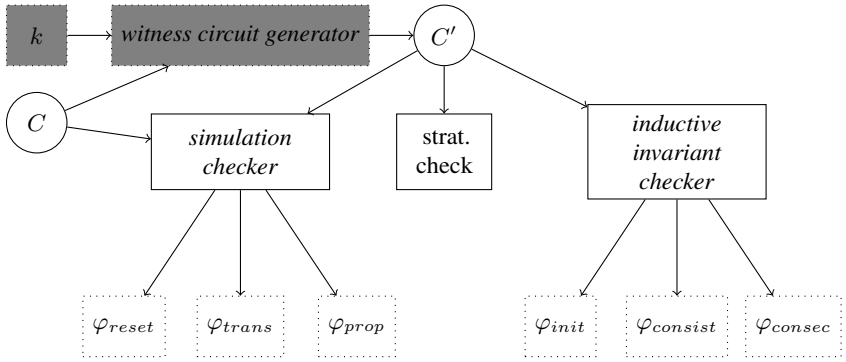

Figure 5.1: An outline of the certification approach. Given some value of k and a model C, C' is the resulting witness circuit. The coloured area is specific to our approach for k-induction, and the rest corresponds to the general certification flow.

The certification approach is outlined in Fig. 5.1. Intuitively, a *witness circuit* is generated from a given value of k (provided by the model checker) and a model (either bit-level or word-level). The witness circuit simulates the original circuit while allowing more behaviours (we formally define it as the *stratified simulation relation*). In practice, the witness circuit would be required to be provided by model checkers as the certificate in hardware model checking competitions. To verify the stratified simulation relation between the two circuits, three conditions are generated as SAT formulas (listed in Def. 6.3).

We also perform a polynomial-time stratification check on the witness circuit. The check requires that the definition of the reset function is stratified, *i.e.*, no cyclic dependencies between the reset definitions of the variables exist. This is the case for all hardware model checking competition benchmarks. Even though cyclic definitions have been the subject of study in several papers [84, 100, 122], they are usually avoided due to the complexity of their analysis and subtle effects on semantics.

The approach in [137] can handle cyclic resets but at the cost of QBF quantification, and thus [137] not being able to be efficiently adapted to the context of word-level verification. Furthermore, the witness circuit includes an inductive invariant which serves as a proof certificate, which is verified by another three SAT checks as defined in Def. 5.3 ($\varphi_{init}, \varphi_{cons}, \varphi_{consec}$). Certification is successful if all seven checks pass.

We begin by defining stratified reset functions.

Definition 5.6. (Dependency graph.) Given a set of latches L and a set of reset functions $R = \{r_l \mid l \in L\}$, the dependency graph G_R has latch variables L as nodes and contains a directed edge (a, b) from a to b iff $a \in vars(r_b)$ and $r_b \neq b$.

Latches with undefined reset value are common in applications. We simply set $r_b = b$ for some uninitialised latch b in such a case (as in AIGER and BTOR) to avoid being required to reason about ternary logic or partial functions. Thus the syntactic condition "$r_b \neq b$" in the last definition simply avoids spurious self-loops in the dependency graph for latches with undefined reset values.

Definition 5.7. (Stratified resets.) Given a set of latches L, and a set of reset functions $R = \{r_l \mid l \in L\}$. R is said to be stratified iff G_R is acyclic.

Definition 5.8. (Stratified circuit.) A circuit $C = (I, L, R, F, P)$ is said to be stratified iff R is stratified.

The stratification check can be done in polynomial time using Def. 5.7 and it is enforced syntactically in the two hardware description formats AIGER and BTOR2.

Definition 5.9. (Stratified simulation.) Given two stratified circuits C and C', where C' combinationally extends C. There is a stratified simulation between C' and C iff,

1. $r_l(L) \equiv r'_l(L')$ for $l \in L$, "reset"
2. $f_l(I, L) \equiv f'_l(I, L')$ for $l \in L$, and "transition"
3. $P'(I, L') \Rightarrow P(I, L)$. "property"

In essence, the crucial change here compared to the combinational simulation definition in [137] is the reset condition, whose simplification was possible under the stratification assumption. The above three conditions are encoded into SAT/SMT formulas ($\varphi_{reset}, \varphi_{trans}, \varphi_{prop}$ in Fig. 5.1) which are then checked by a solver for validity. In the rest of the paper, we simply refer to the stratified simulation relation as simulation relation.

Theorem 5.10. *Given two circuits $C = (I, L, R, F, P)$ and $C' = (I', L', R', F', P')$, where C' simulates C. If C' is safe, then C is also safe.*

Proof. We assume C' is safe such that $R'(L') \wedge U'_i \Rightarrow \bigwedge_{j\in[0,i]} P'(I'_j, L'_j)$ holds for all $i \geq 0$. To show that C is safe, we do a proof by contradiction by assuming the safety check in C fails for some m such that $R(L) \wedge U_m \wedge \neg P(I_m, L_m)$ has a satisfying assignment s and we fix this assignment. We can then construct a satisfying assignment for $R(L) \wedge R'(L')$ (according to Def. 6.3) as under the stratification assumption for resets, $R'(L')$ is guaranteed to be satisfiable with L_0 being a subset of L'_0. This is the case as the stratification of the reset function R' does not allow any cyclic dependencies that could potentially make R' unsatisfiable. Moreover, by the transition check in Def. 6.3, the unrolling U'_m of C' is also satisfiable with the transition function F applied on the projected (or shared) components of both circuits. Since the rest of the latches in L' follow a transition function, they are also satisfiable (transition functions guarantee that there is always a successor state for all states). Therefore the initialised unrolling $R'(L'_0) \wedge U'_m$ is satisfiable. Furthermore, by our assumption, $\bigwedge_{i\in[0,m]} P'(I'_i, L'_i)$ holds. By Def. 5.5, projected latches of C' stay the same as L_i for all $i \in [0, m]$; thus by Def. 6.3 we have that $\bigwedge_{i\in[0,m]} P(I_i, L_i)$ holds. Therefore we reach a contradiction. □

Next, we introduce the witness circuit construction. This is similar to the construction in [137] but differs in several details, *e.g.*, the reset function definition is stratified and significantly simplified compared to [137].

Definition 5.11. (Witness circuit.) Given a circuit $C = (I, L, R, F, P)$ and an integer $k \in \mathbb{N}^+$, its witness circuit $C' = (I', L', R', F', P')$ is defined as follows:

1. $I' = I$ (also referred to as X^{k-1}),
2. $L' = L^{k-1} \cup \cdots \cup L^0 \cup X^{k-2} \cup \cdots \cup X^0 \cup B$ where,
   - $L^{k-1} = L$, the other variables sets are copies of I and L respectively with the same variable domains.
   - $B = \{b^{k-1}, \cdots, b^0\}$ are Booleans.
3. R' :
   - for $l \in L^{k-1}, r'_l = r_l(L^{k-1})$.
   - for $l \in L^0 \cup \cdots \cup L^{k-2} \cup X^0 \cup \cdots \cup X^{k-2}, r'_l = l$.
   - $r'_{b^{k-1}} = \top$.
   - for $i \in [0, k-1), r'_{b^i} = \bot$.
4. F' :
   - for $l \in L^{k-1}, f'_l = f_l(I', L^{k-1})$.
   - $f'_{b^{k-1}} = b^{k-1}$.
   - for $i \in [0, k-1), l^i \in (L^i \cup X^i \cup \{b^i\}), f'_{l^i} = l^{i+1}$.
5. $P' = \bigwedge_{i \in [0,4]} p_i(I', L')$ where
   - $p_0(I', L') = \bigwedge_{i \in [0,k-1)} (b^i \rightarrow b^{i+1})$.
   - $p_1(I', L') = \bigwedge_{i \in [0,k-1)} (b^i \rightarrow (L^{i+1} \simeq F(X^i, L^i)))$.
   - $p_2(I', L') = \bigwedge_{i \in [0,k)} (b^i \rightarrow P(X^i, L^i))$.
   - $p_3(I', L') = \bigwedge_{i \in [1,k)} ((\neg b^{i-1} \wedge b^i) \rightarrow R(L^i))$.
   - $p_4(I', L') = \quad b^{k-1}$.

Here we extend a given circuit to a witness circuit, which has k copies of the original latches and inputs, and additional k latches of B that we refer to as the initialisation bits. We refer to the $\{k-1\}th$ as the most recent, and the 0th as the oldest. Intuitively the most recent copy unrolls in the same way as the original circuit, with the older copies copying the previous values of the younger copies. When all initialisation bits are $\top$, we say the machine has reached a "full initialisation" state.

Lemma 5.12. *Given a circuit C with reset function R and its witness circuit C' with reset function R'. If R is stratified, then R' is also stratified.*

Proof. We assume R is stratified. By Def. 6.16. 3, the initialisation bits in B all have constant initial values, and the copies of latches and inputs are uninitialised. Therefore they have no incoming edges in $G_{R'}$ and we can remove them without removing any cycles in $G_{R'}$. The remaining latches L^{k-1} have the same reset functions as in C, which by our assumption, are stratified. Therefore R' is stratified. □

We now prove the stratified simulation relation between C' and C.

Theorem 5.13. *Given a circuit C and its witness circuit C'. C' simulates C.*

Proof. The inputs stay unchanged in the witness circuit, as well as the latches L (which are mapped to L^{k-1}), therefore C' combinationally extends C. By Def. 6.16.3, for $l \in L^{k-1}, r'_l = r_l(L^{k-1})$ and by Lemma 5.12 R' is stratified thus satisfiable; therefore the reset condition in Def. 6.3 is fulfilled. By Def. 6.16.4, $f'_l = f_l(I', L^{k-1})$ for $l \in L^{k-1}$. Thus the transition condition passes. The property condition follows as p_4 and p_2 of P' together imply P. □

Next we prove that under the assumption that C' stratified simulates C, we have that C is k-inductive iff C' is 1-inductive. We proceed by first considering the BMC check.

Lemma 5.14. *If the BMC check for the unrolling of length $k-1$ passes in C, the BMC check also passes for the unrolling of length 0 in its witness circuit C'.*

Proof. Assume $U_{k-1} \wedge R(L_0) \Rightarrow \bigwedge_{i \in [0,k)} P(I_i, L_i)$ such that the initialised unrolling of C is safe. We do a proof by contradiction by assuming the BMC check fails in C' such that $R'(L'_0) \wedge \neg P'(I'_0, L'_0)$ has a satisfying assignment s. We can then construct a satisfying asignment for $R(L_0)$ by using the mapping $L_0 = L_0^{k-1}$. By our assumption of the BMC check passing in C, we have $P(I_0, L_0)$. According to the reset functions in Def. 6.16, $b_0^{k-1} = \top$, and the rest of initialisation bits are set to $\bot$. The premise of $p_2(I'_0, L'_0)$ is only satisfied for b_0^{k-1}, and with the same assignment satisfying $P(X_0^{k-1}, L_0^{k-1})$, $p_2(I'_0, L'_0)$ is also satisfied. Lastly, the premise of $p_3(I'_0, L'_0)$ is only satisfied for $\neg b_0^{k-2} \wedge b_0^{k-1}$, and since $R(L_0^{k-1}), p_3(I'_0, L'_0)$ is satisfied. Therefore we have $P'(I'_0, L'_0)$, and thus a contradiction is reached. □

Note that all that is proved here is that if C is safe for k time steps then the circuit C' is safe for one time step. This differs from the proof strategy in [137] (Lemmas 3-5), and allows for the simpler reset definition in this paper.

Lemma 5.15. *If the property P is k-inductive in C, the consecution check for the unrolling of length 1 passes in its witness circuit C'.*

Proof. Based on our construction of the witness circuit, there are two possibilities of a state: 1) C' is partially initialised; or 2) all initialisations bits are set. In the following proof, we consider the two cases separately. In both cases, we first assume P is k-inductive in C.

We start by considering the case where L'_0 is partially initialised. Let U'_1 be the unrolling of C', $m \in [1, k)$ be the lowest index such that $b_0^0, ..., b_0^{m-1}$ are set to $\bot$,

while $b_0^m, ..., b_0^{k-1}$ are set to $\top$, which follows from p_0 and p_4. We do a proof by contradiction, and assume there is a satisfying assignment s of the negation of the consecution check formula $U_1' \wedge P'(I_0', L_0') \wedge \neg P'(I_1', L_1')$. Based on $p_3(I_0', L_0')$ we have $R(L_0^m)$. We also have $L_0^{i+1} \simeq F(X_0^i, L_0^i)$ for $i \in [m, k-1)$, based on $p_1(I_0', L_0')$. Furthermore, U_1' implies $L_1' \simeq F'(I_0', L_0')$, and by the transition function construction in Def. 6.16, $L_1^{k-1} \simeq F(X_0^{k-1}, L_0^{k-1})$. Therefore we can construct a satisfying assignment for $U_{k-1} \wedge R(L_0)$ where $I_{i-m} = X_0^i, L_{i-m} = L_0^i$ for all $i \in [m, k)$, and $I_{k-m} = I_1', L_{k-m} = L_1^{k-1}$. The variables in U_{k-1} that are not yet defined have a satisfying assignment since we use transition functions. By our assumption that the BMC check passes in C, we have $P(X_0^i, L_0^i)$ for all $i \in [m, k)$ and $P(I_1', L_1^{k-1})$. We then proceed to prove $P'(I_1', L_1')$ is indeed satisfied. Similar to our proof in Lemma 5.14, based on the definition of transition functions in Def. 6.16, $b_1^i = \top$ for all $i \in [m, k)$ while $b_1^i = \bot$ for all $i \in [0, m)$. Additionally, $X_1^i = X_0^{i+1}, L_1^i = L_0^{i+1}$ for $i \in [0, k)$. The rest of the proof follows the same logic as Lemma 5.14, for showing $P'(I_1', L_1')$ is satisfied. We then reach a contradiction here.

Now we continue to prove the second scenario where all initialisation bits are $\top$. Again, let U_1' be the unrolling of C' with $B_0 = \{b_0^0, ..., b_0^{k-1}\}$ all set to $\top$. We again do a proof by contradiction assuming there is a satisfying assignment s of $U_1' \wedge P'(I_0', L_0') \wedge \neg P'(I_1', L_1')$. By the transition property $p_1(I_0', L_0')$, the components follow the transition function F, such that $L_1^{i+1} \simeq F(X_0^i, L_0^i)$ for all $i \in [0, k-1)$. Similar to our argument above, U_1' implies $L_1^{k-1} \simeq F(I_0', L_0^{k-1})$. We also have $\bigwedge_{i \in [0,k)} P(X_0^i, L_0^i)$ based on $p_2(I_0', L_0')$ and the values of B_0 (*i.e.*, the values of B at timepoint 0). Thus we have a satisfying assignment for $U_k \wedge \bigwedge_{i \in [0,k)} P(I_i, L_i)$ with $L_i = L_0^i \wedge I_i = X_0^i$ for all $i \in [0, k)$ and $I_k = X_1^{k-1}, L_k = L_1^{k-1}$. Based on our assumption that the consecution of C passes, we have $P(I_1', L_1^{k-1})$. Following the same reasoning as in the first case, after one transition, $b_1^i = \top$ for all $i \in [0, k)$, and $X_1^i = X_0^{i+1}, L_1^i = L_0^{i+1}$ for $i \in [0, k-1)$. We will now show $P'(I_1', L_1')$ is satisfied. The sub-property $p_2(I_1', L_1')$ is satisfied as we have proved $P(X_1^i, L_1^i)$ for all $i \in [0, k)$. As U_k is satisfied which implies $L_1^{i+1} \simeq F(X_1^i, L_1^i)$ for $i \in [0, k)$, $p_1(I_0', L_0')$ is preserved. Based on the values of B_1 (*i.e.*, the values of B at timepoint 1), $p_0(I_1', L_1'), p_3(I_1', L_1'), p_4(I_1', L_1')$ are satisfied immediately. We conclude the $P'(I_1', L_1')$ is satisfied and we reach a contradiction. □

Based on Lemma 5.14 and 5.15, we immediately reach the following corollary.

Corollary 5.16. *Given a circuit C and its witness circuit C'. If the property P is k-inductive in C, P' is 1-inductive in C'.*

We continue to prove the reverse direction. Note that we are using the 1-inductiveness of C' in the following Lemma as an assumption instead of the BMC property as used in the Lemma 6 of [137].

Lemma 5.17. *If the property P' is 1-inductive in C', then the BMC check passes in C for the unrolling of length $k-1$.*

Proof. We assume P' is 1-inductive such that the consecution check ($U'_1 \wedge P'(I'_0, L'_0) \Rightarrow P'(I'_1, L'_1)$) and the BMC check ($R'(L_0) \Rightarrow P'(I'_0, L'_0)$) pass. We then do a proof by contradiction by assuming the BMC check in C does not pass, meaning $R(L_0) \wedge U_{k-1} \wedge \neg \bigwedge_{i\in[0,k)} P(I_i, L_i)$ has a satisfying assignment. We fix one such assignment and let $m \in [0, k)$ be the index for which $R(L_0) \wedge U_{k-1} \wedge \bigwedge_{i\in[0,m)} P(I_i, L_i) \wedge \neg P(I_m, L_m)$ is satisfied. We consider two separate cases based on the value of m. In the case where $m > 0$, we construct an assignment such that $U'_1 \wedge P'(I'_0, L'_0) \wedge \neg P'(I'_1, L'_1)$ is satisfied. Let $L_0^{k-m+i} = L_i, X_0^{k-m+i} = I_i, b_0^{k-m+i} = \top,$ for $i \in [0, m), b_0^j = \bot$ for $j \in [0, k-m)$. The assignment of the initialisation bits satisfies p_0 as well as p_4. By Def. 5.2 we have $L_{i+1} \simeq F(I_i, L_i)$ for all $i \in [0, m)$, which satisfies the transition property. As $\bigwedge_{i\in[0,m)} P(I_i, L_i)$ is satisfied, p_2 is also satisfied. Furthermore, p_3 is satisfied as b_0^{k-m} is the oldest initialisation bit that is set, and $L_0^{k-m} = L_0$. The same assignment satisfies $R(L_0)$. Therefore $P'(I'_0, L'_0)$ is satisfied. By our assumption that the consecution check in C' passes, we have $P'(I'_1, L'_1)$. By Def. 6.16, $L_1^{k-1} \simeq F(I'_0, L_0^{k-1})$, thus under our construction $L_1^{k-1} = L_m$. By Def. 6.16, $P'(I'_1, L'_1)$ implies $P(I'_1, L_1^{k-1})$. We let $I'_1 = I_m$, and have $P(I_m, L_m)$ thus the contradiction follows. We then consider the second case where $m = 0$ such that $R(L_0) \wedge U_{k-1} \wedge \neg P(I_0, L_0)$ is satisfied, where an initial state in the original circuit is a bad state. Based on $R(L_0)$ and the fact that R' is stratified, we can construct a satisfying assignment for $R'(L'_0)$ with $L_0^{k-1} = L_0, I'_0 = I_0$ and the rest uninitialised. Under our assumption that the BMC check passes in C', we have $P'(I'_0, L'_0)$, and by p_3 in Def. 6.16 and under the variable mapping, we have $P(I_0, L_0)$. Therefore we reach a contradiction. □

Lemma 5.18. *If the consecution check for the unrolling of length* 1 *passes in the witness circuit C', then the consecution check also passes for the unrolling of length k in C.*

Proof. We assume $U'_1 \wedge P(I'_0, L'_0) \Rightarrow P(I'_1, L'_1)$ holds. We then do a proof by contradiction by assuming that the consecution check for the original circuit fails. Thus there is a satisfying assignment s of the formula $U_k \wedge \bigwedge_{i\in[0,k)} P(I_i, L_i) \wedge \neg P(I_k, L_k)$. Based on s, we have a satisfying assignment for $U'_1 \wedge P'(I'_0, L'_0)$ as follows. Let $X_0^i = I_i, L_0^i = L_i$, and $b_0^i = \top$ for $i \in [0, k)$. Let $X_1^{i-1} = I_i, L_1^{i-1} = L_i, b_1^{i-1} = \top$ for $i \in [1, k]$. We now show this satisfies $L'_1 \simeq F'(I'_0, L'_0)$. Since $X_1^{i-1} = I_i = X_0^i$ and $L_1^{i-1} = L_i = L_0^i$ for all $i \in [1, k)$, the transition relation of copying the components is satisfied. Since s satisfies U_k, by Def. 5.2, it satisfies $L_k \simeq F(I_{k-1}, L_{k-1})$. With $X_0^{k-1} = I_{k-1}, L_0^{k-1} = L_{k-1}$, and $L_1^{k-1} = L_k$, we have $L_1^{k-1} \simeq F(X_0^{k-1}, L_0^{k-1})$, which satisfies the transition definition for the most recent copy. As for the initialisation bits, since all of them are set to $\top$ in both B_0 and B_1, the transition definition of B in Def. 6.16 is satisfied. As a result, U'_1 is satisfied, and we continue to show the same assignment satisfies $P'(I'_0, L'_0)$. Similar to our proof in Lemma 5.14, the values of B_0 satisfy $p_0(I'_0, L'_0)$ and $p_4(I'_0, L'_0)$ immediately. As the premiss of $p_3(I'_0, L'_0)$ is unsatisfied, $p_3(I'_0, L'_0)$ trivially holds. Since U_k is satisfied, by Def. 5.2, we have $L_{i+1} \simeq F(I_i, L_i)$ for all $i \in [0, k-1)$, thus also $p_1(I'_0, L'_0)$. Lastly, since $P(I_i, L_i)$ is satisfied for all

$i \in [0, k)$, the original property is satisfied in every component $P(X_0^i, L_0^i)$, resulting in the satisfaction of $p_2(I'_0, L'_0)$. By our initial assumption, $P'(I'_1, L'_1)$ is satisfied. By Theorem 5.13, we have $P(X_1^{k-1}, L_1^{k-1})$, thus $P(I_k, L_k)$. We reach a contradiction here. We can therefore conclude the consecution check of the original circuit passes. □

We conclude the following corollary according to Lemma 5.17 and 5.18.

Corollary 5.19. *Given a circuit $C = (I, L, R, F, P)$ and its witness circuit $C' = (I', L', R', F', P')$. If P' is 1-inductive in C', then P is k-inductive in C.*

Theorem 5.20. *Given a circuit $C = (I, L, R, F, P)$ and its witness circuit $C' = (I', L', R', F', P')$. P is k-inductive in C iff P' is 1-inductive in C'.*

Theorem 5.20 follows Corollary 5.16 and 5.19 directly.

5.4 Implementation and Experimental Evaluation

We implemented the proposed certification approach into two complete toolkits [38]: CERTIFAIGER for bit-level, and CERTIFAIGER-WL for word-level. We evaluate the performance of our tools against several benchmark sets from previous literature and the hardware model checking competitions and report the experimental results.

5.4.1 Bit-level

We implemented our approach described in the previous section into an open toolkit called CERTIFAIGER, which extends the certification toolkit CERTIFAIGER [137]. Our toolkit takes as inputs a model in AIGER [22] format and a value of k which is given by the model checker, then generates six SAT checks which are verified by a state-of-the-art SAT solver Kissat [21]. Note that the AIGER format only allows stratified resets by default.

We implemented a certificate generator based on the witness circuit construction as described in Def. 6.16. The generated witness circuit is passed to the certifier which consists of three parts which are independent of each other. The stratification check is implemented by the AIGER parser [22]. The simulation checker generates three SAT checks as decribed in Def. 6.3 (the reset check, the transition check, and the property check). The inductive invariant checker generates another three SAT checks (the initiation check, the consistency check, and the consecution check). Note that the inductive invariant checker is part of the general certification flow and is supposed to be not only for k-induction but also for various other model checking algorithms. Here we reuse the inductive invariant checker from CERTIFAIGER. The certification is successful when all SAT checks are unsatisfiable.

For benchmarks we used the TIP suite from [58] and benchmarks from the Hardware Model Checking Competition 2010 [20]. The performance of CERTIFAIGER++ was evaluated against CERTIFAIGER on both sets of benchmarks. All experiments were

performed on a workstation with an Intel® Core™ i9-9900 CPU 3.60GHz computer with 32GB RAM running Manjaro with Linux kernel 5.4.72-1.

To determine the speedups of the new implementation proposed in this paper, we performed experiments on the same sets of the benchmarks used in [137]. The results are reported in Table 5.1. There are significant overall gains in the initiation checks (φ_{init}) as well as the reset checks (φ_{reset}). For the initiation check which checks the invariant holds in all initial states, the performance improvement is largely due to the simplification of the reset functions in the new witness circuit construction. Interestingly, the SAT solving time for the new reset check is close to zero, which checks the equality of the reset functions between the shared set of latches and the latches in the original circuit. The explanation for such small execution time is that the reset functions for this set of benchmarks are $\perp$ for all latches, thus making the SAT checks rather trivial.

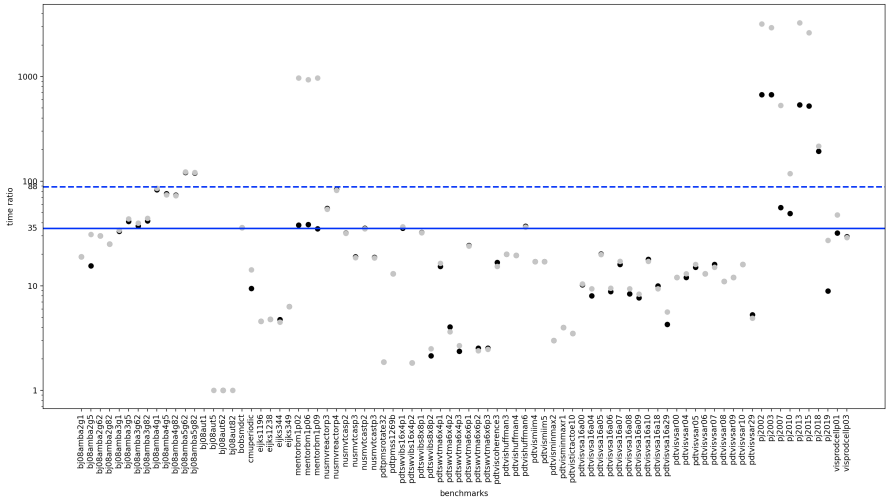

Figure 5.2: Bit-level: the experimental results of the HWMCC 2010. The benchmark names are shown on the x-axis. The time ratio on the y-axis is calculated by computing certification time divided by model checking time. The black dots in the graph are the results obtained from CERTIFAIGER and the grey dots are from CERTIFAIGER. The straight line and the dashed line are the calculated means for CERTIFAIGER and CERTIFAIGER respectively.

The results in Fig. 6.6 demonstrate that CERTIFAIGER in general is much faster than CERTIFAIGER during the overall certification process. We plotted the time ratios of the total certification time and the model checking time (ran on the model checker McAiger [15]) against the HWMCC benchmarks. Compared to CERTIFAIGER, CERTIFAIGER achieved overall speedups of 2.46 times. As we can see from the plot, especially for the instances with certification time greater than 500 seconds, the new implementation significantly improved the certification performance. We observe performance gains

Table 5.1: Summary of certification results for the bit-level TIP suite. Columns report the benchmark names, and the time (in seconds) used for each SAT check by CERTIFAIGER (t_1) and CERTIFAIGER (t_2) respectively.

	φ_{init}		$\varphi_{consist}$		φ_{consec}		φ_{trans}		φ_{prop}		φ_{reset}	
Benchmarks	t_1	t_2	t_1	t_2	t_1	t_2	t_1	t_2	t_1	t_2	t_1	t_2
c.periodic	7.78	0.06	0.06	0.06	56.82	56.29	0.15	0.14	0.05	0.05	84.04	0.00
n.guidance$_1$	0.19	0.01	0.01	0.01	3.73	3.79	0.12	0.12	0.01	0.01	1.21	0.00
n.guidance$_7$	4.09	0.02	0.02	0.02	18.40	18.17	0.12	0.12	0.02	0.02	25.22	0.00
n.tcasp$_2$	0.17	0.01	0.01	0.01	2.64	2.68	0.23	0.23	0.01	0.02	1.79	0.00
n.tcasp$_3$	0.11	0.01	0.01	0.01	1.82	1.70	0.23	0.26	0.02	0.02	1.01	0.00
v.prodcell$_{12}$	2.35	0.03	0.03	0.03	59.05	59.22	0.12	0.12	0.03	0.03	8.48	0.00
v.prodcell$_{13}$	0.22	0.01	0.01	0.01	2.99	2.99	0.12	0.12	0.01	0.01	0.20	0.00
v.prodcell$_{14}$	0.64	0.02	0.02	0.02	13.69	13.69	0.12	0.12	0.02	0.02	1.45	0.00
v.prodcell$_{15}$	2.22	0.02	0.03	0.03	32.66	32.28	0.12	0.12	0.02	0.02	2.26	0.00
v.prodcell$_{16}$	0.01	0.01	0.01	0.01	1.19	1.20	0.12	0.12	0.01	0.01	0.06	0.00
v.prodcell$_{17}$	2.34	0.03	0.03	0.03	48.51	48.17	0.12	0.12	0.03	0.03	6.86	0.00
v.prodcell$_{18}$	0.67	0.01	0.01	0.01	8.67	8.78	0.12	0.12	0.02	0.02	0.79	0.00
v.prodcell$_{19}$	1.66	0.02	0.02	0.03	31.98	31.78	0.12	0.12	0.03	0.03	3.73	0.00
v.prodcell$_{24}$	3.32	0.04	0.04	0.04	112.12	115.18	0.12	0.12	0.04	0.04	17.64	0.00

in most benchmarks, as the previous performance bottleneck for certain benchmarks is the QBF solving time for the reset check. For other instances, the bottleneck is the SAT solving time for the consecution check, which is also improved due to a simpler reset construction (as part of the inductive invariant).

5.4.2 Word-level

We further lifted the method to certifying word-level model checking by implementing an experimental toolkit called CERTIFAIGER-WL written in Nim. CERTIFAIGER-WL follows the same architecture design as CERTIFAIGER and uses Boolector [116] as the underlying SMT solver. All models and SMT encodings are in BTOR2 [116] format, which is the standard word-level model checking format used in hardware model checking competitions. Upon invocation the tool first generates a witness circuit according to the given value of k; then six SMT formulas from the inductive invariant check and the stratified simulation check as described in Section 6.5. The resulting SMT formulas are verified by Boolector. The stratification check is implemented in the BTOR2 parser. The certification passes if all six SMT formulas are proven unsatisfiable. In the case of $k = 1$, no witness circuit is generated and the inductive invariant checker is executed on the original model.

In order to evaluate our implementation and its scalability, we consider two sets of experiments. First of all we consider the Counter example, in which there is a 500-bit counter doing exact-modulo checks with initial value 0, and an enabler which resets the counter value to 0. The counter increments by 1 at each timestamp until it reaches a modulo bound m (then resets). The property encodes the fact that the counter value does not reach some bad state b, which is then k-inductive with $k = b - m + 1$. For the experiments, we fixed the modulo bound at 32 and scaled the inductive depth up to 1000.

We ran all benchmarks on the word-level model checker AVR [70] to get the values of k. Fig. 5.3 shows the experimental results obtained with CERTIFAIGER-WL under the same setting as Section 5.4.1. We record model checking time and certification time separately to show comparison. Interestingly, the certification time is much lower than the model checking time as can be seen in the diagram, meaning certification is at a low cost. As the value of k increases, on average the certification time is proportionally lower.

In Table 5.2 we report the experimental results obtained on a superset of the hardware model checking competition 2020 [121] benchmarks[1]. To select the benchmarks presented, we first ran AVR with a timeout of 5000 seconds. We display the results in Table 5.2 that are of particular interest with a running time of more than ten seconds (there are 7 instances with $k = 1$ which were certified and solved under 0.2s). We observe that the certification time is much lower than model checking time. Including certification would increase the runtime of AVR on the model checking benchmarks

[1] All benchmarks will be published in the artifact.

by less than 6%. As expected, the consecution check as the most complicated formula among the six takes up the majority of the running time.

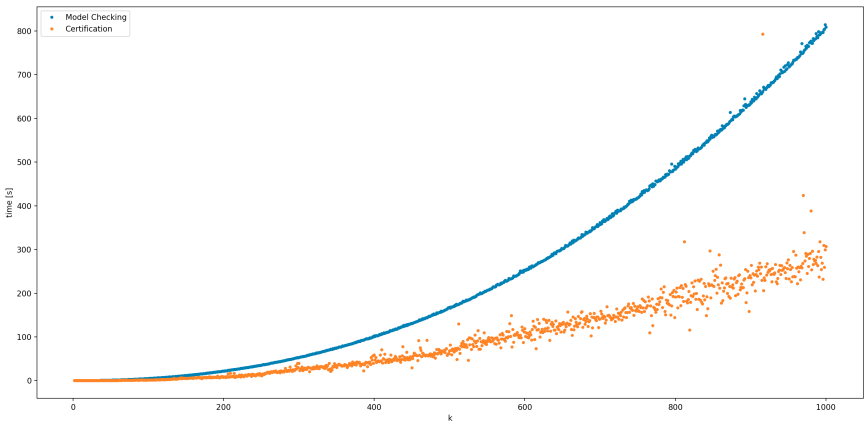

Figure 5.3: Word-level: model checking vs. certification time for the Counter example with increasing values of k. The certification time is significantly smaller than the model checking time.

5.5 Conclusion and future work

We have presented a new certification framework which allows certification for k-induction to be done by six SAT checks and a polynomial-time check. We further lifted our approach from bit-level to word-level, and implemented our method in both contexts. Experimental results on various sets of benchmarks demonstrate the effectiveness and computational efficiency of our toolkits. Time needed for word-level certification is significantly smaller than the time needed for word-level model checking. This makes adding certification for word-level model checking a very interesting and viable proposition. The removal of the QBF quantifiers has reduced the theoretical complexity of the problem compared to [137] and also reduced the overall runtime overhead of the certification.

We leave to future work how to certify k-induction with simple paths constraints. Additionally, we plan to obtain formally verified certificate checkers by using theorem proving. Finally, how to certify liveness properties is another important avenue of further research.

Table 5.2: Summary of certification results word-level benchmarks from the HWMCC20. Columns report the benchmark names, the value of k, the size of the model (measured in number of instructions) and the generated witness, the model checking time, and certification time (in seconds). Additionally we list the time Boolector took to solve the consecution check, as well as the ratio of model checking vs. certification time. We only list the consecution check (Consec.) here as it takes up the majority of the certification time.

Benchmarks	k	#model	#witness	ModelCh.	Certifi.	Consec.	Ratio
paper_v3	256	35	12801	10.25	1.14	0.90	0.11
VexRiscv-regch0-15-p0	17	2149	43077	10.31	4.04	3.29	0.39
zipcpu-pfcache-p02	37	1818	105874	13.95	4.40	2.73	0.32
zipcpu-pfcache-p24	37	1818	105874	14.35	4.49	2.83	0.31
zipcpu-busdelay-p43	101	950	145466	15.29	6.14	3.86	0.40
dspfilters_fastfir_second-p42	15	6732	115388	16.11	14.80	12.96	0.92
zipcpu-pfcache-p01	41	1818	117434	18.33	6.34	4.47	0.35
dspfilters_fastfir_second-p10	11	6732	84348	24.56	9.76	8.44	0.40
zipcpu-busdelay-p15	101	950	145466	58.17	8.18	5.89	0.14
qspiflash_dualflexpress_divfive-p120	97	3100	394412	63.58	22.07	14.58	0.35
zipcpu-pfcache-p22	93	1818	267714	166.07	23.66	19.06	0.14
VexRiscv-regch0-20-p0	22	2149	55862	240.50	16.76	15.76	0.07
dspfilters_fastfir_second-p14	15	6732	115388	354.01	21.27	19.44	0.06
dspfilters_fastfir_second-p11	21	6732	161948	627.69	46.88	44.30	0.07
dspfilters_fastfir_second-p45	17	6732	130908	1 094.11	30.14	28.06	0.03
VexRiscv-regch0-30-p1	32	2150	81464	1 444.47	83.38	81.95	0.06
dspfilters_fastfir_second-p43	19	6732	146428	2 813.61	58.02	55.69	0.02

Chapter 6

Towards Compositional Hardware Model Checking Certification

Submitted

Authors Emily Yu, Armin Biere and Keijo Heljanko.

Acknowledgement This work is funded by FWF project W1255-N23, the LIT AI Lab funded by the State of Upper Austria, and Academy of Finland project 336092.

Abstract In this paper, we revisit and formalize temporal decomposition, which is one of the most basic, widely-used and effective preprocessing techniques in hardware model checking. The main contribution is a certification framework for hardware model checking using temporal decomposition as preprocessing. Our approach enables generation of a single inductive invariant in a compositional way using inductive invariant certificates provided by existing certifying model checkers on the result of preprocessing a model through temporal decomposition. We implement and evaluate the method on hardware model checking competition benchmarks. The experiments confirm the effectiveness of temporal decomposition. The proposed certification approach makes it feasible to generate a generic proof for model checking and preprocessing.

6.1 Introduction

The study of compositional reasoning for safety-critical systems can be traced back to a few decades ago [45]. Compositional model checking breaks the model checking procedure down into several smaller problems, thus enabling faster and more efficient verification. For example, preprocessing techniques are widely used in current industry in combination with standard model checking algorithms.

Among these, temporal decomposition [36] is considered important in industrial hardware model checking [32, 37]. It computes an over-approximation of reachable states to efficiently find a set of transient signals that stabilize to their constant values during any possible execution. A sequence of transformations is applied to the design under verification, including time-shifting it from the reset states and elimination of transient logic. The verification problem thus consists of verifying that the property holds within the time frame the design has been shifted, and that the transformed circuit is safe. For the latter an existing model checker that can provide certificates is employed, such as k-induction [124], symbolic model checking using BDDs [33], and IC3/PDR [29].

While progress in verification using compositional reasoning continues, the number of certifiable approaches are limited [25, 137]. One central objective of verification is to develop a *standardised* method to generate machine-checkable proofs for certifying model checking [137]. This is especially crucial in safety-critical industrial environments, as a faulty processor design can be extremely costly for a hardware manufacturer. Even though a number of single-engine model checkers are able to generate proofs, a key difficulty in this area is to produce a single generic certificate for complex verification pipelines. Furthermore, as in SAT solving [78, 81, 83], proof generation becomes rather involved when using preprocessing techniques, as the proofs from the preprocessors need to be lifted to proofs for the original model checking problem. This problem is exacerbated by the fact that the certificate given by a model checker can be more complex than a simple inductive invariant [25, 137]. For these reasons there is currently no viable industrial-strength model checker that provides certificates.

In this paper, we make a contribution toward this direction by revisiting temporal decomposition and developing a novel, practical, compositional framework for certifying the model checking result of the base engine and the employed preprocessing technique in a single proof. The distinguishing feature of our approach is to generate a *single witness circuit* as a certificate for the entire verification procedure, while related work [59, 73, 111] relies on conceptually more complex deductive frameworks or has not been applied to industrial relevant hardware model checking problems that we are targeting. Different from [2,59,73,111] as well as [92] which aims at handling more expressiveness, our work focuses on efficiently certifying *safety properties* which is the most prominent part in hardware model checking competitions (HWMCCs) and arguably in an industrial setting too. In addition to that, others have focused on either verifying model checkers themselves or lifting it to theorem proving [61, 62, 135]. Similar problems have also been addressed in the context of software verification [12, 13].

The resulting certificate in our framework can be checked via six simple SAT checks and a polynomial check following the certification flow established in [25], thus reducing

trusting a PSPACE hard model checking flow into verifying a simple certificate circuit in co-NP. Our compositional approach breaks the formal proofs into manageable parts while enabling more certifiable techniques. For demonstration we focus on the k-induction algorithm as the base engine for the transformed circuit that provides a more complex certificate, as well as a BDD-based model checker. The reason is that our proposed flow requires model checkers to provide certificates for models with reset functions, which is technically rather involved to add to existing complex multi-engine model checkers, but a feature we want to encourage to be implemented.

Based on this, we have implemented a prototype certifying hardware model checker CHMC that performs temporal decomposition and provides a certificate that can be verified using a SAT solver. We also present two optimisations for detecting transient logic in a circuit. The experiments show our tool is able to solve 29 (out of 818) additional instances enabling temporal decomposition with the k-induction base engine, and an additional of 39 instances using the BDD backend. Our method can produce certificates for all instances solved by the model checker, and effectively verify them.

6.2 Background

This paper extends the set of certifiable model checking techniques thus adopts the certificate format from [25, 137].

We assume a set of Boolean variables $\mathcal{V}$. A literal l is either a variable $l \in \mathcal{V}$ or its negation. We denote $\mathbb{B}(\mathcal{V})$ as the set of all Boolean functions over $\mathcal{V}$ conveniently represented and with formulas (Boolean expressions). A *cube* is a non-contradictory set of literals. For $f \in \mathbb{B}(\mathcal{V})$ we write $f|_l$ to denote the formula after replacing all occurrences of l in f with $\top$ and all occurrences of $\neg l$ with $\bot$. This naturally extends to cubes (interpreted as conjunction of literals). We also write $vars(f)$ to denote the variables that occur in f.

In the following we use a symbolic representation of hardware circuits matching the notation of [25] summarised below. This definition has the advantage of being highly compatible with the AIGER format [22] used for hardware model checking in practice. Note that we use "$\Rightarrow$" for semantic implication, "$\rightarrow$" for syntactic implication and "$\equiv$" for semantic equivalence cf. [137].

Definition 6.1 (Circuit [137])**.** A circuit is a tuple $C = (I, L, R, F, P)$ where I is a set of Boolean *input* variables, L is a set of Boolean *latch* variables, $R = \{r_l(L) \in \mathbb{B}(L) \mid l \in L\}$ is a set of *reset functions*, $F = \{f_l(I, L) \in \mathbb{B}(I, L) \mid l \in L\}$ is a set of *transition functions*, and $P(I, L) \in \mathbb{B}(I, L)$ is the *property* formula.

We write $R(L') = \bigwedge_{l \in L'} (l \simeq r_l(L))$ for some $L' \subseteq L$, where "$\simeq$" is used for syntactic equivalence [51]. Furthermore L_i denotes a copy of L in the temporal direction, thus $U_m = \bigwedge_{i \in [0,m)} (L_{i+1} \simeq F(I_i, L_i))$ denotes an unrolling of length m.

Definition 6.2 (Stratified circuit [25])**.** Given a circuit $C = (I, L, R, F, P)$ with $R = \{r_l \mid l \in L\}$. The dependency graph G_R has latch variables L as nodes and contains a

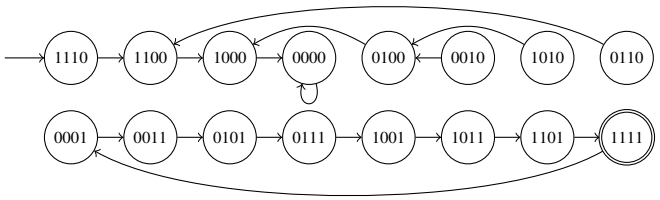

Figure 6.1: State space of a 4-bit shift counter.

directed edge (a, b) from a to b iff $a \in vars(r_b)$ and $r_b \neq b$. The circuit is stratified iff G_R is acyclic.

Definition 6.3 (Stratified simulation [25])**.** Given two stratified circuits $C = (I, L, R, F, P)$ and $C' = (I, L', R', F', P')$ where $L \subseteq L'$. The circuit C is simulated by C' iff: (i) $r_l(L) \equiv r'_l(L')$ for $l \in L$; (ii) $f_l(I, L) \equiv f'_l(I, L')$ for $l \in L$; and (iii) $P'(I, L') \Rightarrow P(I, L)$.

Definition 6.4 (k-induction [137])**.** Given a circuit C, P is *k-inductive* in C iff, (i) $U_{k-1} \wedge R(L_0) \Rightarrow \bigwedge_{i \in [0,k)} P(I_i, L_i)$; and (ii) $U_k \wedge \bigwedge_{i \in [0,k)} P(I_i, L_i) \Rightarrow P(I_k, L_k)$.

We make use of k-induction [124] formulated as a combination of BMC check and a consecution check. It generalizes the concept of checking an *inductive invariant* which is equivalent to 1-induction where the invariant is simply the property.

6.3 Overview

Temporal decomposition helps to simplify model checking by removing parts of the circuit that are only needed for initialisation. Using it as a preprocessing technique, the model checking problem is decomposed into smaller sub-problems. A series of transformation is applied to the circuit by time-shifting it and removing transient signals found via an over-approximation of the reachable states. We begin with an example to show that it can be quite useful and complementary to model checking techniques such as k-induction.

Example 6.5 (Shift Counter)**.** Figure 6.1 shows the state transition diagram of the considered circuit, i.e., a shift counter over 4 bits, with an initial state 1110, where the least significant bit controls the operational mode and never changes. When this bit is set to 0, a left shift is performed; if it is set to 1, the other bits operate as a three-bit binary counter.

Consider the property that at least one bit is zero. As the diagram shows, it is 8-inductive (as the single bad state with all bits one is not reachable, but the longest path only having it as last state has 8 states). In the more general case, with an arbitrary large number of bits n, the inductive depth k is exponentially large (2^{n-1}) in n, for

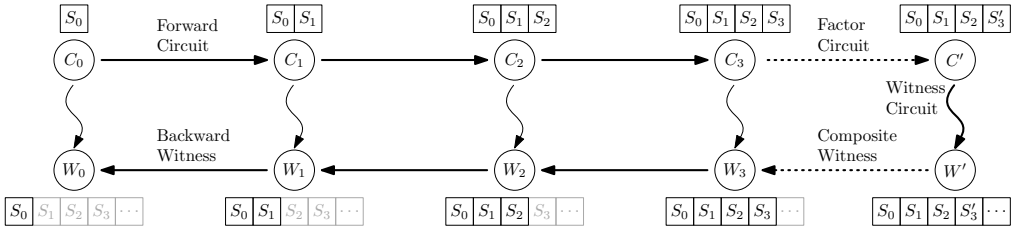

Figure 6.2: An outline of our certification framework (for duration of 3).

a k-induction-based model checker. The size of the certificate for k-induction using only the approach presented in [25] without temporal decomposition would then be in $\mathcal{O}(k \cdot n)$ and thus exponential too.

However, if we consider only the reachable part of the state space, all bits are transient signals as they all eventually stabilise to zero. Therefore we can use temporal decomposition to simplify the design by time-shifting it and removing transient logic. The time-shifting depth (later we call it the *duration*) is linear in n as it grows linearly in the number of bits. In this particular scenario, the model checking problem for the terminal part becomes trivial. We only need a BMC check for the initial $n - 1$ time frames. This leaves us with the problem: how to certify model checking with temporal decomposition?

The overall certification approach we propose is outlined in Fig. 6.2. For this example suppose we have a set of transient signals found that stabilise to constant values with duration 3. In practice the time-shifting namely *circuit-forwarding* of a design is implemented implicitly by computing the successor states originating from reset, however, with the objective of producing elegant and compositional proofs, we construct intermediate circuits forwarded by one transition only.

At each forwarding, the circuit gets unrolled by one time step from reset. For the sequence of circuits $C_0, ..., C_3$, a bounded model checker is used to verify that all initial states are good states. The last forwarded circuit along the pipeline (C_3) is simplified with transient elimination to obtain the factor circuit C'_3, which is given to a base model checker (e.g., k-induction, BDD or IC3/PDR) that also produces a certificate (i.e., a witness circuit [25]). We now construct a composite witness circuit certifying both the preprocessing algorithm for the transients and the safety property. We then build a sequence of backward witness circuits while adding the BMC check for the initial states each time. At each step W_i is a witness circuit of C_i. In the end, we get W_0 as the final witness circuit with an inductive invariant for the entire procedure.

The final witness can be checked by an external proof checker [25]. If the check passes, the original circuit is guaranteed to be safe. It is thus not necessary to trust the correctness of the presented framework nor its implementation. The formal proofs provided in the following sections serve to show that if the original circuit is safe, a valid certificate can be produced.

6.4 Temporal Decomposition

As one of our contributions we revisit temporal decomposition by defining a precise formalism and proving its correctness, which is an improvement over the theory in [36]. We show that our method is complete and will provide a valid certificate whenever temporal decomposition is employed.

In practice, temporal decomposition uses ternary simulation [123] to find transient signals [36]. As generalization we define *cube semantics* and the notion of *cube simulation*, that subsumes ternary simulation as well as other optimisations. We make use of this in our implementation (see Section 7.1).

Definition 6.6 (Cube simulation)**.** Given a circuit $C = (I, L, R, F, P)$, then a cube simulation $c_0, \cdots, c_\delta, \cdots, c_{\delta+\omega}$ for $\delta, \omega \in \mathbb{N}$ is a sequence of cubes over latch variables L such that,

- $R(L) \Rightarrow c_0$.
- For $i \in [0, \delta + \omega)$ we require validity of $c_i \wedge (L' \simeq F(I, L)) \Rightarrow c'_{i+1}$ (L' and c'_{i+1} denote primed copies).
- In addition, it is called a *cube lasso* iff $c_{\delta+\omega} \wedge (L' \simeq F(I, L)) \Rightarrow c'_\delta$.

Note that for $\delta = 0$ and $\omega = 0$, it would simply be a self-loop. This symbolic definition of cube simulation could be implemented directly by a symbolic engine. However, ternary simulation is more appropriate for industrial benchmarks [36].

For brevity we omit (the natural) formal definitions here, but remark that a "bounded version" of cube simulation is subsumed by model checking and in particular ternary simulation does not yield more transients than those produced by Boolean constraint propagation in the SAT solver. As a result, for SAT based bounded model checkers (and therefore k-induction-based model checkers on satisfiable instances), there is hardly any gain using temporal decomposition.

Proposition 6.7. *Bounded model checking subsumes bounded cube simulation.*

Under cube simulation, transient signals can be found by identifying those that stay constant from a certain point onwards along the cube lasso.

Definition 6.8 (Transients via cube lasso)**.** Given a circuit C and a cube lasso $c_0, \cdots, c_\delta, \cdots, c_{\delta+\omega}$. A set of transient signals T is a cube over latches that satisfies $c_i \Rightarrow \bigwedge_{l \in T} l$ for $i \in [\delta, \delta + \omega]$.

For the purpose of temporal decomposition, we could always time-shift the circuit by δ, however, it can make model checking of the final circuit easier to reduce this value as much as possible. For a set of transients found via cube lasso, we formally define the *unstable duration* of a circuit. It is the smallest value for which the last transient becomes constant.

Definition 6.9 (Unstable duration)**.** Given a cube lasso $c_0, \cdots, c_\delta, \cdots, c_{\delta+\omega}$ and a set of transients T, the unstable duration $d \leq \delta$ is the lowest index that satisfies $c_i \Rightarrow \bigwedge_{l \in T} l$ for $i \in [d, \delta + \omega]$.

For the rest of the paper, we simply refer to it as duration.

By taking the disjunction of all cubes from the duration onwards, we get an over-approximation of all the reachable states in the time-shifted circuit.

Definition 6.10 (Loop invariant)**.** Given a cube lasso $c_0, \cdots, c_\delta, \cdots, c_{\delta+\omega}$, and a set of transients T with a duration d. The loop invariant ϕ_T is defined as:

$$\phi_T = \bigvee_{i \in [d, \delta+\omega]} c_i.$$

An immediate observation is that the loop invariant is simply the inductive invariant that implies the stabilised transients in the time-shifted circuit by the duration.

To formalize "time shifting" we introduce the notion of *stable variables* which have the same values throughout the entire execution and do not appear in any other transition function nor in the property.

Definition 6.11 (Stable variable)**.** Given a circuit $C = (I, L, R, F, P)$, the set $\Lambda \subseteq L$ is a set of stable variables if for all $l \in \Lambda$, the following holds:

- $f_l = l$;
- for all $l' \in L \backslash \{l\}, l \notin vars(f_{l'})$;
- $l \notin vars(P)$.

This simple purely syntactic definition of stable variables is the weakest condition that allows us to avoid a spurious exponential blowup during circuit forwarding (Def. 6.12). Note that here identifying stable variables is a simple polynomial time check. Replacing it with stronger conditions, such as cone-of-influence reduction [43], after every forward circuit construction would potentially yield a smaller certificate, which we leave to future work.

A forward circuit is constructed based on a given circuit by copying the set of active variables (i.e., non-stable), and updating the resets of the original active variables to one transition ahead using oracles (uninitialised latches) instead of inputs. The formal definition is given below.

Definition 6.12 (Forward circuit)**.** Given a circuit $C = (I, L, R, F, P)$ with a set of stable variables $\Lambda \subseteq L$ (we refer to $A = L \backslash \Lambda$ as the set of active variables). The forward circuit $C' = (I, L', R', F', P)$ is defined as follows:

- $L' = L \cup I' \cup A'$ where I' and A' are copies of I and A.
- R' :

  - For $l \in \Lambda \cup A', r'_l = r_l$.
  - For $l \in I', r'_l = l$.
  - For $l \in A, r'_l = f_l(I', A')$.
- F' :
  - For $l \in L, f'_l = f_l(I, L)$.
  - For $l \in I' \cup A', f'_l = l$.

An alternative to Def. 6.12 is to simply copy all latches when forwarding a circuit, however, the resulting circuit would be exponential in the duration when doing so iteratively.

For the forward circuit, we proceed to simplify it by removing the set of transients that have been found. The resulting simplified circuit (namely *factor circuit*) is formally defined in the following definition.

Definition 6.13 (Factor circuit)**.** Given a circuit $C = (I, L, R, F, P)$ and a set of transients T from cube simulation. Its factor circuit is a tuple $C|_T = (I, L', R', F', P')$ such that,

- $L' = L \backslash vars(T)$.
- $R' = \{r_l|_T \mid l \in L'\}$.
- $F' = \{f_l|_T \mid l \in L'\}$.
- $P' = P|_T$.

An important observation we make is that the inductive depth of the property does not increase after temporal decomposition, which is later confirmed in our experimental evaluation. However, we have already seen in Example 6.2 an exponential reduction in the inductive depth. We formally prove it and summarise it in the following theorem, which follows directly from Lemmas 6.24, and 6.25 (see Appendix).

Theorem 6.14. *Given a circuit C where the property is k-inductive. Let $C|_T$ be the circuit resulting from temporal decomposition of C. Its property is k-inductive.*

6.5 Certification

In this section, we present a compositional certification framework that is complete. Along the model checking procedure with temporal decomposition, a certificate can be automatically generated by the model checker following the format defined in this section.

Example 6.15. Consider the scenario of a delayed clock (Fig. 6.3). The clock has one bit (the second bit in the diagram) that oscillates between zero and one after the enabler bit (first bit) is set to one. There is only one initial state where the clock is set to zero and enabler bit not set. A state is bad if the clock is high without being enabled.

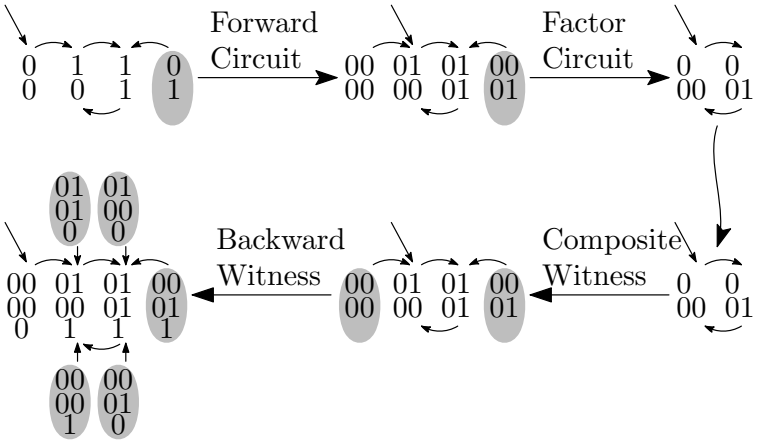

Figure 6.3: An illustration of the overall certification flow for the delayed clock example. Initial states are marked with an additional arrow; bad states are marked gray.

After preprocessing, a base model checker is called for verifying the factor circuit (In Fig. 6.3 the property of the factor circuit is already an inductive invariant thus the factor circuit is simply the certificate). It is required to provide a certificate that is then used to build the final certificate. We give a formal definition of the general certificate format below.

Definition 6.16 (Witness circuit)**.** Given a circuit C, a witness circuit $W = (I, M, S, G, Q)$ of C satisfies the following:

- W simulates C under stratified simulation relation.
- Q is an inductive invariant in W.

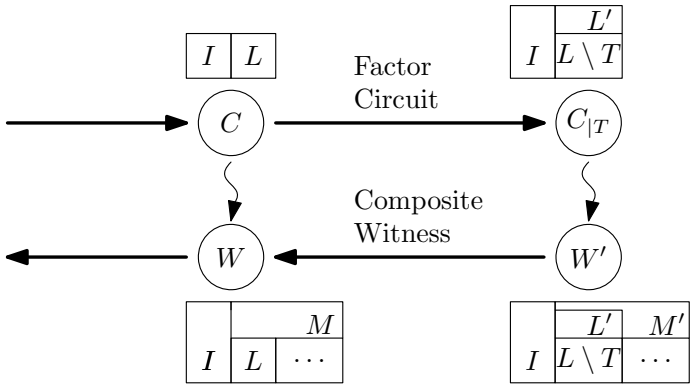

Figure 6.4: Constructing composite witness circuit.

Since the method is compositional, with the witness circuit of the factor circuit produced from the model checker, we combine it with the loop invariant for cube lasso to construct a composite witness circuit that certifies both.

Definition 6.17 (Composite witness circuit)**.** Given a stratified circuit $C = (I, L, R, F, P)$ with the loop invariant ϕ_T for the set of transients T, and its factor circuit $C|_T = (I, L', R', F', P')$ with its witness circuit $W' = (I, M', S', G', Q')$. The composite witness circuit $W = (I, M, S, G, Q)$ is constructed as follows:

- $M = L \cup (M' \backslash L')$.
- S : for $l \in L, s_l = r_l$; for $l \in M' \backslash L', s_l = s'_l$.
- G : for $l \in L, g_l = f_l$; for $l \in M' \backslash L', g_l = g'_l$.
- $Q = \phi_T(L) \wedge Q'(I, M')$.

Intuitively, in the composite witness circuit we add back the previously eliminated transients. The new property simply combines the loop invariant from cube simulation and the property of the witness circuit of the factor circuit, which is an inductive invariant in the composite witness circuit.

Theorem 6.18. *Given a stratified circuit C, its factor circuit $C|_T$ with an inductive invariant ϕ_T, a witness circuit W' of $C|_T$, and the composite witness circuit W. Then W is a witness circuit of C.*

We now introduce *backward witness circuit* built based on a given witness circuit for one backward step, as illustrated in Fig. 6.5. Intuitively, at each backward step, the backward witness circuit certifies the BMC check for the initial states of the corresponding circuit C while maintaining and delaying the behaviours of the given witness circuit W' by one step.

This is achieved by using one bit b which becomes constant true exactly one step after initialisation. At reset, W has the same values as C, with the additional latches holding the reset values from W'; once b is set, it operates as W'. Recall that when constructing the forward circuit, an additional copy of inputs and active latches is added for copying the initial values of those in the given circuit. Since input values are non-deterministic, this needs to be recovered when constructing W such that the inputs are copied to ensure matching values.

Definition 6.19 (Backward witness circuit)**.** Given a stratified circuit $C = (I, L, R, F, P)$, and its forward circuit $C' = (I, L', R', F', P)$. Let $W' = (I, M', S', G', Q')$ be the witness circuit C'. The backward witness circuit $W = (I, M, S, G, Q)$ is defined as follows:

1. $M = M' \cup \{b\}$.
2. $S = \{s_l \mid l \in M\}$ such that:
   - For $l \in L, s_l = r_l$.
   - For $l \in M' \backslash L, s_l = s'_l$.
   - $s_b = \bot$.

3. $G = \{g_l \mid l \in M\}$ such that:
   - For $l \in L, g_l = f_l$.
   - For $l \in M' \backslash L', g_l = ite(b, g'_l, s'_l)$.
   - For $l' \in L' \backslash L, g_{l'} = ite(b, l', l)$ where l is the literal with the same index in $I \cup A$ as l' in $I' \cup A'$ $(= L' \backslash L)$.
   - $g_b = \top$.

4. $Q = \bigwedge_{i \in [0,3]} q_i$, where
   - $q_0 = P(I, L)$.
   - $q_1 = b \rightarrow Q'(I, M')$.
   - $q_2 = \neg b \rightarrow R(L)$.
   - $q_3 = \neg b \rightarrow S'(M' \backslash L)$.

We can thus obtain a witness circuit that certifies (i) the property holds at reset for forward circuits and (ii) the transformed property holds in the factor circuit. In an iterative manner eventually we can construct a witness circuit for the entire pipeline illustrated in Fig. 6.2. Here the property consists of four subproperties: q_0 is the original property; q_1 states that the property from W' needs to hold when b is set; q_2 states that b is false only at initialisation; and q_3 states that all latches except those in L need to hold the reset values as in W'.

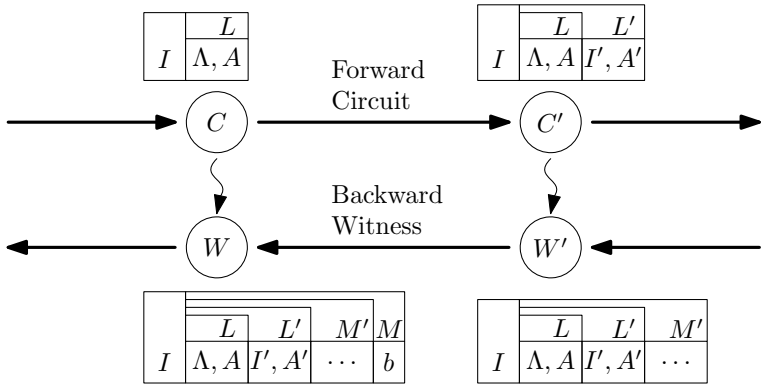

Figure 6.5: An illustration of the latch contents of circuits. C is a time-shifted circuit with latches of Λ (stable variables) and A (active variables). We obtain C' by adding a copy of I' and A' which are used for unrolling the initial states. W' is the witness circuit simulating C'. In addition to the common latches L', M' can also include a set of unknown latches (potentially generated from the model checker). W is the backward witness circuit, containing all latches from M' and an additional bit b used for an initialisation step.

By Def. 6.16 the backward witness needs to pass the six SAT checks for the stratified simulation relation and the witness inductiveness, as well as the stratification check of its reset functions. We summarise this in the following theorem.

Theorem 6.20. *Given a stratified circuit C where the property holds in all reset states, the forward circuit C' with its witness circuit W', and the backward witness circuit W as defined in Def. 6.19. Then W is a witness circuit of C.*

The above concludes the description and formal proof of our certification framework. To perform temporal decomposition on a given circuit under verification, we use cube simulation to find a set of transient signals and determine the duration d. The original circuit is then time-shifted by constructing forward circuits iteratively d times. After eliminating transient signals we obtain a simplified circuit that is then given to a base model checker for verifying the simplified safety property while producing a witness circuit. We construct a final witness circuit by building backward witness circuit d times again in an iterative manner. This final witness circuit serves as a single certificate for both preprocessing and backend model checking. It can be easily verified by an external certifier.

6.6 Implementation and Experimental Evaluation

We implemented the method proposed in Section 6.4 and 6.5 in the certifying model checker CHMC [40]. Our model checker CHMC supports two configurations for the backend solver performing certification on simplified models (*i.e.,* factor circuits) after preprocessing: k-induction and BDD. As factor circuits use reset functions, we extended the k-induction-based open-source model checker MCAIGER [16] to support reset functions and also extended the tool presented by the authors of [25] to generate witness circuits. For the BDD-based configuration, we used the simple BDD-based model checker AIGTRAV[1] developed at JKU Linz which already supports reset functions. We reencoded the convergent BDD to an AIG encoding the membership in the set of reachable states, which is simply the inductive invariant. The witness circuit is the factor circuit having the inductive invariant as the new property. Our model checker CHMC then performs witness composition and backwarding to generate the final witness verified by CERTIFAIGER++ [38].

In principle our approach naturally works for any model checker that produces a certificate (e.g., IC3/PDR, interpolation), however, currently the available implementations for these do not support reset functions nor do they follow the pre-defined certificate format. It seems non-trivial to do so for certain model checkers. Note that a simple constraint-based construction to eliminate reset functions does not solve the problem, as the produced certificate needs to be lifted for the overall certification. It is however natural to modify symbolic model checkers to support reset functions, which we strongly encourage for extending the certification capability to a larger set of model checking tools.

[1] Source code was provided to us directly by the author of AIGTRAV.

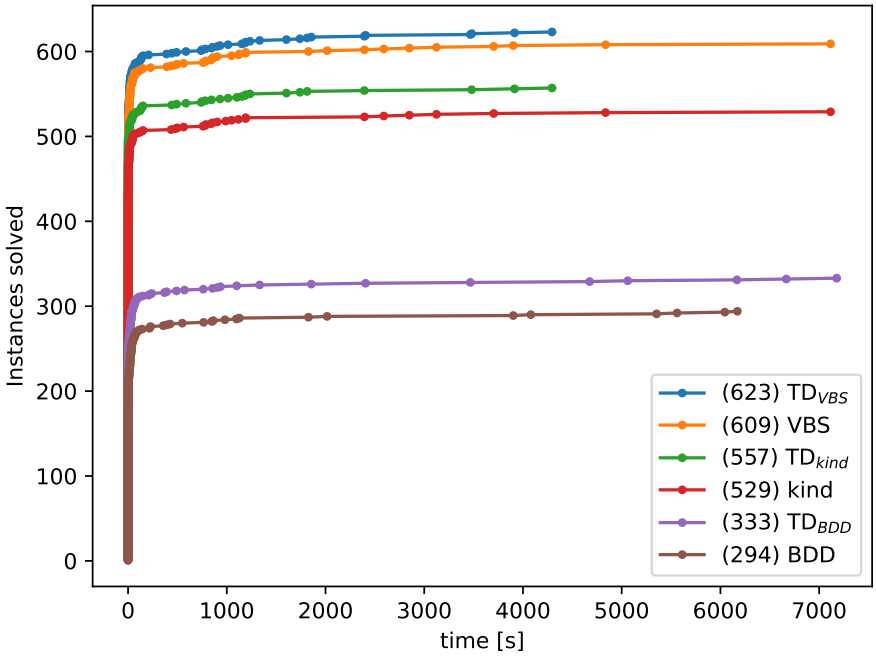

Figure 6.6: Comparison for temporal decomposition with k-induction (TD_{kind}) and BDDs (TD_{BDD}) and the base engines on HWMCC10 instances. We show the results of the best performing solver for each instance with temporal decomposition (TD_{VBS}) as well as the best performing solver without it (VBS).

State-of-the-art model checkers such as ABC [32] use additional preprocessing techniques and are multi-engines for efficient verification, which are hard to separate from each other thus making it difficult to certify a complete model checker with a multitude of preprocessing algorithms. However, we have shown how the certification of an important preprocessing technique, namely temporal decomposition, can be certified in a compositional manner. We believe that similar compositional certification techniques can be identified for other preprocessing techniques. In particular the approach used here can be used for any other preprocessing technique subsumed by cube simulation.

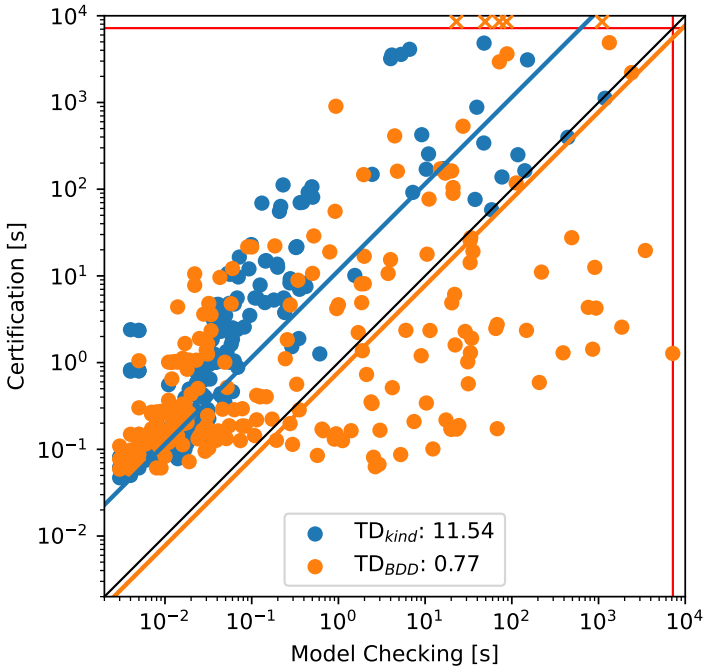

Figure 6.7: Certification vs. model checking time.

To increase trust in our tool we generated 42 million random circuits [22] and checked all produced certificates. We used the HWMCC'10 benchmarks [20] and additionally scaled the shift counter example for further evaluation. All experiments were conducted in parallel on 32 nodes of a cluster. Each node has access to two 8-core Intel Xeon E5-2620 v4 CPUs running at 2.10 GHz (turbo-mode disabled) and 128 GB main memory. We allocate 8 instances to every node with a timeout of 2 hours and memory limit of 16 GB per instance.

We studied the impact of the preprocessing technique on k-induction and BDD-based model checking. The preprocessing (ternary simulation and circuit transformation) terminated within 200ms on all instances. Fig. 6.6 displays the number of instances solved over time by the four configurations. We observe that CHMC with k-induction (TD$_{\text{kind}}$) outperforms the rest. It verified a total of 557 instances, among which 235

Table 6.1: Columns report cycle length (ω), stem length of cube lasso (δ), duration (d), no. transients found (τ), inductive depth of factor circuits (k), original model size ($\#_M^K$), model checking time ($t_M(s)$), certification time ($t_C(s)$), and certificate size ($\#_M^K$). All circuit sizes ($\#_M^K$) are measured in no. thousands of gates including latches and inputs. The certification time is the total of generation time and SAT solving time. We report three mean values on different subsets of benchmarks. Mean TD_{BDD} and TD_{kind} are computed over 223 and 235 instances respectively, where certification checks terminated. Mean $\text{TD}_{\cap}$ concerns the intersection of both sets, from which we display 20 instances that produced the biggest certificate sizes for TD_{kind}. Additionally, we list 5 instances at the bottom for which the BDD-based algorithm succeeded in model checking the simplified circuit, but produced certificates that could not be checked within the time limit.

	Decomposition				Model		TD_{kind}			TD_{BDD}		
	ω	δ	d	τ	k	$\#_M^K$	t_M	t_C	$\#_M^K$	t_M	t_C	$\#_M^K$
Mean $\text{TD}_{\text{BDD}}(223)$	1.9	5.8	0.3	4.9	1.7	4	–	–	–	100.29	77.71	58
Mean $\text{TD}_{\text{kind}}(235)$	2.1	8	0.9	46.3	4.1	7	10.40	120.06	93	–	–	–
Mean $\text{TD}_{\cap}(157)$	2.2	5.7	0.4	5.1	1.7	4	0.19	46.40	35	65.33	46.69	36
pj2002	1	4	2	2	9	37	4.13	3522.82	1757	33.09	25.56	129
pj2003	1	4	2	2	9	37	4.00	3203.89	1757	33.57	27.57	129
cmuperiodic	1	1	1	1	96	2	16.41	173.04	480	0.05	0.52	5
mentorbm1p09	1	39	1	122	2	36	0.45	92.00	331	0.18	22.17	123
mentorbm1or	1	39	1	122	1	36	0.32	21.24	123	0.10	21.56	123
nusmvreactorp4	1	1	1	1	13	1	0.20	12.59	85	7177.65	1.28	15
neclaftp5001	1	10	10	21	0	2	0.05	1.34	79	0.03	1.37	79
neclaftp5002	1	10	10	21	0	2	0.05	1.34	79	0.03	1.40	79
bobsynthand	1	38	1	1	0	19	0.12	7.85	73	0.02	7.79	73
139464p0	1	4	1	2	0	21	0.09	12.05	45	0.06	12.16	45
bj08amba4g5	1	6	0	0	3	14	0.13	69.07	42	219.68	11.06	14
139463p0	1	4	1	2	0	15	0.07	9.72	33	0.04	9.60	33
bj08amba3g62	1	4	0	0	3	10	0.10	22.99	31	1.85	4.92	10
139454p0	1	4	1	2	0	13	0.06	4.73	30	0.06	4.78	30
139462p0	1	4	1	2	0	10	0.06	4.82	23	0.03	4.81	23
139453p0	1	4	1	2	0	9	0.05	3.51	21	0.03	3.58	21
bj08amba5g82	1	5	0	3	0	20	0.07	16.49	20	1.98	16.85	20
139444p0	1	4	1	2	0	8	0.05	3.60	19	0.03	3.62	19
pdtvisvsa16a04	5	7	0	0	2	7	0.05	2.57	18	1.95	147.20	258
pdtvisvsa16a00	5	7	0	0	2	7	0.05	2.62	18	0.02	1.12	7
pdtswvtma6x4p3	1	7	0	0	44	2	77.00	138.26	138	71.75	to	13534
pdtswvtma6x4p2	1	7	0	0	37	2	37.69	76.23	116	86.86	to	13327
nusmvreactorp3	1	1	1	1	4	1	0.04	2.30	24	1100.38	to	6588
pdtvisvsar04	5	7	0	0	2	2	0.02	0.61	8	22.97	to	11074
pdtvisminmax2	1	3	0	0	2	1	0.02	0.14	2	49.29	to	2776

are UNSAT. In particular, compared with base MCAIGER (kind), we gain 29 UNSAT instances. We further inspected the inductive depths of the original circuits and their simplified ones; the results match the claim of Theorem 6.14. In our benchmark set k was reduced for 19 instances. As for the BDD configuration (TD_{BDD}), CHMC performed well on SAT cases, solving an additional of 39 instances. These results clearly demonstrate that simplifying transient logic in circuits can be beneficial for model checking.

We now proceed to evaluate the certification framework on both k-induction and BDD configurations. Table 6.1 summarises the results obtained on certifying the HWMCC10 benchmarks, where we display a subset of interesting instances (sorted by witness size). CHMC was able to produce certificates for all 235 unsatisfiable instances that it was able to model-check. For the k-induction variant, all certificates were successfully verified, with a mean ratio of certification time and model checking time of 11.54, which can be further inspected in Fig. 6.7. This shows the practicality of our method.

On average, duration d tends to be small, which can be partially explained as the starting sequence in designs playing a role. On the BDD variant, the average ratio of certification and model checking time is 0.77. However, 5 instances experienced time-outs when verifying the certificates produced by the BDD backend. Indeed, the BDD algorithm can produce large certificates due to the nature of using the exact set of reachable states as the inductive invariant.

We further investigate the shift counter example by scaling the number of bits (n). The results obtained are reported in Table 6.2. As expected, we found that temporal decomposition significantly simplifies model checking, whereas MCAIGER quickly experiences timeouts. Note that in this particular example, the value of k simply decreases to 0 on the simplified model which becomes trivial after temporal decomposition.

6.7 Conclusion

While certification for model checking has been studied for decades, the number of certifiable approaches are still limited. In this paper we revisited the popular preprocessing technique temporal decomposition by formally defining it and proving its correctness. We further present a certification framework in a compositional manner, that builds a single witness circuit. The approach was evaluated on a wide range of benchmarks. Note that our framework requires model checkers to allow reset functions, that is a feature we would like to encourage existing model checkers to implement. Experimental results show that our approach is quite effective in practice.

In the future we plan to investigate other preprocessing techniques such as retiming [90] and phase abstraction [26]. Furthermore, our simple generic model checking certificate makes it appealing to work on verified proof checkers too.

Table 6.2: Comparison of kind and TD_{kind} for the *Shift Counter*. The n column reports no. of bits. Circuits sizes ($\#_M$) measured in number of gates. Compared with TD_{kind}, as n increases the model checking quickly becomes hard for the k-induction only engine and it experiences time-outs (to), while TD_{kind} was still able to solve the instances within reasonable time.

	Decomposition				Model		kind			TD_{kind}		
n	ω	δ	d	τ	k	$\#_M$	t_M	t_C	$\#_M$	t_M	t_C	$\#_M$
2	1	1	1	2	2	4	0.003	0.051	66	0.011	0.053	23
3	1	2	2	3	4	13	0.003	0.053	250	0.010	0.057	79
4	1	3	3	4	8	22	0.004	0.066	739	0.010	0.057	159
5	1	4	4	5	16	31	0.005	0.093	1974	0.010	0.061	263
6	1	5	5	6	32	40	0.008	0.165	5045	0.013	0.069	391
7	1	6	6	7	64	49	0.031	0.361	12764	0.014	0.069	543
8	1	7	7	8	128	58	0.111	0.987	32883	0.010	0.080	719
9	1	8	8	9	256	67	0.461	3.460	88618	0.011	0.086	919
10	1	9	9	10	512	76	2.409	14.830	255649	0.011	0.084	1143
11	1	10	10	11	1024	85	13.711	59.410	799128	0.012	0.105	1391
12	1	11	11	12	2048	94	86.877	164.968	2698130	0.013	0.108	1663
13	1	12	12	13	4096	103	498.395	501.894	9693060	0.014	0.125	1959
14	1	13	13	14	8192	112	2686.540	2217.410	36368300	0.011	0.108	2279
1000	1	999	999	1000	2^{999}	8986	to	to	to	5.902	1809	11996000

6.8 Appendix

Lemma 6.21. *Given a circuit C, a cube lasso $c_0, \cdots, c_d, \cdots, c_\delta, \cdots, c_{\delta+\omega}$, and a set of transients T with a duration d. Let C' be the resulting circuit of applying circuit forwarding to C iteratively d times. The loop invariant is an inductive invariant of C' for the property $\bigwedge_{l \in T} l$.*

In the following proposition, we state that the conjunction of the factor property $P|_T$ and $\bigwedge_{l \in T} l$ (*i.e.,* the transients are constant) implies the original property. The same follows for reset and transition functions.

Proposition 6.22. $(P|_T \wedge \bigwedge_{l \in T} l) \Rightarrow P$, $(R|_T \wedge \bigwedge_{l \in T} l) \Rightarrow R$, *and* $(F|_T \wedge \bigwedge_{l \in T} l) \Rightarrow F$.

Lemma 6.23. *Given a stratified circuit C. Its forward circuit C' is also stratified.*

Proof. Based on the reset function definition in Def. 6.12, the reset functions of $\Lambda \cup A'$ are the same as in $R(L)$, which are acyclic. The variables in I' are uninitialised, thus do not depend on the rest of the variables. As $R'(A)$ only uses variables distinct from A, we conclude C' is stratified. □

Lemma 6.24. *Given a circuit $C = (I, L, R, F, P)$ where the property is k-inductive. Let $C' = (I, L', R', F', P')$ be the factor circuit. P' is k-inductive in C'.*

Proof. We do a proof by contradiction by assuming P' is not k-inductive in C'. First we consider the case where the BMC check fails in C' (i.e., $U'_{k-1} \wedge R'(L'_0) \wedge \neg \bigwedge_{i \in [i,k)} P'(I_i, L'_i)$ has a satisfying assignment). By Lemma 6.21, the transients stay constant in C, and by Propositions 6.22, we can construct a satisfying assignment for $U_{k-1} \wedge R(L_0) \wedge \neg \bigwedge_{i \in [i,k)} P(I_i, L_i)$. This contradicts our assumption. The second scenario where the consecution check fails in C' follows the same logic. □

Lemma 6.25. *Given a circuit $C = (I, L, R, F, P)$ where the property is k-inductive. Let $C' = (I, L', R', F', P)$ be the forward circuit. P is k-inductive in C'.*

Proof. We provide a proof by contradiction assuming P is not k-inductive in C'. Similarly we assume $R'(L'_0) \wedge U'_{k-1} \wedge \neg \bigwedge_{i \in [0,k)} P(I_i, L_i)$ has a satisfying assignment. By Def. 6.12 the same satisfies $R(L_0) \wedge U_k \wedge \neg \bigwedge_{i \in [1,k]} P(I_i, L_i)$. This implies that a bad state is reachable from the initial states thus contradiction. We then consider the consecution check fails in C' such that $U'_k \wedge \bigwedge_{i \in [0,k)} P(I_i, L_i) \wedge \neg P(I_k, L_k)$ has a satisfying assignment. By Def. 6.12 the transition function stays the same for the common latches thus the same assignment satisfies the formula $U_k \wedge \bigwedge_{i \in [0,k)} P(I_i, L_i) \wedge \neg P(I_k, L_k)$. Therefore we have reached a contradiction. □

6.8.1 Proof of Theorem 2

Given a stratified circuit C, its factor circuit $C|_T$ with an inductive invariant ϕ_T, a witness circuit W' of $C|_T$, and the composite witness circuit W. W is a witness circuit of C.

Proof. We show that W simulates C. The factor circuit $C|_T$ is obviously stratified as C is stratified by Def. 6.13. The witness circuit W' is also stratified. The reset functions of L remain the same in W, and $R(L)$ do not contain latches in $M' \setminus L'$. Thus W is stratified. Based on Def. 6.17, the common latches $L \subseteq M$ and inputs are the same in both circuits. As the reset functions and transition functions of L remain the same, this satisfies the reset check and transition check. Since W' is a witness of $C|_T$, we have $Q' \Rightarrow P'$, where $P' \equiv P|_T$ by Def. 6.13. Since ϕ_T is an inductive invariant for transients and by Lemma 6.21, we have $\phi_T \Rightarrow \bigwedge_{l \in T} l$. By Proposition 6.22, we have $Q \Rightarrow P$. Therefore W simulates C.

We then show the BMC check passes ($S(M) \Rightarrow Q(I, M)$) by assuming that $S(M)$ and thereby $R(L) \wedge S'(M' \setminus L')$ holds, and shall proceed to the conclusion $\phi_T(L) \wedge Q'(I, M')$, by Def. 6.17. We begin with $R(L)$ and may deduce $\phi_T(L)$ immediately since Lemma 6.21 applies to C and the circuit is in its reset state.

From this point on, our attention will be on the subset of L that is free of transients, L' by Def. 6.13. We have $R(L') = R'(L')$ again by Def. 6.13, and then $R'(L') = S'(L')$ since W' simulates $C|_T$, which by Def. 6.3 implies the resets to be the same. We combine this with the second half of our initial assumption to arrive at $S'(M')$. The witness circuit W' is therefore at reset, and its inductive invariant $Q'(I, M')$ holds. We conclude with $Q(I, M)$.

We make the observation that $G(I, L) \equiv F(I, L)$, $F(I, L \setminus T) \equiv F'(I, L \setminus T) \equiv G'(I, L \setminus T)$ by Def. 6.16 and Def. 6.13. The latter together with Def. 6.17 gives us $G(I, M') \equiv G'(I, M')$. Assume a fixed satisfying assignment for $V_1 \wedge Q(I_0, M_0)$, where V_1 is the unrolling of length 1 in W. By the observation above, this also satisfies U_1 and V_1' which are the unrollings of C and W' respectively. By Def. 6.17 the assignment satisfies $\phi_T(L_0) \wedge Q'(I_0, M_0')$ which are inductive invariants in C and W' respectively. We thus have $\phi_T(L_1) \wedge Q'(I_1', M_1')$. As we have shown the inductiveness of Q with the BMC check and consecution check, together with the simulation relation, we conclude W is a witness circuit of C.

□

6.8.2 Proof of Theorem 3

Given a stratified circuit C where the property holds in all reset states, the forward circuit C' with its witness circuit W', and the backward witness circuit W as defined in Def. 6.19. Then W is a witness circuit of C.

Proof. First we show W is stratified. By Def. 6.19, $S(L) \equiv R(L)$, therefore stratified and does not depend on latches outside L, and $S(\{b\})$ is independent of other variables. Furthermore, for the rest of latches $M' \setminus L$, the reset functions are the the same as in

W' therefore stratified. We conclude that W is stratified. By Def. 6.19, $L \subseteq M$ and the inputs I stay the same in W. Based on Def. 6.19, for the common latches L, their reset function and transition function are the same as in C, and $q_0 \equiv P$. Therefore W simulates C.

We proceed to prove that the BMC check passes in W, i.e., $S(M) \Rightarrow Q(I, M)$. Since b is false at reset, q_1 is trivially satisfied. The reset also directly implies $R(L)$ and $S'(M' \backslash L)$, which gives us q_2 and q_3. We have q_0 since the property holds at reset in C. We then move on to prove the consecution check passes ($V_1 \wedge Q(I_0, M_0) \Rightarrow Q(I_1, M_1)$). We consider two cases based on the value of b_0 and begin with the case $b \equiv \top$. By Def. 6.19, since b always transitions to true, $q_2(I_1, L_1)$ and $q_3(I_1, L_1)$ are trivial. Additionally, $q_1(I_1, M_1)$ implies $Q'(I_1, M_1')$ which by Def. 6.16 implies $P(I_1, L_1)$, as C and C' share the same property. Thus we only need to show $Q'(I_1, M_1')$ holds after one transition to satisfy $q_1(I_1, M_1)$. Consider a fixed satisfying assignment for $V_1 \wedge Q(I_0, M_0)$ and b is true. We show that the same assignment satisfies V_1' (the unrolling of W'). For latches in $M' \backslash L'$ this follows directly from the definition. By Def. 6.16 and Def. 6.12 L has the same transition function in all 4 circuits (Fig. 6.5). The latches in $L' \backslash L$ stay constant which matches Def. 6.12. The rest follows from Def. 6.16. With this, $q_1(I_0, M_0')$ gives us $Q'(I_0, M_0')$ which is an inductive invariant in W'. $Q'(I_1, M_1')$ follows.

Now consider a satisfying assignment where b is false thereby $R(L_0)$ and $S'(M_0' \backslash L_0)$. By Def. 6.12, for the latches in L the transition function matches the reset function in C', thus we get $R'(L_1)$. The latches in $L' \backslash L$ copy the values of $I \cup A$. By Def. 6.12 $R'(I_1')$ holds trivially and $R'(A_1')$ follows from $R(L_0)$. We get $R'(L_1' \backslash L_1)$. Together with the previous result we have $R'(L_1')$, which by Def. 6.3 yields $S'(L_1')$. The reset for the rest of the latches $M' \backslash L'$ follows directly from Def. 6.19 and we get $S'(M_1')$. Since the inductive invariant of W' holds in its reset we conclude with $Q'(I_1, M_1')$.

□

6.8.3 Additional Experimental Results

In this section, we present additional experimental results to further illustrate our findings. Previously, in Figure 6.6, we showcased the overall performance comparison for both satisfiable and unsatisfiable instances. Now, we examine the performance comparison separately for these instances.

As depicted in Figure 6.8, our plot indicates that the usage of temporal decomposition as an additional preprocessing step does not appear to provide significant benefits for k-induction in the case of satisfiable instances. This observation aligns with Proposition 1, which sheds light on the underlying explanation. However, it is worth noting that temporal decomposition demonstrates a remarkable improvement when employed in conjunction with the BDD-based algorithm.

In Figure 6.9, we observe that both algorithms exhibit noticeable benefits from incorporating temporal decomposition as a preprocessing step in the case of unsatisfiable instances. This outcome emphasizes the positive impact of temporal decomposition in enhancing the performance of these algorithms.

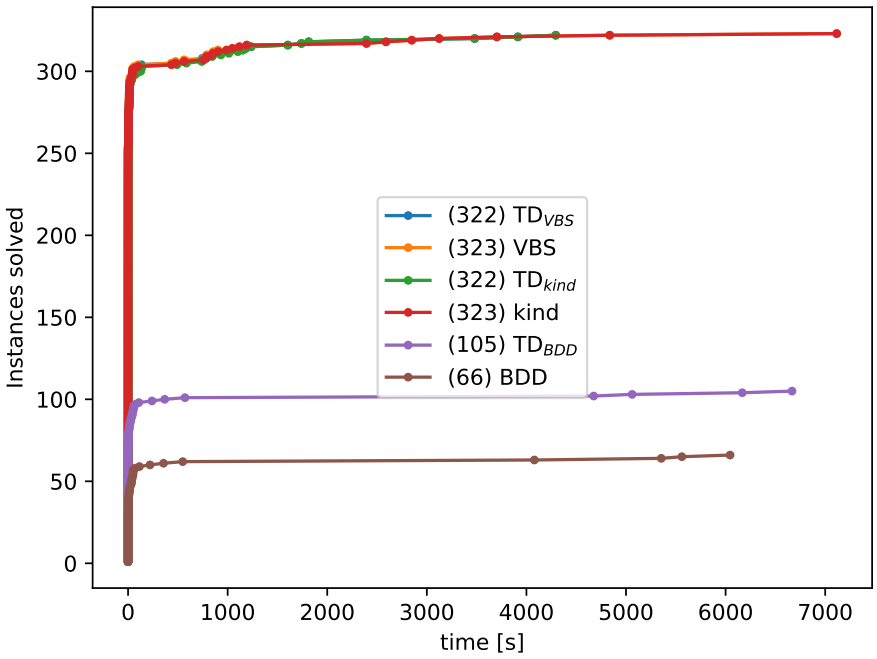

Figure 6.8: Number of model checking problems solved as a function of time for HWMCC10 benchmarks on SAT instances. Temporal decomposition is particularly beneficial for BDD-based model checking on SAT instances.

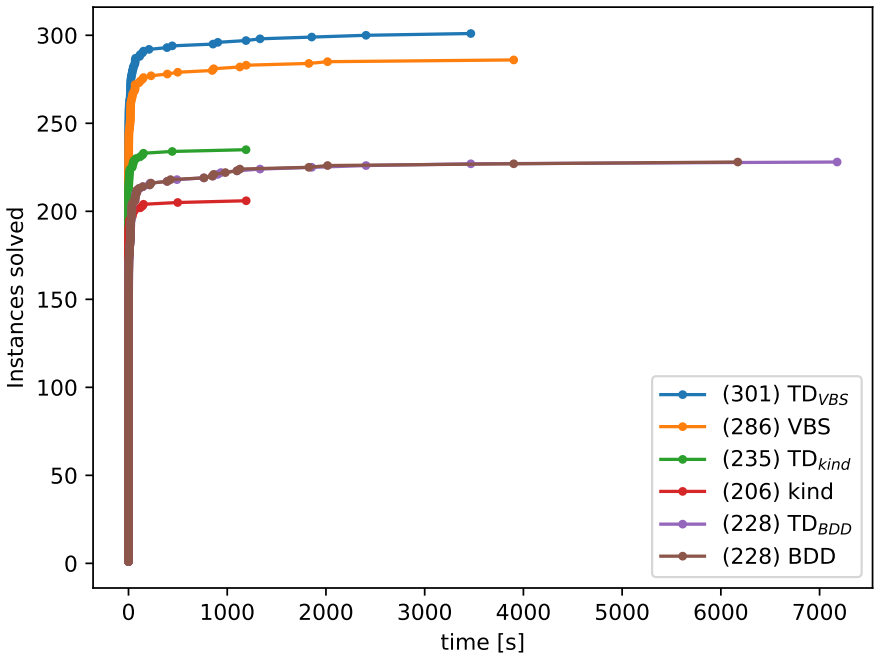

Figure 6.9: Number of model checking problems solved as a function of time for HWMCC10 benchmarks on UNSAT instances. On these instances, temporal decomposition does not seem to help much for the BDD-base engine while TD_{kind} outperforms the rest.

Chapter 7

A Framework for Model Checking against CTLK Using Quantified Boolean Formulas

Published This paper is published in Proceedings of the 7th International Workshop on on Formal Techniques for Safety-Critical Systems (FTSCS 2019), Satellite workshop of ICFEM 2019, pages 127–132, Shenzhen, China, November, 9, 2019.

Authors Emily Yu, Martina Seidl, and Armin Biere.

Acknowledgement This work was supported by the Austrian FWF grant W1255-N23 and the LIT AI Lab funded by the State of Upper Austria.

Abstract We present a novel bounded model checking (BMC) tool chain for multi-agent systems. This framework automatically translates the verification of system models against properties formulated in computation tree logics with epistemic modalities (CTLK) into quantified Boolean formulas (QBFs). Our framework exploits recent QBF technology for solving those verification problems and for certifying the result, making the implementation of a dedicated CTLK solver obsolete. The translation to QBF is based on existing theoretical work and implemented in our novel tool MCMAS$_{\text{qbf}}$ which extends the open-source model checker MCMAS. First experimental results are very promising and indicate the practical feasibility of our approach. Furthermore we provide novel benchmarks to the QBF community.

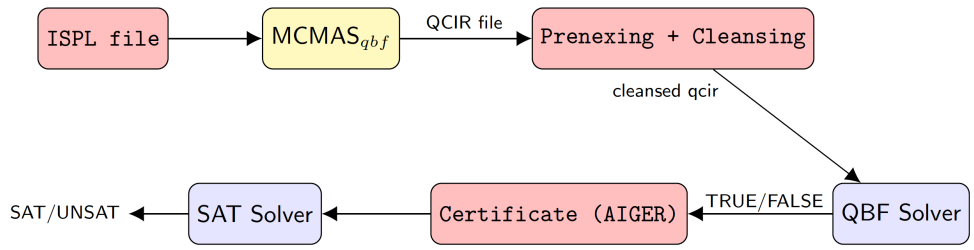

Figure 7.1: The complete MCMAS$_{\text{qbf}}$ tool chain.

7.1 Introduction

Multi-agent systems (MAS) are nowadays applied in various fields to describe complex systems. For example, MAS are used to formalize the interactions of different components that act independently [54]. To verify their correctness, *Computation Tree Logic* with knowledge (CTLK) has been introduced [129]. Besides temporal operators like "**A**lways", "**U**ntil", and "**F**inally", CTLK also includes formulas with knowledge modalities $\mathbf{K}_i\phi$ expressing that "Agent i knows ϕ".

With CTLK it becomes possible to perform model checking for MAS [99]. *Model Checking* [4, 42, 44] is an important technique for verifying safety-critical systems against properties expressed in temporal logics like LTL or CTL. To deal with the so-called *state explosion problem* of model checking, SAT-based bounded model checking (BMC) [19] was introduced. To obtain more compact encodings of BMC problems than possible with SAT, encodings of BMC to *quantified Boolean formulas* (QBFs) have been presented [52]. Such encodings exploit the power of existential and universal quantifiers to avoid duplications of formula parts.

In this paper, we present a fully automatic tool chain for verifying descriptions of multi-agent systems against properties in CTLK. Therefore, we implemented MCMAS$_{\text{qbf}}$ for translating such BMC problems to QBFs building upon the bounded semantics of CTLK introduced in [143].

7.2 The MCMAS$_{qbf}$ Tool Chain

Our tool MCMAS$_{\text{qbf}}$ [104] extends MCMAS [98], an open-source model checker for the verification of multi-agent systems supporting various temporal epistemic logics. We reused the parser of MCMAS to obtain the interpreted system data structures based on which we generate the QBF encodings. We implemented the translation of bounded semantics of CTLK into QBFs based on the theoretical work in [143] which includes both existential and universal fragments of the logic. As an approximation to unbounded model checking, the bounded semantics considers a finite state space where each path in the system is restricted to a length of k. However, in the verification process the search space is extended progressively as the formula is evaluated.

The input of MCMAS$_{qbf}$ is an ISPL file which contains a description of the system and a CTLK formula for the property to be checked. ISPL is an agent-based, modular language based on the interpreted systems [63] formalism commonly used for MAS. Our extension is invoked with parameters

```
-QBFbmc [k] [QCIR-File] [ISPL-File],
```

where k is a value specifying the bound followed by an ISPL file and a QCIR output file for the QBF. Our tool MCMAS$_{qbf}$ is embedded in the tool chain as shown in Figure **??**. It produces QBFs in the most general variant of the QCIR format [85], i.e., in non-prenex form which allows to position quantifiers arbitrarily within a formula. Since there is no state-of-the-art QBF solver that supports this general format, an additional prenexing step is necessary to shift the quantifiers to the front. For example, the formula $\forall x \exists y \phi \wedge \forall a \exists b \psi$ has to be rewritten to $\forall a, x \exists b, y(\phi \wedge \psi)$. Therefore, we implemented a simple tool that performs not only quantifier shifting, but also the translation to the cleansed QCIR format that requires the names of the Boolean variables to be numbers and not strings. Now the QBF can be passed to any QBF solver that is able to process formulas in the cleansed QCIR format. We applied the Quabs [127] that can not only decide the truth value of the formula but also produce certificates. These certificates are And-Inverter-Graphs (AIGs) [22] representing the solution to the BMC problem, and can be checked by a SAT solver for increasing trust in the QBF solver. For this purpose, we use the SAT solver PicoSAT [14].

7.3 Case Study

As a case study, we consider the popular Train-Gate-Controller (TGC) example [129]. In this scenario, there are multiple trains on different tracks and a controller. The tracks intersect at one tunnel which has red-green lights controlled by the controller, and only one train can operate in the tunnel at a time when the light is green. The following code snippet describes this scenario for one train modeled in ISPL:

```
Agent train1
Vars:
state: {wait, tunnel, away};
end Vars
Actions = {enter, leave, nothing};
Protocol:
state = wait: {enter, nothing};
state = tunnel: {leave, nothing};
state = away: {nothing};
end Protocol
Evolution:
state = wait if state = away and Action = nothing;
state = tunnel if state = wait and Action = enter and Environment.Action=
    enter1;
state = away if (state = tunnel and Action = leave and Environment.Action=
    leave1)
or (state=wait and Action=nothing);
end Evolution
end Agent
```

An interpreted system typically contains a set of agents (`train1`, ...) with possible local states (`wait`, `tunnel`, `away`), actions (`enter`, `leave`, `nothing`), as well as protocols and evolution functions for describing the system behavior. The global states are composed of each agent's local states. Further an initial state is also defined in the ISPL description. To translate the model checking problem into QBFs, we firstly need to encode the interpreted system as follows:

- *state space*: $\lceil \log |L_i| \rceil$ Boolean variables are needed for representing the local states L_i of agent i. The same number of variables is needed for the local successor state. The global current state $\mathbf{v} = (v_e, v_1, ..., v_N)$ and the global successor state $\mathbf{v}' = (v'_e, v'_1, ..., v'_N)$ are vectors of local states where N is the number of agents and e refers to the environment.

- *actions*: For the actions, $\sum_{i \in \{e,1..,N\}} \lceil \log |Act_i| \rceil$ Boolean variables are needed.

- *transition relation*: For each agent, the protocol function and evolution function are encoded symbolically using v_i and v'_i. The global transition relation is the composition of protocol and evolution functions based on $\mathbf{v}$ and $\mathbf{v}'$.

We have implemented the encoding presented in [143] and our implementation allows to generate a QBF as a QCIR file which then can be solved and certified by existing QBF solvers. The property holds if the verification result of the QBF solver shows the formula is satisfied, and vice versa.

We verify the following property in our case study: along all paths in the system, it is always the case that if train_1 is in the tunnel then it knows that the other trains cannot be operating in the tunnel at the same time. In CTLK, this property can be expressed as follows:

$$\phi = \mathbf{AG}\ (\ \text{in\_tunnel}_1\ \rightarrow\ \mathbf{K}_{\text{train}_1} \bigvee_{i=2}^{N} \neg \text{in\_tunnel}_i)$$

To evaluate the performance of the tool chain, we ran several experiments on an Intel® Core™ i7-2600 machine with 3.40GHz CPU and 16GB RAM running Ubuntu v18.04.2 (Linux kernel v4.15). We evaluated the bounded model checking problem with different values of k, and in order to test the scalability of the framework, we ran experiments with $5, 8$, and 10 trains (we use N to represent the number of trains).

Table 7.1 reports the results of our case study. The obtained cleansed QBFs in prenex form contain up to $2M$ gates in the QCIR format, while the certificates in AIGER contain only up to $100K$ gates plus the inputs and outputs related to the QBF variables (C in Table 7.1). The solving time t_{total} includes the time for the whole tool chain including the QBF solving time t_{qs} and the time for checking the certificates t_{sat}. While the solving times are quite small, much time is needed for the encoding and the cleansing. Here, many optimizations are possible.

Table 7.1: Experimental results obtained for the Train-Gate-Controller case study.

N	k	ϕ_{qbf} (gates)	C	t_{total}(s)	t_{qs}(s)	t_{sat}(s)
3	5	30616	990	0.961	0.056	0.023
3	10	110971	2945	3.445	0.222	0.088
3	15	252226	5950	8.018	0.673	0.201
3	20	464881	10005	14.942	1.178	0.370
5	5	57695	2354	1.785	0.112	0.050
5	10	206735	7299	6.469	0.487	0.156
5	15	464875	14994	15.057	1.261	0.359
5	20	848615	25439	28.560	2.820	0.685
8	5	100482	5480	3.127	0.226	0.084
8	10	357472	17460	11.299	0.982	0.280
8	15	798762	36240	26.195	2.794	0.627
8	20	1449852	61820	51.409	6.882	1.259
10	5	140217	9747	4.607	0.557	0.114
10	10	495812	30452	16.771	2.247	0.389
10	15	1101507	62707	38.680	6.073	0.871
10	20	1988802	106512	76.313	14.219	1.578

7.4 Discussion

We presented a complete tool chain for solving bounded model checking of multi-agent systems against CTLK specifications using QBF solving technology. First experiments are very promising, allowing us not only to solve the BMC problems but also to obtain quite small certificates from the QBF solvers. Further, this work provides practical benchmarks in the general QCIR format to the QBF community.

As future work, we plan to integrate the model checker with a QBF solver more tightly using an incremental QBF approach to speed up model checking. The translation algorithm can also be optimized further, by for instance picking an arbitrary value of k as a starting point and increase k step-wise in a loop. Furthermore, sub-formulas can be encoded separately then verified, and the verification results can be cached in order to speed up the whole model checking process when applied in a real-world setting.

Chapter 8

Certifying Phase Abstraction

8.1 Introduction

Preprocessing plays a crucial role in enhancing the efficiency of model checking procedures. Among the array of techniques available, one widely employed approach is the generalized phase abstraction [26] technique. This simple preprocessing methodology is commonly employed in conjunction with basic model checking techniques such as IC3/PDR [29] and k-induction [124].

Similar to temporal decomposition, phase abstraction also relies on the power of ternary simulation to efficiently identify a certain group of latches. But instead of finding transient signals, phase abstraction uses it to find oscillating signals that exhibit clock-like behaviours. These signals are essentially the clocks embedded in the circuit designs. Based on the periodic behaviours of the clock-like signals, an optimal phase number can be computed, which is later used to unfold the original circuit. Phase abstraction then applies Boolean minimization and cone-of-influence reduction on the unfolded circuit to obtain a simplified circuit that can undergo base model checking.

Our recently proposed certificate format allows certification via a few simple SAT checks. In the previous work, we have successfully applied the format to k-induction and temporal decomposition. In this chapter, we revisit phase abstraction and show how to generate a single witness circuit as certificate for this technique.

8.2 Preliminaries

We assume a set of Boolean variables V, and a literal l is either a variable $v \in V$ or its negation. We use cube semantics, where a cube is a non-contradictory set of literals. Let c be a cube over a set of variables L. We write $c(L')$ to denote the resulting cube after replacing the variables in c with variables in L'.

Let f be a Boolean formula over a set of literals L. For $l \in L$ we write $f \setminus \{l\}$ to denote the resulting formula after removing all occurrences of l and $\neg l$ in f. For simplicity, we use the same notation for sets of literals.

We write $f[l \backslash g]$ to denote substituting all occurrences of the literal l in f with g, where g is a formula over L. With an abuse of notation, we write $f[V \backslash G]$ for substituting a set

of literals V with a set of formulas G under the same variable ordering. For the rest of the chapter, we use the same notations as defined in Chapter 6.

8.3 Restricted simulation

The stratified simulation relation defined in [25] allows the given circuit to be extended to a larger circuit. More specifically, the simulating circuit extends the original circuit by having the same set of inputs and more latches. This constrains the witness circuit to always have more latches than the original. After a cone-of-influence reduction, the circuit is typically simplified by removing an unused set of inputs and latches, which entails a potentially smaller certificate. To achieve this, in the following, we revisit this definition in order to allow more behaviours for the extended circuit.

Definition 8.1 (Restricted Simulation)**.** Given two stratified circuits $C = (I, L, R, F, P)$ and $C' = (I', L', R', F', P')$. Let $K = L \cap L'$. Then C' simulates C under restricted simulation relation iff,

1. $R(K) \Rightarrow R'(K)$.
2. For $l \in K$, $f_l(I, L) \equiv f'_l(I', L')$.
3. $P'(I', L') \Rightarrow P(I, L)$.

The restricted simulation relation concerns a set of common latches between two circuits, which can be verified by three simple SAT checks. For the rest of the chapter we simply refer to restricted simulation as simulation.

Theorem 8.2. *Given two stratified circuit $C = (I, L, R, F, P)$ and $C' = (I', L', R', F', P')$, where C' simulates C. If C' is safe, then C is also safe.*

Proof. We assume C' is safe, and provide a proof by contradiction by assuming C is not safe. Let s be an assignment over $L \cup I$ satisfying $U_m \wedge R(L_0) \wedge \neg P(I_m, L_m)$. For any such s there exists an assignment s' over $L \cup I \cup L' \cup I'$ with $s \subseteq s'$ that 1) satisfies U'_m by 8.1.2 and by the fact that every state has a successor according to the definition of transition functions, and 2) satisfies $R'(L'_0)$ by 8.1.1 and the fact that C' is stratified. In other words $U'_m \wedge R'(L'_0)$ is guaranteed to be satisfied by an extension of s. Thus, by assumption $P'(I'_m, L'_m)$ follows which together with 8.1.3 results in a contradiction. □

For example in the case where $K = \emptyset$, Def. 8.1.3 simply implies that $P(I, L) \equiv \top$.

In the following, we define the witness circuit based on the restricted simulation relation, which requires six SAT checks and a polynomial time check for stratification.

Definition 8.3 (Witness circuit)**.** Given two stratified circuits $C = (I, L, R, F, P)$ and $C' = (I', L', R', F', P')$. C' is said to be a witness circuit of C iff,

- C' simulates C under the restricted simulation relation.

- P' is an inductive invariant of C'.

Modifying the simulation relation requires us to revisit the previous certification approaches for k-induction and temporal decomposition. In Chapter 6, we defined the witness circuit to satisfy two conditions: (i) it simulates the given circuit under stratified simulation relation; (ii) it contains an inductive invariant.

Proposition 8.4. *Given two stratified circuits $C = (I, L, R, F, P)$ and $C' = (I', L', R', F', P')$. If C' simulates C under the stratified simulation relation, then C' also simulates C under the restricted simulation.*

Proof. In this case, K is simply equivalent to L. The rest follows immediately. □

Proposition 8.5. *Given a circuit $C = (I, L, R, F, P)$ and its stratified k-witness circuit $C' = (I', L', R', F', P')$ as defined in [25]. Then C' is a witness circuit of C according to Definition 8.3.*

8.4 Phase Abstraction

In this section, as one of the contributions we revisit phase abstraction from a formal perspective and provide a clear and precise definitions for it. The main reason is to apply compositionality to phase abstraction in order to make use of this during certification.

The concept of phase abstraction involves unfolding a circuit multiple times, where each unfolding in the new circuit is equivalent to multiple steps of unfolding in the original circuit. Fig. 8.6 illustrates the flow of phase abstraction. The process begins by using ternary simulation to identify clock-like or oscillating signals and determine the optimal number of phases n. Once the circuit is unfolded n times, factoring is performed by assigning constant values to the clock-like signals in each phase. Next, the property is rewritten by applying some rewriting techniques and the circuit is simplified using cone-of-influence analysis. Finally, the simplified circuit C_4, undergoes a terminal model checking technique such as k-induction.

Cone-of-influence reduction is popular technique used for preprocessing, and it can be tightly integrated with existing model checking algorithms such as IC3/PDR. Phase abstraction also makes use of it.

Definition 8.6 (Cone of influence)**.** Given a circuit $C = (I, L, R, F, P)$, the cone-of-influence of P, denoted $coi(P)$ is defined as the smallest set of inputs and latches such that:

- $vars(P) \subseteq coi(P)$.
- If $l \in L$,
  - for $l \in coi(P), vars(r_l) \subseteq coi(P)$;
  - for $l \in coi(P), vars(f_l) \subseteq coi(P)$.

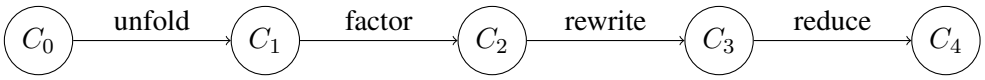

Figure 8.1: Phase abstraction. Base model checking is performed on the circuit C_4.

Clock-like signals are those that transition in a pattern and are periodic. A simplest example is a latch that always oscillates between zero and one. In practice, ternary simulation is often used to find clock-like signals. Similar to Chapter 6, we continue the cube semantics and use cube simulation which subsumes ternary simulation.

Definition 8.7 (Clock-like signal)**.** Given a circuit $C = (I, L, R, F, P)$, a cube lasso $c_0, \cdots, c_\alpha, \cdots, c_{\alpha+\beta}$ and a phase number $n \in \mathbb{N}^+$, a latch $l \in L$ is a clock-like signal iff it satisfies the following, where l^i is a literal of l.

- For some $k^0, k^1 \in \mathbb{N}^+$, $k^0 \cdot n = \alpha; k^1 \cdot n = \beta + 1$.
- $\forall i \in [0, n), \exists l^i \in \{l, \neg l\}, \forall j \in [0, k^0 + k^1) : c_{i+j \cdot n} \Rightarrow l^i$.

The unfolded circuit is different from our definition of forward circuit in Chapter 6. A forward circuit is constructed based on a given circuit, with the reset states being one step forward. The unfolded circuit contains n copies of the given circuit, and its property is fast computed n steps at each transition.

Definition 8.8 (Unfolded circuit)**.** Given a circuit $C = (I, L, R, F, P)$ and a phase number $n \in \mathbb{N}^+$. The unfolded circuit $C' = (I', L', R', F', P')$ is defined as follows:

1. $I' = I^0 \cup \cdots \cup I^{n-1}$.
2. $L' = L^0 \cup \cdots \cup L^{n-1}$.
3. $R' = \{r'_l \mid l \in L'\}$:
   a) for $l \in L^0, r'_l = r_l$.
   b) for $i \in (0, n), l^i \in L^i, r'_{l^i} = F(I^i, L^{i-1})$.
4. $F' = \{f'_l \mid l \in L'\}$:
   a) for $l \in L^0, f'_l = f_l(I^0, L^{n-1})$.
   b) for $i \in (0, n), l^i \in L^i, f'_{l^i} = f_{l^i}(I^i, L^{i-1})$.
5. $P' = \bigwedge_{i \in [0,n)} P(I^i, L^i)$.

Definition 8.9 (Unfolded cube lasso)**.** Given a circuit $C = (I, L, R, F, P)$ with a phase number $n \in \mathbb{N}^+$, and its unfolded circuit $C' = (I', L', R', F', P')$. Let $c_0, \cdots, c_\alpha, \cdots, c_{\alpha+\beta}$ be the cube lasso of C. The unfolded cube lasso $c'_0, \cdots, c'_{\alpha'}, \cdots, c'_{\alpha'+\beta'}$ is defined as follows:

- $\alpha' * n = \alpha; \beta' * n + n - 1 = \beta$.

- For $i \in [0, \alpha' + \beta'), c'_i = \bigwedge_{j \in [0,n)} c_{i*n+j}(I'^j_i, L'^j_i)$.

Lemma 8.10. *The unfolded cube lasso is a cube lasso in the unfolded circuit.*

Definition 8.11 (Unfolded loop invariant)**.** Given a circuit $C = (I, L, R, F, P)$ and its unfolded circuit $C' = (I', L', R', F', P')$, and a set of clock-like signals Γ found from the cube lasso $c_0, \cdots, c_m$ of C. Let $c'_0, ..., c'_{m'}$ be the unfolded cube lasso. The unfolded loop invariant ϕ is defined as $\bigvee_{i \in [0,m']} c'_i$.

Lemma 8.12. *Given a circuit $C = (I, L, R, F, P)$ and its unfolded circuit $C' = (I', L, R', F', P')$ with a phase number n. Let Γ be a set of clock-like signals found from the cube lasso $c_0, ..., c_{\alpha+\beta}$ of C. Let Γ^i be a set of literals for the variables in Γ, such that $\Gamma^i = \{l \in c_i \mid var(l) \in \Gamma\}$. The unfolded loop invariant ϕ is an inductive invariant in C' for the property $\bigwedge_{i \in [0,n]} (\bigwedge_{l^i \in \Gamma^i} l^i)$.*

In the above lemma, the property states in essence that the clock-like signals stay constant along the unfolded cube lasso. Each cube in the unfolded cube lasso is a partial assignment to L', which is n copies of the original set of latches L. Therefore we use Γ^i to denote a set of literals that represent the periodic values of the clocks. In the unfolded circuit, the unfolded loop invariant is an over-approximation of the reachable states for which the clock-like signals stay constant.

After unfolding, the clock-like signals simply become constants in the unfolded circuit. We can therefore obtain a simplified circuit by replacing the clock-like signals with constant values.

Definition 8.13 (Factor circuit)**.** Given a circuit $C = (I, L, R, F, P)$ and its unfolded circuit $C' = (I', L, R', F', P')$ with a phase number $n \in \mathbb{N}^+$, and a set of clock-like signals Γ found from the cube lasso $c_0, \cdots, c_{\alpha+\beta}$. The factor circuit $C'' = (I', L', R'', F'', P'')$ is defined as follows:

1. $R'' = \{r''_l \mid l \in L'\}$:
   - For all $l \in \Gamma$, for all $i \in [0, n)$, $r''_{l^i} = \begin{cases} \top & \text{if } c_i \Rightarrow l_i, \\ \bot & \text{if } c_i \Rightarrow \neg l_i. \end{cases}$
   - For all $l \in (L \setminus \Gamma), r''_{l^i} = r'_{l^i}$.
2. $F'' = \{f''_l \mid l \in L'\}$:
   - For all $l \in \Gamma$, for all $i \in [0, n), f''_{l^i} = l$.
   - For all $l \in (L \setminus \Gamma), f''_{l^i} = f'_{l^i}$.

Similar to SAT, there are various rewriting techniques used by model checkers to rewrite the property. Since the property is essentially a Boolean formula of inputs and latches, for example, it can be rewritten in AIGER by re-structuring the and gates.

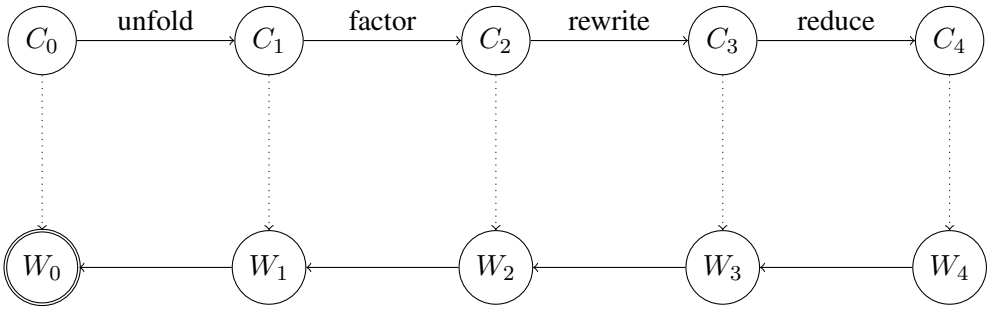

Figure 8.2: Certification for phase abstraction. Base model checking is performed on the circuit C_4, which produces a witness circuit W_4. We construct step-wise to obtain W_0, which is the certificate for the entire model checking procedure.

Definition 8.14 (Rewrite circuit)**.** Given a circuit $C = (I, L, R, F, P)$, the rewrite circuit $C' = (I, L, R, F, P')$ satisfies $P \equiv P'$.

For a given circuit, we apply cone-of-influence reduction to obtain a reduced circuit.

Definition 8.15 (Reduced circuit)**.** Given a circuit $C = (I, L, R, F, P)$. The reduced circuit $C' = (I', L', R', F', P)$ is defined as follows:

1. $I' = I \cap \mathit{coi}(P)$.
2. $L' = L \cap \mathit{coi}(P)$.
3. $R' = \{r_l \mid l \in L'\}$.
4. $F' = \{f_l \mid l \in L'\}$.

8.5 Certification

In this section, we draw inspiration from the compositional framework introduced in Chapter 6 to outline the construction of a certificate for phase abstraction. Following a similar approach, we demonstrate how to build a single certificate that certifies both phase abstraction and the base model checking.

The reduced circuit is the resulting circuit obtained after performing cone-of-influence reduction. In the following theorem, we show that the witness circuit of the reduced circuit serves as a witness circuit for the original circuit too.

Theorem 8.16. *Given a circuit $C = (I, L, R, F, P)$ and its reduced circuit $C' = (I', L', R', F', P')$. The witness circuit of C' is also a witness circuit of C.*

Proof. Let $W' = (X', M', S', G', Q')$ be the witness circuit of C'. First of all, we show that W' simulates C. As $L' \subseteq L, K = L' \cap M'$ is also the common set of latches between C and W'. The resets of C remain stratified. As the reset functions and transition functions of K stay the same, together with Def. 8.15 where $P' = P$, the

three checks in Def. 8.1 are satisfied. We conclude that W' simulates C. By Def. 8.3, Q' is an inductive invariant. Therefore W' is also a witness circuit of C. □

One popularly adopted technique employed by model checkers is rewriting, which serves to simplify properties. The outcome of this process is a circuit with a simplified property that maintains semantic equivalence with the original property. Therefore in our framework, the certificate for the property is still valid for the simplified property. We summarize this in the following theorem.

Proposition 8.17. *Given a circuit C and its rewrite circuit C'. The witness circuit of C' is also a witness circuit of C.*

We define the composite witness circuit to combine the certificates for cube simulation and the factor circuit.

Definition 8.18 (Composite witness circuit)**.** Given a stratified circuit $C = (I, L, R, F, P)$ and its factor circuit $C' = (I', L', R', F', P')$, and the unfolded loop invariant ϕ. Let $W' = (X', M', S', G', Q')$ be the witness circuit of C'. The composite witness circuit $W = (X, M, S, G, Q)$ is defined as follows:

1. $X = I \cup X'$.
2. $M = L \cup (M' \backslash L')$.
3. S :
   - for $l \in L, s_l = r_l$;
   - for $l \in M' \backslash L', s_l = s'_l$.
4. G :
   - for $l \in L, g_l = f_l$;
   - for $l \in M' \backslash L', g_l = g'_l$.
5. $Q = \phi(L) \wedge Q'(X', M')$.

Theorem 8.19. *The composite witness is a witness circuit of C.*

Proof. First we show the composite witness W simulates its corresponding circuit C. W' and C are stratified and R only references latches in L_1 thus no new cyclic dependencies is introduced. Therefore W is stratified too. L is the common set of latches for W and C.By Def. 8.13, the reset and transition functions of L are the same in both circuits, this satisfies the reset check as well as the transition check. Since W' is a witness circuit of C', we have $Q' \Rightarrow P'$ and therefore $Q \Rightarrow P'$. By Proposition 6.22 from Chapter 6, we have $Q \Rightarrow P$. We omit the rest of the proof as it follows the same logic as the proof in Theorem 6.18 in Chapter 6. □

When constructing a witness circuit from the witness of the unfolded circuit W', the resulting witness circuit, namely n-folded witness circuit, contains one copy of the latches from W' . In the following, for demonstration we present the construction for $n = 2$ and generalize it to arbitrary n afterwards.

Definition 8.20 (Two-folded witness circuit)**.** Given a circuit $C = (I, L, R, F, P)$ with a phase number 2, and its unfolded circuit $C' = (I', L', R', F', P')$. Let $W' = (X', M', S', G', Q')$ be the witness circuit of C'. The two-folded witness circuit $W = (X, M, S, G, Q)$ is defined as follows:

1. $X = I^0 \cup X^0$, where I^0 and X^0 are copies of I and X'.
2. $M = I^1 \cup I^2 \cup L^0 \cdots L^2 \cup N \cup X^1 \cup \{b^0 \cdots b^2, e\}$, where $N = M' \setminus L'$, and I^i, L^i are copies of I and L, and X^1 is a copy of X'.
3. $S = \{s_l \mid l \in M\}$:
   a) $s_{b^0} = \top$;
   b) For $i \in [1, 2], b^i = \bot$.
   c) $s_e = \top$.
   d) For $l \in L^0, s_l = r'_l$.
   e) For $l \in N, s_l = s'_l$.
   f) For $l \in (I^1 \cup I^2 \cup L^1 \cup L^2 \cup X^1), s_l = l$.
4. $G = \{g_l \mid l \in M\}$:
   a) $g_{b^0} = \top$.
   b) For $i \in [1, 2], g_{b^i} = b^{i-1}$.
   c) $g_e = ite(b^2, \neg e, e)$.
   d) For $l \in L^0, g_l = f_l(I^0, L^0)$.
   e) For $i \in [1, 2], l^i \in (I^i \cup L^i), g_{l^i} = l^{i-1}$.
   f) For $l^1 \in X^1, g_{l^1} = l^0$.
   g) For $l \in N, g_l = ite(b^2 \wedge e, g'_l(X^1, M' \cap (L^1 \cup L^2 \cup N)), l)$.
5. $Q = \bigwedge_{i \in [0,5]} q^i$ such that:
   a) $q^0 = P(I^0, L^0)$.
   b) $q^1 = b^1$.
   c) $q^2 = \bigwedge_{i \in [1,2]} (b^i \rightarrow b^{i-1})$.
   d) $q^3 = \bigwedge_{i \in [1,2]} (b^i \rightarrow (L^i \simeq F(I^{i-1}, L^{i-1})))$.
   e) $q^4 = \bigwedge_{i \in [1,2]} ((\neg b^i \wedge b^{i-1}) \rightarrow (R(L^{i-1}) \wedge S'(N)))$.

$$\text{f) } q^5 = b^2 \rightarrow ((Q'(X^0, M' \cap (L^0 \cup L^1 \cup N)) \wedge e) \vee Q'(X', M' \cap (L^1 \cup L^2 \cup N)) \wedge \neg e)).$$

In this construction, we introduce an additional bit e as a flipping bit. This bit starts flipping after the circuit is fully initialized. When e is set, the latches from W' transition as in W', otherwise they freeze for one time step.

Theorem 8.21. *Given a stratified circuit $C = (I, L, R, F, P)$ with a phase number of 2, its unfolded cicuit $C' = (I', L', R', F', P')$ with a witness circuit of $C' = (X', M', S', G', Q')$. Let $W = (X, M, S, G, Q)$ be the circuit constructed as in Def. 8.20. Then W is a witness circuit of C.*

Proof. First, we prove the simulation relation. It is clear that S is stratified. Let $K = L \cap M$. By Def. 8.20, for $l \in L^0$, $s_l \equiv r'_l$. Together with Def. 8.8, we have $R(K) \Rightarrow S(K)$. By Def. 8.20, $f_l \equiv g_l$ for $l \in K$. Also by q^0, we have $Q \Rightarrow P$.

Next we show that the BMC check passes for W such that $S(M) \Rightarrow Q(X, M)$. Since only b^0 is set at reset, q^1, q^2, q^3 and q^5 are satisfied. The reset in Def. 8.20 directly implies $R'(L^0)$ and $S'(N)$, and by Def. 8.8 it also satisfies $R(L^0)$, which satisfies q^4. Based on Def. 8.1, $R'(L^0) \Rightarrow S'(M' \cap L^0)$, which together with the stratification of S', results in $S'(M')$. This implies $Q'(X', M')$, based on the BMC check of the inductive invariant Q'. By Def. 8.1, we have $P'(I', L')$, and based on Def. 8.8 this gives us $P(I^0, L^0)$ thus q^0 is also satisfied.

Let V_1 be the unrolling of W and we move on to prove $V_1 \wedge Q(X_0, M_0) \Rightarrow Q(X_1, M_1)$ by providing a proof by contradiction. We assume $V_1 \wedge Q(X_0, M_0) \wedge \neg Q(X_1, M_1)$ has a satisfying assignment, and fix this assignment. We consider two cases based on the values of b_0^i: (i) all b_0^i are set to $\top$; (ii) B_0 is partially initialised, i.e., not all b_0^i set to $\top$. We begin with the first case. Since b^is always transition to $\top$ or b^{i-1}, and under the assumption that all b^is are $\top$, q_1^1, q_1^2 and q_1^4 are immediately satisfied. Def. 8.22, L^0 transitions in the same way as in C, and the rest of the latches are simply copying the values during the transition, thus q_1^3 is satisfied. We move on to consider p_1^5. Based on q_0^5, we get a satisfying assignment for either $Q'(X_0^0, M_0' \cap (L_0^0 \cup L_0^1 \cup N_0)) \wedge \neg e_0$ or $Q'(X_0^1, M_0' \cap (L_0^1 \cup L_0^2 \cup N_0)) \wedge e$. We thus consider two different cases based on the disjunction and the value of e_0.

- Case $e_0 \equiv \top$: as stated in Def. 8.20, when $e_0 \equiv \top$, latches in N transition as in W'. In this case, we already have $Q'(X_0^1, M_0' \cap (L_0^1 \cup L_0^2 \cup N_0))$ satisfied, together with the transition function defined and q_0^3, the same assignment satisfies $e_1 \equiv \bot$ and $Q'(X_1^0, M_1' \cap (L_1^0 \cup L_1^1 \cup N_1))$, which satisfies q_0^5. By Def. 8.1, we have $P'(I_1', L_1')$ which implies $P(I_1^0, L_1^0)$. Therefore p_1^0 is satisfied.

- Case $e_0 \equiv \bot$: similarly, according to Def. 8.20, the latches in N stay the same. As the latches in $I^1 \cup L^1$ and $I^2 \cup L^2$ simply copy the values from their copies with lower indices, $Q'(X_1^1, M_1' \cap (L_1^1 \cup L_1^2 \cup N_1))$ is satisfied. Based on the transition function definition in Def. 8.20, $e_1 \equiv \bot$, thus q_1^5. Lastly, under the same condition, latches in L^0 transition in the same way as in C', the same assignment satisfies $Q'(X_1', M_1')$ that implies $P'(I_1', L_1')$ and thus $P(I_1^0, L_1^0)$.

In either case, we can apply the same assignment to satisfy $Q(X_1, M_1)$ and therefore contradiction. The rest of the proof follows the similar logic. □

We generalize Def. 8.22 in the following definition to show the certificate construction for an arbitrary phase number.

Definition 8.22 (n-folded witness circuit)**.** Given a circuit $C = (I, L, R, F, P)$ with a phase number $n \in \mathbb{N}^+$, and its unfolded circuit $C' = (I', L', R', F', P')$. Let $W' = (X', M', S', G', Q')$ be the witness circuit of C'. The n-folded witness circuit $W = (X, M, S, G, Q)$ is defined as follows:

1. $X = I^0 \cup X^0$, where I^0 and X^0 are copies of I and X'.
2. $M = I^1 \cdots I^m \cup L^0 \cdots L^m \cup N \cup X^1 \cup \{b^0 \cdots b^m, e^0 \cdots e^{n-2}\}$, where $m = 2 \times n - 2$, $N = M' \setminus L'$, and I^i, L^i are copies of I and L, and X^1 is a copy of X'.
3. $S = \{s_l \mid l \in M\}$:
   a) $b^0 = \top$;
   b) For $i \in (0, m]$, $s_{b^i} = \bot$.
   c) For $i \in [0, n-1)$, $s_{e^i} = \bot$.
   d) For $l \in L^0$, $s_l = r'_l$.
   e) For $l \in (I^1 \cdots I^m \cup L^1 \cdots L^m \cup X^1)$, $s_l = l$.
   f) For $l \in N$, $s_l = s'_l$.
4. $G = \{g_l \mid l \in M\}$:
   a) $g_{b^0} = \top$.
   b) For $i \in [1, m]$, $g_{b^i} = b^{i-1}$.
   c) $g_{e^0} = b^m \wedge \neg e^{n-2}$.
   d) For $i \in [1, n-1)$, $g_{e^i} = e^{i-1} \wedge \neg e^{n-2}$.
   e) For $l \in L^0$, $g_l = f_l$.
   f) For $l^1 \in X^1$, $g_{l^1} = l^0$.
   g) For $i \in [1, m]$, $l^i \in (I^i \cup L^i)$, $g_{l^i} = l^{i-1}$.
   h) For $l \in N$, $g_l = ite(e^{n-2}, g'_l(X^1, M' \cap (I^{m-n+1} \cdots I^m \cdots L^{m-n+1} \cdots L^m \cup N)), l)$.
5. $Q = \bigwedge_{i \in [0,5]} q^i$:
   a) $q^0 = P(I^0, L^0)$.
   b) $q^1 = b^1$.

c) $q^2 = \bigwedge_{i\in[1,m]} (b^i \to b^{i-1})$.

d) $q^3 = \bigwedge_{i\in[1,m]} (b^i \to (L^i \simeq F(I^{i-1}, L^{i-1})))$.

e) $q^4 = \bigwedge_{i\in[1,m]} ((\neg b^i \wedge b^{i-1}) \to (R(L^{i-1}) \wedge S'(N)))$.

f) $q^5 = b^m \to (\bigvee_{i\in[0,n)} ((\bigwedge_{j\in[i,n-1)} \neg e^j) \wedge (\bigwedge_{j\in[0,i)} e^j) \wedge Q'(X^0, M' \cap (L^i \cdots \cup L^{i+n-1} \cup N)))$.

g) $q^6 = \bigwedge_{i\in[1,m]} (e^i \to e^{i-1})$.

In Def. 8.20, one flipping bit e is introduced for two-phase abstraction. When generalizing this method, $n-1$ flipping bits are needed to freeze the transitions of latches in N. They operate in a similar fashion as a one-hot counter, and when saturated, it resets to zero (all bits set to $\perp$). An alternative to this is to use a binary counter instead. We generalize Theorem 8.21 to obtain the following corollary.

Corollary 8.23. *Given a circuit $C = (I, L, R, F, P)$ with a phase number $n \in \mathbb{N}^+$, its unfolded cicuit $C' = (I', L', R', F', P')$ with a witness circuit of C'. Let $W = (I, M, S, G, Q)$ be the circuit constructed as in Def. 8.22.*
Then W is a witness circuit of C.

In summary, by taking a compositional approach, we can build a witness circuit step-wise for the entire model checking pipeline including both phase abstraction and base model checking. For future work, it is necessary to implement the theoretical framework into an experimental toolkit.

Chapter 9

Conclusion and Future Work

In this thesis we presented a novel and refined certificate format for hardware model checking and demonstrated that it is compatible with a wide variety of state-of-the-art model checking techniques. We presented different certification techniques that are the topics of four peer-reviewed papers (Chapter 4-7), as well as an unsubmitted paper (Chapter 8). The contributions and results of each publication are thoroughly discussed and reflected upon. Of all four publications, the author of this thesis is the main author.

In Chapter 4, with the goal of establishing a standardized certificate format for hardware model checking, we presented a bit-level certificate format that involves five simple SAT checks and a one-alternation QBF check. This certificate format is called a witness circuit, which combinationally simulates the circuit under verification and contains an inductive invariant. Based on this defined format, whenever the model checker produces such a certificate and it is then validated by an independent checker, it is guaranteed that the model checking result is correct, as formally proved in Chapter 4. In addition to this, we show how to produce such a certificate for the popular model checking technique k-induction at bit-level. We also present a thorough experimental evaluation to demonstrate the practicality of the method. In Chapter 3.1, we engaged in a thorough discussion regarding the challenges encountered when integrating k-induction with simple paths constraints into our certification framework. It remains an interesting open question for future research.

In Chapter 5, we refined the certificate format defined in Chapter 4 by eliminating the quantifier. The resulting certificate requires six simple SAT checks and one polynomial-time check for the acyclicity of the reset function definitions. In addition to that, we simplified the witness circuit construction for k-induction and lifted the framework to word-level. Experimental results are presented for both bit-level and word-level, showing the effectiveness of the framework. In the word-level case, the certificate simply requires six SMT checks using fixed-size bit-vectors logic instead of SAT checks. Along the line of word-level certification, there are lots of potential research problems to explore. Word-level model checkers employ a different range of techniques, although there may be intersections with bit-level model checking approaches such as k-induction. One potential direction for future work is the certification of data abstraction.

In Chapter 6, we adopted the same certificate format from Chapter 5, and showed how to build a certificate for the preprocessing technique temporal decomposition when used in conjunction with base model checking techniques. To do so, we revisited temporal decomposition by defining its precise formalism, as an improvement over existing

work. In particular, we defined the notion of cube simulation that subsumes ternary simulation. Another approach sometimes used for finding transient signals is to use k-induction. Both approaches are incomplete, and believed to be orthogonal to each other [36]. For future work, it is worth exploring the alternative approach too. Based on the defined semantics for temporal decomposition, we presented a compositional method of constructing a witness circuit resulting in elegant proofs, together with experimental results. In Chapter 3.2 and 3.3, we presented additional comprehensive experimental results on a more recent set of benchmarks from HWMCC'19 to provide further insights into the effectiveness of our approaches.

In Chapter 8, our focus was on improving the standardized certification framework to incorporate additional verification techniques. We introduced a proof-of-concept construction to produce certificates for phase abstraction and provided theoretical results. In a more specific context, we further generalized the certificate format defined in Chapter 5 in order to make it compatible with more preprocessing techniques by introducing the notion of restricted simulation relation. Consequently, it becomes imperative to revisit the construction and formal proofs pertaining to certifying temporal decomposition in Chapter 6, which can be seen as the immediate next step. From the practical perspective, the subsequent phase in this line of work is to implement the proposed certification method and demonstrate its practical utility.

Chapter 6 showcased our experimental evaluation, where we employed BDD-based model checking and k-induction as the underlying model checking algorithms using temporal decomposition for preprocessing. Moving forward, our goal is to expand the scope of the certification tool suite to include the IC3/PDR-based model checking approach. This entails implementing an IC3/PDR-based model checker that accepts reset functions and produces certificates according to the proposed format.

Furthermore, Chapter 7 presents an exploratory certification framework for model checking multi-agent systems employing QBF solvers with the goal of generating more QBF benchmarks. As part of future work, we envision applying the framework proposed in Chapter 4-8 to certify QBF or higher order logic based model checkers, thereby expanding the scope of our certification approach.

In summary, several exciting avenues for future research present themselves in light of this study. Our proposed approach paves the way for a simple and generic certificate format for a complex combination of diverse model checking techniques. While we have successfully addressed a number of these techniques, there is an abundance of them for further investigation using our format, such as localization [93], retiming [90] and SAT sweeping [65, 91, 109, 110, 144]. Another direction of interest is to certify constraints for assertion-based verification [141]. Additionally, it is worth noting that so far the focus of this thesis has primarily been on safety properties. It is also an interesting avenue for future research to extend the certification framework to liveness properties.

After establishing the generic certificate format, an intriguing prospect is to incorporate certification requirements for all model checkers participating in the hardware model checking competitions. This would significantly enhance the quality of the model checkers, drawing parallels with the influential SAT competitions that have propelled the advancement of SAT solvers enabling more aggressive optimizations over the years.

While the focus thus far has been on the functionality and versatility of the certificates, it is imperative for future research endeavors to delve into the optimization of these certificates for smaller certificate size and faster certification. For example, through experimental evaluation, for k-induction certificates the consecution check seems to be the performance bottleneck, which could potentially be improved through trimming. By seeking ways to optimize the certificate size, we aim to expedite the certificate checking process, thereby facilitating faster and more efficient verification. Beyond the scope of hardware model checking competitions and the research community, we envision the widespread adoption of certification methods in the industry, increasing trust and confidence in the results of verification process.

Lastly, another line of research lies at developing a certificate checker that is formally verified in a theorem proving environment such as using Isabelle/HOL [117]. This opens up more research questions, especially in the context of word-level certification, where SMT proofs would need to be incorporated [87, 94, 118].

Bibliography

[1] Martín Abadi and Leslie Lamport. The existence of refinement mappings. In *LICS*, pages 165–175. IEEE Computer Society, 1988.

[2] Alex Abuin, Alexander Bolotov, Unai Díaz-de-Cerio, Montserrat Hermo, and Paqui Lucio. Towards certified model checking for PLTL using one-pass tableaux. In *TIME*, volume 147 of *LIPIcs*, pages 12:1–12:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[3] Hasan Amjad. Programming a symbolic model checker in a fully expansive theorem prover. In *Theorem Proving in Higher Order Logics: 16th International Conference, TPHOLs 2003, Rome, Italy, September 8-12, 2003. Proceedings 16*, pages 171–187. Springer, 2003.

[4] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[5] Tomás Balyo, Marijn J. H. Heule, and Matti Järvisalo. SAT competition 2016: Recent developments. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 5061–5063. AAAI Press, 2017.

[6] Clark Barrett and Cesare Tinelli. *Satisfiability modulo theories*. Springer, 2018.

[7] Jason Baumgartner, Tamir Heyman, Vigyan Singhal, and Adnan Aziz. Model checking the ibm gigahertz processor: An abstraction algorithm for high-performance netlists. In *Computer Aided Verification: 11th International Conference, CAV'99 Trento, Italy, July 6–10, 1999 Proceedings 11*, pages 72–83. Springer, 1999.

[8] Jason Baumgartner, Tamir Heyman, Vigyan Singhal, and Adnan Aziz. An abstraction algorithm for the verification of level-sensitive latch-based netlists. *Formal Methods in System Design*, 23:39–65, 2003.

[9] Sam Bayless, Celina G Val, Thomas Ball, Holger H Hoos, and Alan J Hu. Efficient modular sat solving for ic3. In *2013 Formal Methods in Computer-Aided Design*, pages 149–156. IEEE, 2013.

[10] Francesco Belardinelli, Alessio Lomuscio, Vadim Malvone, and Emily Yu. Approximating perfect recall when model checking strategic abilities: theory and applications. *Journal of Artificial Intelligence Research*, 73:897–932, 2022.

[11] Francesco Belardinelli, Alessio Lomuscio, and Emily Yu. Model checking temporal epistemic logic under bounded recall. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7071–7078, 2020.

[12] Dirk Beyer, Matthias Dangl, Daniel Dietsch, and Matthias Heizmann. Correctness witnesses: exchanging verification results between verifiers. In *SIGSOFT FSE*, pages 326–337. ACM, 2016.

[13] Dirk Beyer, Matthias Dangl, Daniel Dietsch, Matthias Heizmann, Thomas Lemberger, and Michael Tautschnig. Verification witnesses. *ACM Trans. Softw. Eng. Methodol.*, 31(4):57:1–57:69, 2022.

[14] Armin Biere. Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. Technical report, FMV Reports Series, Inst. FMV, JKU Linz, Austria, 2010.

[15] Armin Biere and Robert Brummayer. Consistency checking of all different constraints over bit-vectors within a SAT solver. In *FMCAD*, pages 1–4. IEEE, 2008.

[16] Armin Biere and Robert Brummayer. Consistency checking of all different constraints over bit-vectors within a SAT solver. In *FMCAD*, pages 1–4. IEEE, 2008.

[17] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *Tools and Algorithms for the Construction and Analysis of Systems: 5th International Conference, TACAS'99 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS'99 Amsterdam, The Netherlands, March 22–28, 1999 Proceedings 5*, pages 193–207. Springer, 1999.

[18] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Masahiro Fujita, and Yunshan Zhu. Symbolic model checking using sat procedures instead of bdds. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pages 317–320, 1999.

[19] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *Proc. of TACAS'99*, volume 1579 of *LNCS*, pages 193–207. Springer.

[20] Armin Biere and Koen Claessen. Hardware model checking competition 2010. 2010.
http://fmv.jku.at/hwmcc10/.

[21] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo,

and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

[22] Armin Biere, Keijo Heljanko, and Siert Wieringa. AIGER 1.9 and beyond. Technical report, FMV Reports Series, Inst. FMV, JKU Linz, Austria, 2011.

[23] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021.

[24] Armin Biere, Tom van Dijk, and Keijo Heljanko. Hardware model checking competition 2017. In Daryl Stewart and Georg Weissenbacher, editors, *Formal Methods in Computer-Aided Design, FMCAD.*, page 9. IEEE, 2017.

[25] Armin Biere, Emily Yu, and Nils Froleyks. Stratified certification for k-induction. In *FMCAD*, volume 3, page 59. TU Wien Academic Press, 2022.

[26] Per Bjesse and James H. Kukula. Automatic generalized phase abstraction for formal verification. In *ICCAD*, pages 1076–1082. IEEE Computer Society, 2005.

[27] Nikolaj Bjørner, Arie Gurfinkel, Kenneth L. McMillan, and Andrey Rybalchenko. Horn clause solvers for program verification. In *Fields of Logic and Computation II*, volume 9300 of *Lecture Notes in Computer Science*, pages 24–51. Springer, 2015.

[28] Aaron Bradley, Fabio Somenzi, Michael Dooley, Zyad Hassan, and Yan Zhang. Incremental inductive model checker.

[29] Aaron R. Bradley. SAT-based model checking without unrolling. In *VMCAI*, volume 6538 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2011.

[30] Aaron R Bradley. Sat-based model checking without unrolling. In *Verification, Model Checking, and Abstract Interpretation: 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings 12*, pages 70–87. Springer, 2011.

[31] Aaron R Bradley and Zohar Manna. *The calculus of computation: decision procedures with applications to verification*. Springer Science & Business Media, 2007.

[32] Robert K. Brayton and Alan Mishchenko. ABC: an academic industrial-strength verification tool. In *CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 24–40. Springer, 2010.

[33] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10ˆ20 states and beyond. In *LICS*, pages 428–439. IEEE Computer Society, 1990.

[34] Gianpiero Cabodi, Sergio Nocco, and Stefano Quer. The pdtrav tool.

[35] Gianpiero Cabodi, Sergio Nocco, and Stefano Quer. Thread-based multi-engine model checking for multicore platforms. *ACM Trans. Design Autom. Electr. Syst.*, 18(3):36:1–36:28, 2013.

[36] Michael L. Case, Hari Mony, Jason Baumgartner, and Robert Kanzelman. Enhanced verification by temporal decomposition. In *FMCAD*, pages 17–24. IEEE, 2009.

[37] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuxmv symbolic model checker. In *CAV*, volume 8559 of *Lecture Notes in Computer Science*, pages 334–342. Springer, 2014.

[38] Certifaiger. Certifaiger. 2021. http://fmv.jku.at/certifaiger.

[39] Adrien Champion, Alain Mebsout, Christoph Sticksel, and Cesare Tinelli. The Kind 2 model checker. In *CAV (2)*, volume 9780 of *Lecture Notes in Computer Science*, pages 510–517. Springer, 2016.

[40] CHMC. CHMC. 2023. http://fmv.jku.at/chmc.

[41] Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal methods in system design*, 19:7–34, 2001.

[42] Edmund M. Clarke, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model Checking*. MIT press, 2018.

[43] Edmund M. Clarke, Orna Grumberg, Daniel Kroening, Doron A. Peled, and Helmut Veith. *Model checking, 2nd Edition*. MIT Press, 2018.

[44] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018.

[45] Edmund M. Clarke, David E. Long, and Kenneth L. McMillan. Compositional model checking. In *LICS*, pages 353–362. IEEE Computer Society, 1989.

[46] Sylvain Conchon, Alain Mebsout, and Fatiha Zaïdi. Certificates for parameterized model checking. In Nikolaj Bjørner and Frank S. de Boer, editors, *FM 2015: Formal Methods - 20th International Symposium, Oslo, Norway, June 24-26, 2015, Proceedings*, volume 9109 of *Lecture Notes in Computer Science*, pages 126–142. Springer, 2015.

[47] Luís Cruz-Filipe, Marijn JH Heule, Warren A Hunt, Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified rat verification. In *Automated Deduction–CADE 26: 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6–11, 2017, Proceedings*, pages 220–236. Springer, 2017.

[48] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[49] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

[50] Leonardo Mendonça de Moura, Sam Owre, Harald Rueß, John M. Rushby, Natarajan Shankar, Maria Sorea, and Ashish Tiwari. SAL 2. In *CAV*, volume 3114 of *Lecture Notes in Computer Science*, pages 496–500. Springer, 2004.

[51] Anatoli Degtyarev and Andrei Voronkov. Equality reasoning in sequent-based calculi. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 611–706. Elsevier and MIT Press, 2001.

[52] Nachum Dershowitz, Ziyad Hanna, and Jacob Katz. Bounded model checking with QBF. In *Proc. of SAT'05*, volume 3569 of *LNCS*, pages 408–414. Springer, 2005.

[53] Alastair F. Donaldson, Leopold Haller, Daniel Kroening, and Philipp Rümmer. Software verification using k-induction. In *SAS*, volume 6887 of *Lecture Notes in Computer Science*, pages 351–368. Springer, 2011.

[54] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. Multi-agent systems: A survey. *IEEE Access*, 6:28573–28593, 2018.

[55] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *SAT*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.

[56] Niklas Eén, Alan Mishchenko, and Robert Brayton. Efficient implementation of property directed reachability. In *2011 Formal Methods in Computer-Aided Design (FMCAD)*, pages 125–134. IEEE, 2011.

[57] Niklas Eén and Niklas Sörensson. Temporal induction by incremental sat solving. *Electronic Notes in Theoretical Computer Science*, 89(4):543–560, 2003.

[58] Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *Electron. Notes Theor. Comput. Sci.*, 89(4):543–560, 2003.

[59] Salomé Eriksson, Gabriele Röger, and Malte Helmert. Unsolvability certificates for classical planning. In Laura Barbulescu, Jeremy Frank, Mausam, and Stephen F. Smith, editors, *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017*, pages 88–97. AAAI Press, 2017.

[60] Javier Esparza, Peter Lammich, René Neumann, Tobias Nipkow, Alexander Schimpf, and Jan-Georg Smaus. A fully verified executable ltl model checker. In

Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25, pages 463–478. Springer, 2013.

[61] Javier Esparza, Peter Lammich, René Neumann, Tobias Nipkow, Alexander Schimpf, and Jan-Georg Smaus. A fully verified executable LTL model checker. In *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 463–478. Springer, 2013.

[62] Javier Esparza, Peter Lammich, René Neumann, Tobias Nipkow, Alexander Schimpf, and Jan-Georg Smaus. A fully verified executable LTL model checker. *Arch. Formal Proofs*, 2014, 2014.

[63] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, Cambridge, MA, USA, 2003.

[64] Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda. Sat competition 2020. *Artificial Intelligence*, 301:103572, 2021.

[65] Masahiro Fujita. Toward unification of synthesis and verification in topologically constrained logic design. *Proceedings of the IEEE*, 103(11):2052–2060, 2015.

[66] Andrew Gacek, John Backes, Mike Whalen, Lucas G. Wagner, and Elaheh Ghassabani. The JKind model checker. In *CAV (2)*, volume 10982 of *Lecture Notes in Computer Science*, pages 20–27. Springer, 2018.

[67] QBF Gallery. Qcir-g14: A non-prenex non-cnf format for quantified boolean formulas, 2014.

[68] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[69] Ning Ge, Eric Jenn, Nicolas Breton, and Yoann Fonteneau. Integrated formal verification of safety-critical software. *Int. J. Softw. Tools Technol. Transf.*, 20(4):423–440, 2018.

[70] Aman Goel and Karem A. Sakallah. AVR: abstractly verifying reachability. In Armin Biere and David Parker, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part I*, volume 12078 of *Lecture Notes in Computer Science*, pages 413–422. Springer, 2020.

[71] Evguenii I. Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *DATE*, pages 10886–10891. IEEE Computer Society, 2003.

[72] Alberto Griggio, Marco Roveri, and Stefano Tonetta. Certifying proofs for LTL model checking. In *FMCAD*, pages 1–9. IEEE, 2018.

[73] Alberto Griggio, Marco Roveri, and Stefano Tonetta. Certifying proofs for SAT-based model checking. *Formal Methods Syst. Des.*, 57(2):178–210, 2021.

[74] Daniel Große, Hoang Minh Le, and Rolf Drechsler. Induction-based formal verification of SystemC TLM designs. In *MTV*, pages 101–106. IEEE Computer Society, 2009.

[75] Arie Gurfinkel and Alexander Ivrii. K-induction without unrolling. In *2017 Formal Methods in Computer Aided Design (FMCAD)*, pages 148–155. IEEE, 2017.

[76] Arie Gurfinkel and Alexander Ivrii. K-induction without unrolling. In *FMCAD*, pages 148–155. IEEE, 2017.

[77] G Hasteer, A Mathur, and P Bannerjee. A framework for equivalence checking of multi-phase fsms. In *Proc. High Level Design Validation and Test Symp*, 1997.

[78] Marijn J. H. Heule. Proofs of unsatisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 635–668. IOS Press, 2021.

[79] Marijn J. H. Heule, Matti Järvisalo, and Martin Suda. SAT competition 2018. *J. Satisf. Boolean Model. Comput.*, 11(1):133–154, 2019.

[80] Marijn JH Heule. Proofs of unsatisfiability. In *Handbook of Satisfiability*, pages 635–668. IOS Press, 2021.

[81] Marijn J.H. Heule and Armin Biere. Proofs for satisfiability problems. In Bruno Woltzenlogel Paleo and David Delahaye, editors, *All about Proofs, Proofs for all*, chapter 1. College Publications, 2015.

[82] Marijn JH Heule, Warren A Hunt, and Nathan Wetzler. Trimming while checking clausal proofs. In *2013 Formal Methods in Computer-Aided Design*, pages 181–188. IEEE, 2013.

[83] Matti Järvisalo, Marijn Heule, and Armin Biere. Inprocessing rules. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, 2012.

[84] Jie-Hong Roland Jiang, Alan Mishchenko, and Robert K. Brayton. On breakable cyclic definitions. In *ICCAD*, pages 411–418. IEEE Computer Society / ACM, 2004.

[85] Charles Jordan, Will Klieber, and Martina Seidl. Non-cnf QBF solving with QCIR. In *AAAI Workshop: Beyond NP*, volume WS-16-05 of *AAAI Workshops*. AAAI Press, 2016.

[86] Dejan Jovanovic and Bruno Dutertre. Property-directed k-induction. In *FMCAD*, pages 85–92. IEEE, 2016.

[87] Guy Katz, Clark Barrett, Cesare Tinelli, Andrew Reynolds, and Liana Hadarean. Lazy proofs for dpll (t)-based smt solvers. In *2016 Formal Methods in Computer-Aided Design (FMCAD)*, pages 93–100. IEEE, 2016.

[88] Konstantin Korovin, André Duarte, and Edvard K Holden. iprover v3. 5 (smt-comp).

[89] Hari Govind Vediramana Krishnan, Yakir Vizel, Vijay Ganesh, and Arie Gurfinkel. Interpolating strong induction. In *CAV (2)*, volume 11562 of *Lecture Notes in Computer Science*, pages 367–385. Springer, 2019.

[90] Andreas Kuehlmann and Jason Baumgartner. Transformation-based verification using generalized retiming. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 104–117. Springer, 2001.

[91] Andreas Kuehlmann, Viresh Paruthi, Florian Krohm, and Malay K Ganai. Robust boolean reasoning for equivalence checking and functional property verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(12):1377–1394, 2002.

[92] Tuomas Kuismin and Keijo Heljanko. Increasing confidence in liveness model checking results with proofs. In *Haifa Verification Conference*, volume 8244 of *Lecture Notes in Computer Science*, pages 32–43. Springer, 2013.

[93] Robert P Kurshan. Computer-aided verification of coordinating processes. In *Computer-Aided Verification of Coordinating Processes*. Princeton university press, 2014.

[94] Hanna Lachnitt, Mathias Fleury, Leni Aniva, Andrew Reynolds, Haniel Barbosa, Andres Nötzli, Clark Barrett, and Cesare Tinelli. Automatic verification of SMT rewrites in Isabelle/HOL. 2023. Submitted.

[95] Jussi Lahtinen, J. Valkonen, Kim Björkman, J. Frits, Ilkka Niemelä, and Keijo Heljanko. Model checking of safety-critical software in the nuclear engineering domain. *Reliab. Eng. Syst. Saf.*, 105:104–113, 2012.

[96] Chien-Yu Lai, Cheng-Yin Wu, and Chung-Yang Ric Huang. Adaptive interpolation-based model checking. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 744–749. IEEE, 2014.

[97] Alessio Lomuscio, T Lasica, and Wojciech Penczek. Bounded model checking for interpreted systems: Preliminary experimental results. In *Formal Approaches to Agent-Based Systems: Second International Workshop, FAABS 2002, Greenbelt, MD, USA, October 29-31, 2002. Revised Papers 2*, pages 115–125. Springer, 2003.

[98] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *CAV*, volume 5643 of *LNCS*, pages 682–688. Springer, 2009.

[99] Alessio Lomuscio and Franco Raimondi. The complexity of model checking concurrent programs against CTLK specifications. In *DALT*, volume 4327 of *LNCS*, pages 29–42. Springer, 2006.

[100] Sharad Malik. Analysis of cyclic combinational circuits. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 13(7):950–956, 1994.

[101] Makai Mann, Ahmed Irfan, Florian Lonsing, Yahan Yang, Hongce Zhang, Kristopher Brown, Aarti Gupta, and Clark Barrett. Pono: a flexible and extensible smt-based model checker. In *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part II 33*, pages 461–474. Springer, 2021.

[102] Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems - safety*. Springer, 1995.

[103] Joao Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning sat solvers. In *Handbook of satisfiability*, pages 133–182. IOS press, 2021.

[104] MCMAS-QBF. 2019.

[105] Kenneth L. McMillan. Interpolation and sat-based model checking. In *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2003.

[106] Kenneth L. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1):101–121, 2005.

[107] Alain Mebsout and Cesare Tinelli. Proof certificates for smt-based model checkers for infinite-state systems. In Ruzica Piskac and Muralidhar Talupur, editors, *2016 Formal Methods in Computer-Aided Design, FMCAD 2016, Mountain View, CA, USA, October 3-6, 2016*, pages 117–124. IEEE, 2016.

[108] Alan Mishchenko and Robert K. Brayton. Recording synthesis history for sequential verification. In *FMCAD*, pages 1–8. IEEE, 2008.

[109] Alan Mishchenko, Satrajit Chatterjee, Robert Brayton, and Niklas Een. Improvements to combinational equivalence checking. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 836–843, 2006.

[110] Alan Mishchenko, Satrajit Chatterjee, Roland Jiang, and Robert K Brayton. Fraigs: A unifying representation for logic synthesis and verification. Technical report, ERL Technical Report, 2005.

[111] Kedar S. Namjoshi. Certifying model checkers. In *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 2–13. Springer, 2001.

[112] Aina Niemetz, Mathias Preiner, and Armin Biere. Precise and complete propagation based local search for satisfiability modulo theories. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*, volume 9779 of *Lecture Notes in Computer Science*, pages 199–217. Springer, 2016.

[113] Aina Niemetz, Mathias Preiner, and Armin Biere. Propagation based local search for bit-precise reasoning. *Formal Methods Syst. Des.*, 51(3):608–636, 2017.

[114] Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark W. Barrett, and Cesare Tinelli. Solving quantified bit-vectors using invertibility conditions. In *CAV (2)*, volume 10982 of *Lecture Notes in Computer Science*, pages 236–255. Springer, 2018.

[115] Aina Niemetz, Mathias Preiner, Andrew Reynolds, Yoni Zohar, Clark W. Barrett, and Cesare Tinelli. Towards satisfiability modulo parametric bit-vectors. *J. Autom. Reason.*, 65(7):1001–1025, 2021.

[116] Aina Niemetz, Mathias Preiner, Clifford Wolf, and Armin Biere. Btor2 , btormc and boolector 3.0. In *CAV (1)*, volume 10981 of *Lecture Notes in Computer Science*, pages 587–595. Springer, 2018.

[117] Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer, 2002.

[118] Alex Ozdemir, Aina Niemetz, Mathias Preiner, Yoni Zohar, and Clark Barrett. Drat-based bit-vector proofs in cvc4. In *Theory and Applications of Satisfiability Testing–SAT 2019: 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9–12, 2019, Proceedings 22*, pages 298–305. Springer, 2019.

[119] Wojciech Penczek and Alessio Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 209–216, 2003.

[120] Mathias Preiner, Armin Biere, and Nils Froleyks. Hardware model checking competition 2020. 2020.

[121] Mathias Preiner, Armin Biere, and Nils Froleyks. Hardware model checking competition 2020. 2020.
http://fmv.jku.at/hwmcc20/.

[122] Marc D Riedel. *Cyclic combinational circuits*. California Institute of Technology, 2004.

[123] Carl-Johan H. Seger and Randal E. Bryant. Formal verification by symbolic evaluation of partially-ordered trajectories. *Formal Methods Syst. Des.*, 6(2):147–189, 1995.

[124] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking safety properties using induction and a SAT-solver. In *FMCAD*, volume 1954 of *Lecture Notes in Computer Science*, pages 108–125. Springer, 2000.

[125] Christoph Sprenger. A verified model checker for the modal μ-calculus in coq. In *Tools and Algorithms for the Construction and Analysis of Systems: 4th International Conference, TACAS'98 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS'98 Lisbon, Portugal, March 28–April 4, 1998 Proceedings 4*, pages 167–183. Springer, 1998.

[126] Martin Suda and Christoph Weidenbach. A pltl-prover based on labelled superposition with partial model guidance. In *Automated Reasoning: 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings 6*, pages 537–543. Springer, 2012.

[127] Leander Tentrup. Non-prenex QBF solving using abstraction. In *Proc. of SAT'16*, volume 9710 of *LNCS*, pages 393–401. Springer, 2016.

[128] Grigori S Tseitin. On the complexity of derivation in propositional calculus. *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*, pages 466–483, 1983.

[129] Wiebe van der Hoek and Michael J. Wooldridge. Tractable multiagent planning for epistemic goals. In *AAMAS*, pages 1167–1174. ACM, 2002.

[130] Allen Van Gelder. Verifying rup proofs of propositional unsatisfiability. In *ISAIM*, 2008.

[131] Allen Van Gelder. Producing and verifying extremely large propositional refutations: Have your cake and eat it too. *Annals of Mathematics and Artificial Intelligence*, 65:329–372, 2012.

[132] Yakir Vizel and Arie Gurfinkel. Interpolating property directed reachability. In *Computer Aided Verification: 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings 26*, pages 260–276. Springer, 2014.

[133] Lucas G. Wagner, Alain Mebsout, Cesare Tinelli, Darren D. Cofer, and Konrad Slind. Qualification of a model checker for avionics software verification. In Clark W. Barrett, Misty Davies, and Temesghen Kahsai, editors, *NASA Formal Methods - 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings*, volume 10227 of *Lecture Notes in Computer Science*, pages 404–419, 2017.

[134] Chao Wang, Bing Li, HoonSang Jin, Gary D Hachtel, and Fabio Somenzi. Improving ariadne's bundle by following multiple threads in abstraction refinement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(11):2297–2316, 2006.

[135] Simon Wimmer and Joshua von Mutius. Verified certification of reachability checking for timed automata. In *TACAS (1)*, volume 12078 of *Lecture Notes in Computer Science*, pages 425–443. Springer, 2020.

[136] Cheng-Yin Wu, Chi-An Wu, Chien-Yu Lai, and Chung-Yang Huang. A counterexample-guided interpolant generation algorithm for sat-based model checking. In *Proceedings of the 50th Annual Design Automation Conference*, pages 1–6, 2013.

[137] Emily Yu, Armin Biere, and Keijo Heljanko. Progress in certifying hardware model checking results. In *CAV (2)*, volume 12760 of *Lecture Notes in Computer Science*, pages 363–386. Springer, 2021.

[138] Emily Yu, Armin Biere, and Keijo Heljanko. Progress in certifying hardware model checking results. In *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part II 33*, pages 363–386. Springer, 2021.

[139] Emily Yu, Martina Seidl, and Armin Biere. A framework for model checking against CTLK using quantified boolean formulas. In *FTSCS*, volume 1165 of *Communications in Computer and Information Science*, pages 127–132. Springer, 2019.

[140] Zhengqi Yu, Armin Biere, and Keijo Heljanko. Certifying hardware model checking results. In *ICFEM*, volume 11852 of *Lecture Notes in Computer Science*, pages 498–502. Springer, 2019.

[141] Jun Yuan, Carl Pixley, and Adnan Aziz. *Constraint-based verification*. Springer Science & Business Media, 2006.

[142] Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *DATE*, pages 10880–10885. IEEE Computer Society, 2003.

[143] Conghua Zhou, Zhenyu Chen, and Zhihong Tao. QBF-based symbolic model checking for knowledge and time. In *TAMC*, volume 4484 of *LNCS*, pages 386–397. Springer, 2007.

[144] Qi Zhu, Nathan Kitchen, Andreas Kuehlmann, and Alberto Sangiovanni-Vincentelli. Sat sweeping with local observability don't-cares. In *Proceedings of the 43rd Annual Design Automation Conference*, pages 229–234, 2006.