Manipulating Proof Certificate Invariants in the Presence of Model Transformations

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 25 March 2026, accepted 22 April 2026, date of publication 27 April 2026, date of current version 4 May 2026.

Digital Object Identifier 10.1109/ACCESS.2026.3687715

RESEARCH ARTICLE

Manipulating Proof Certificate Invariants in the Presence of Model Transformations

GIANPIERO CABODI[1], PAOLO ENRICO CAMURATI[1], MARCO PALENA[2], AND PAOLO PASINI[3]

[1]Department of Control and Computer Engineering (DAUIN), Politecnico di Torino, 10129 Turin, Italy
[2]National Inter-University Consortium for Telecommunications (CNIT), 10129 Turin, Italy
[3]Department of Electronics and Telecommunications (DET), Politecnico di Torino, 10129 Turin, Italy

Corresponding author: Paolo Pasini (paolo.pasini@polito.it)

This work was supported in part by European Union–Next Generation EU through Italian National Recovery and Resilience Plan (NRRP), Mission 4, Component 2, Investment 1.3, partnership on "Telecommunications of the Future" (PE00000001–Program "RESTART") under Grant CUP E13C22001870001; and in part by Semiconductor Research Corporation (SRC) under Contract 2024-CT-3280.

ABSTRACT Trust in the outcomes of state-of-the-art hardware model checkers is more and more related to the availability of external certification tools, especially when the result is a "pass" verdict that cannot be simulated. This article addresses one of the major weaknesses of proof certifiers: the difficulty of exploiting a proof certificate obtained after the model was pre-processed with optimizations and transformations. Most proof engines provide a certificate in terms of an inductive invariant that could be easily certified by an external checker on the transformed model. On the other hand, proving the correctness of each model transformation is basically infeasible using an external independent tool. Recent works have followed the idea of post-processing both the model and the inductive invariant, by generating a witness model and a transformed invariant: the certifier then checks that the invariant holds on the witness model and that the witness model is consistent with the original one. We follow here the alternative path of fully lifting the invariant, back to the original model, with a two-fold target: 1) simplifying the task of the model checker, which is only required to output the final invariant and a log of the transformations done; 2) possibly reusing the invariant for other tasks on the model, such as logic synthesis optimizations and proofs of other properties. Specifically, we target the following transformations: temporal decomposition, phase abstraction and equivalence and constant propagation. As this process requires existential quantification operations, we show how to exploit Craig's interpolation to generate new inductive invariants. We also address scalability issues by applying a state-of-the-art SAT-based simplification procedure capable of reducing the size of the generated invariants when converted to NNF form. We experimentally demonstrate the effectiveness of this approach on a set of invariants derived from publicly available hardware model-checking benchmarks.

INDEX TERMS Boolean satisfiability (SAT), hardware model checking, formal verification, invariants, Craig's interpolants.

I. INTRODUCTION

Hardware verification and model checking remain challenging problems even today, due to the ever-increasing complexity of designs requiring verification. State-of-the-art hardware model checkers are based on rich and complex portfolios of multiple model transformations and proof engines, often implemented by concurrent independent applications. In such a scenario, certifying the outcome of a model checker has become essential. Although counterexamples for failed properties are not problematic because they can be simulated, certifying proved properties remains a challenge, despite recent advances in proof certificates. This paper addresses one of the major weaknesses of proof certifiers: the difficulty of leveraging a proof certificate when the underlying model differs from the original, due to the intermediate optimizations and transformations applied before activating the proof engines. Since most proof engines provide certificates in the

The associate editor coordinating the review of this manuscript and approving it for publication was Ludovico Minati.

form of inductive invariants that can be easily verified by an external checker, establishing the correctness of each model transformation is essentially infeasible for an independent external tool. Recent works [1], [2], [3], [4] have followed the idea of post-processing both the model and the inductive invariant, by generating a witness model and a transformed invariant: then the certifier checks that the invariant holds on the witness model and that the witness model is consistent with the original one. In this work, we take an alternative approach by fully lifting the invariant back to the original model, aiming at two main goals: (a) making the model checker's task easier, since it only needs to produce the final invariant along with a log of the transformations applied; and (b) enabling potential reuse of the invariant for other purposes, such as logic synthesis optimizations and verifying additional properties. Specifically, we target the following transformations: temporal decomposition, phase abstraction and equivalence, and constant propagation. In this work, we focus on safety properties, which are the most common use-cases in hardware model checking scenarios. Using the proposed approach as a base, our findings could also be extended to include additional kinds of transformation and to support a wider range of properties. For example, one could consider LTL model checking algorithms such as FAIR [5] and liveness-to-safety [6].

Because the proposed approach involves existential quantification, we demonstrate how Craig's interpolation can be used to generate new inductive invariants. To address scalability concerns, we further introduce a fine-tuning step for the lifted invariant, employing a state-of-the-art SAT-based simplification technique that substantially reduces the size of the generated invariants after conversion into negation normal form (NNF). Our experimental results validate the effectiveness of this method on a range of invariants obtained from publicly available hardware model-checking benchmarks.

A. RELATED WORKS

Given the context of application, our approach is closely related to recent works on certification based on witness circuits. In [1], the authors introduce a formal framework designed to certify the results of $k-$induction-based model checking. At the core of such a framework there is the concept of a $k-$witness circuit, acting as a simulation of the original circuit and including an inductive invariant that serves as a proof certificate. Further expansions and improvements are presented in [2], [3], and [4], and the related tool, *Certifaiger* [7], is adopted as a "de facto" standard for the certification of model checker proofs at the Hardware Model Checking competitions [8].

1) CERTIFAIGER

More in detail, Certifaiger addresses model transformations with the two following rules:

- When the final proof is obtained on a transformed model, the inductive invariant (an *inductive strengthening of the property*) has to be brought back to the original model (before transformations). The model checker is in charge of producing the so called *witness circuit*, and a related inductive invariant. The witness circuit simulates the original model, so if the property holds on the witness circuit, it also holds on the original model.
- The certification tool (e.g., Certifiger) is in charge of two tasks: (a) checking that the witness circuit simulates the original model; this is done by a pure syntactic (topological) check, based on the assumption that the witness circuit is basically obtained by adding extra circuitry (under given constraints) to the original model; (b) checking that the property holds on the witness circuit, done by SAT based checks on the inductive invariant.

The difficulty of the approach stands in the fact that the backward transformation can be a complex/tricky additional burden for the model checker. In summary, the original circuit must be augmented —focusing, for simplicity, on two transformations, namely temporal decomposition and phase abstraction— with:

- additional state variables to identify time frames in the initial transient (for temporal decomposition) or in the omitted time steps (for phase abstraction);
- additional circuitry to extend the inductive invariant with new terms that capture the states reached during those time steps.

2) OTHER APPROACHES

The idea of reducing the burden on model checker developers to generate certifying proofs, by providing a comprehensive certification tool that can deal with model transformations, is followed in [9], where a $k-$liveness algorithm is exploited to extend proof generation capabilities that cover full linear-time temporal logic (LTL) properties, with little overhead for the model checker. The work is extended in [10] exploring the usage of a theorem prover as a certification tool.

3) EQUIVALENCE CHECKING

Although model transformations are at the base of equivalence checking tools, both combinational and sequential, equivalence checking is currently out of the scope of model checking certification, which mainly addresses property checking. In fact, model checking problems and benchmark suites include equivalence checking problems, as equivalences can be expressed in terms of properties. They generally trigger specific engines/tunings in the portfolio of a model checker, and transformations such as signal correspondence and retiming, speculative reduction. Some of these transformations will be addressed in future work. But real life equivalence checking problems and commercial equivalence checkers need a tight interaction with (logic and/or high level) synthesis tools, which makes certification

of equivalence checking oriented transformations a very specific and peculiar task.

4) THIS WORK

In this work, we partially deviate from the idea of a comprehensive and trusted certifier. We rather focus on invariant transformation by exploiting our experience on Craig's interpolation [11] in model checking.

With respect to [2], [3], [4], we share the idea of augmenting the invariant with terms representing the behaviour in the missing (lost/removed in transformations) time frames. But we do not need a witness circuit. This is possible because we exploit a quantification operator (the interpolant) that allows projecting invariants on existing state variables only, with no need for additional inputs and circuitry. Moreover, transformations are done by an external tool, so that the extra work in charge of the model checker is minimized. Invariant transformations have a time/memory cost, depending on the model and the transformation done, but this also happens in the Certifaiger approach. Our approach has an additional advantage: the possibility to associate the invariant to the original circuit, which opens the door to possibly reusing it for other tasks and/or other properties on the same model.

Invariant manipulation and simplification by size reduction is highly reliant on our previous works on interpolant optimization by weakening and strengthening [12], [13]. The work is also related to various papers that exploit invariants in improving the efficiency and scalability of model-checking techniques. Invariants have been used to support various verification methods, including Bounded Model Checking (BMC) [14], [15], k-induction [16], [17], predicate abstraction [18], interpolation [19], and IC3/PDR [20]. The core idea behind these approaches is to restrict the search space that a verification algorithm must explore when proving a safety property, thereby improving the verification efficiency.

B. CONTRIBUTION

In this paper, we introduce a new technique, called *inductive invariant lifting*, and a set of rules for transforming computed invariants to accommodate model transformations. The proposed approach leverages interpolation to generate new invariants that can incorporate the parts of state space and behaviour lost (or missing) due to model transformations. With respect to literature works [1], [2], [3], [4], which modify the model in order to avoid existential quantification through inefficient QBF (Quantified Boolean Formula) operators, we use the intrinsic existential quantification capabilities of interpolation to avoid building a transformed model that includes the behaviour of the original. Consequently, our approach produces (lifted) invariants that are inductive on the original model and can therefore be reused for other tasks, such as logic synthesis or supporting additional model checking tasks. We also exploit our previous work [12], in implementing a simplification/optimization procedure for the lifted invariants. The procedure enables us to fine-tune inductive invariants by reducing their size, enhancing their strength, or both.

Figure 1 illustrates the sequential workflow of the multi-step approach. Data flow from left to right through the distinct phases: model transformations, verification via model checking, invariant lifting, and eventual fine-tuning. In the figure, the original model ($\mathcal{S}$) is the sole input of the flow. Different intermediate data may be generated as a result of transformations (Ω) alongside the new model ($\hat{\mathcal{S}}$). Verification is responsible for generating invariants ($\hat{F}$) for the transformed model, before moving on to lifting such invariants to the original model (F) and eventual fine-tuning (F_s).

In summary, in this paper, we make the following contributions:

1) A novel technique based on Craig interpolation to transform existing invariants and lift them back to the original model prior to transformations;
2) A procedure for fine-tuning the lifted invariants;
3) An experimental evaluation demonstrating the effectiveness of the proposed approaches.

C. OUTLINE

The paper is organized as follows. Section II introduces the notation used throughout the paper, as well as some preliminary concepts. Section III introduces our novel approach to invariants manipulation and rewriting. Section IV presents the experimental evaluation of the proposed techniques on state-of-the-art hardware model checking benchmarks. Section V draws conclusions on the proposed techniques, provides some insight, and outlines possible future work.

II. PRELIMINARIES

A. TRANSITION SYSTEMS AND NOTATION

We take into account systems modelled by labelled state transition structures, implicitly represented using Boolean (propositional) logic. We rely on the standard notions of formulas, truth assignments, satisfiability, validity, and models within Boolean logic.

Definition 1: A *transition system* $\mathcal{S}$ is a triple $\langle X, \mathcal{I}, T \rangle$, where X is a set of Boolean *state variables*, $\mathcal{I}(X)$ is a Boolean formula representing the set of *initial states* and $T(X, X')$ is a Boolean formula representing the *transition relation* of the system.

A state of the system can therefore be represented as a complete truth assignment s over X, while sets of states are naturally expressed by Boolean formulas over X. We denote the state space of $\mathcal{S}$ by Space($\mathcal{S}$). Given a Boolean formula F over X and a complete truth assignment s such that $s \models F$, the assignment s corresponds to a state in the set represented by F, and can be referred to as an *F-state*. We denote with Vars(F) the *support* of a Boolean formula F. Future states of $\mathcal{S}$, i.e., those reached after a single transition from the current states, are represented using primed state variables X'. Accordingly, Boolean formulas over X' are used

to characterize sets of future states. When reasoning across multiple time frames, we adopt a superscript notation: the variables instantiated at the i-th time frame are denoted by X^i. By extension, given a Boolean formula F, we denote with F' (F^i) the corresponding formula defined on the next state variables ($i-$th time frame variables). We use the notation $F[X \mapsto Y]$ to denote the formula obtained from F by replacing each occurrence of $x \in X$ with the corresponding $y \in Y$. For simplicity and conciseness, the support variables are omitted whenever they can be inferred unambiguously from the context.

A (finite) *path* of $\mathcal{S}$ is a sequence of states $s_0, s_1, \ldots, s_n$ such that $s_0 \models \mathcal{I}$ and $(s_i, s_{i+1}) \models T$ for all $0 \leq i < n$. Given a path $\pi_n := s_0, s_1, \ldots, s_n$ we denote with $\pi_n[i]$ the state s_i.

Definition 2: A *path formula* of length $k = j - i$, starting from time frame i and reaching time frame j, is the propositional formula $\Pi_{i,j}$ over $X^i \cup \cdots \cup X^j$:

$$\Pi_{i,j} = \bigwedge_{k=i}^{j-1} T(X^k, X^{k+1}) \tag{1}$$

A state $s \in \text{Space}(\mathcal{S})$ is *reachable* in $\mathcal{S}$ if there exists a finite path π_n such that $\pi_n[n] = s$. The set of all reachable states of $\mathcal{S}$ is denoted by $\mathcal{R}(\mathcal{S})$. We denote by $\mathcal{R}_i^E(\mathcal{S})$ the set of states reachable in *exactly* i transitions, and by $\mathcal{R}_i(\mathcal{S})$ the set of states reachable *within* i transitions, i.e.,

$$\mathcal{R}_i(\mathcal{S}) \stackrel{\text{def}}{=} \bigcup_{0 \leq j \leq i} \mathcal{R}_j^E(\mathcal{S}).$$

When unambiguous, we omit $\mathcal{S}$ when referring to a given set of reachable states.

Definition 3: An *exact reach ring* R_i^E is a Boolean formula that holds true in all states reachable in exactly i transitions in $\mathcal{S}$, i.e.,

$$R_i^E(X) \stackrel{\text{def}}{=} \exists X^0 \ldots \exists X^{i-1} \left(\mathcal{I}(X^0) \wedge \Pi_{0,i}\right)[X^i \mapsto X]$$

Definition 4: A *reach ring* R_i is a Boolean formula that holds true in all states reachable within i transitions in $\mathcal{S}$, i.e.,

$$R_i(X) \stackrel{\text{def}}{=} \bigvee_{k=0}^{i} R_k^E(X)$$

With a slight abuse of notation, we occasionally use R_i^E and $\mathcal{R}_i^E$ (resp. R_i and $\mathcal{R}_i$) interchangeably.

Definition 5: Given a transition system $\mathcal{S}$, an *invariant property* P is a Boolean formula that must hold true in all reachable states s of $\mathcal{S}$, i.e.,

$$\forall s \in \mathcal{R}(\mathcal{S}) : s \models P$$

We denote as *target* the set of states represented by $\neg P$. We informally call *bad* states those states, or set of states, that either belong to the target or that can reach it.

Definition 6: Given a transition system $\mathcal{S}$ and an invariant property P, a Boolean formula F is *safe* with respect to P iff F is stronger than P, i.e., $F \rightarrow P$.

Definition 7: Given a transition system $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ and a Boolean formula F, F is an *inductive invariant* of $\mathcal{S}$ if the following conditions hold:

$$\mathcal{I} \rightarrow F \qquad \textit{(Base step)}$$
$$F \wedge T \rightarrow F' \qquad \textit{(Inductive step)}$$

An inductive invariant F of $\mathcal{S}$ can be seen as an over-approximation of the reachable states set $\mathcal{R}(\mathcal{S})$.

Definition 8: Given a transition system $\mathcal{S}$ and an invariant property P, an inductive invariant F of $\mathcal{S}$ is an *inductive strengthening* of P if it is safe with respect to P.

Definition 9: Given a transition system $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$, let $\equiv$ be an equivalence relation over $X \cup \{\top, \bot\}$ where:

- $x_i \equiv x_j$ indicates that the truth value of $x_i, x_j \in X$ is equal in all reachable states of $\mathcal{S}$, i.e., $\forall s \in \mathcal{R}(\mathcal{S}) : s(x_i) = s(x_j)$;
- $x \equiv \top$ indicates that x is always true, i.e., $\forall s \in \mathcal{R}(\mathcal{S}) : s(x) = \top$;
- $x \equiv \bot$ indicates that x is always false, i.e., $\forall s \in \mathcal{R}(\mathcal{S}) : s(x) = \bot$.

The equivalence relation $\equiv$ partitions $X \cup \{\top, \bot\}$ into k disjoint sets of *equivalence classes*:

$$\mathcal{C} = \{C_1, \ldots, C_k\}$$

where each class $C_i \subseteq X \cup \{\top, \bot\}$ with the constraint that at most one constant appears in each class. Let $[x]_\equiv$ denote the equivalence class of $x \in X \cup \{\top, \bot\}$.

Definition 10: Let L be a set of representative variables, arbitrarily selected for each equivalence class with no constant members:

$$L = \{r \in C \mid C \in \mathcal{C} \text{ and } \top, \bot \notin C\}$$

we define the *representative mapping function* $\rho : X \rightarrow L \cup \{\top, \bot\}$:

$$\rho(x) := \begin{cases} x, & \text{if } x \in L \\ l, & \text{if } x \notin L, l \in L \text{ and } l \in [x]_\equiv \\ \top, & \text{if } x \in [\top]_\equiv \\ \bot, & \text{if } x \in [\bot]_\equiv \end{cases}$$

With a slight abuse of notation, we extend the representative mapping function ρ to sets of variables by defining

$$\rho(X) \stackrel{\text{def}}{=} \{\rho(x) \mid x \in X\}.$$

B. CRAIG'S INTERPOLANTS

Craig's interpolation theorem is a key result in mathematical logic that connects model theory and proof theory. Despite its strong theoretical basis, interpolation has many practical applications in computer science. The author [21] first stated the theorem for first-order logic, while versions of the theorem exist for other logics too, including propositional logic, which we focus on as it is common in model checking.

Craig's interpolation theorem provides a high-level mechanism in hardware model checking by generating intermediate

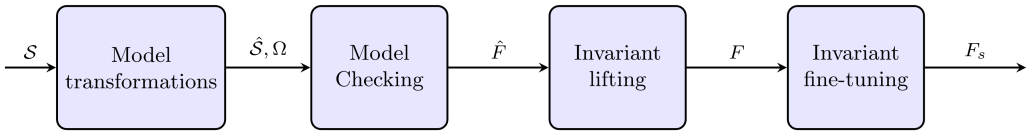


FIGURE 1. Flow diagram of the multi-step procedure consisting of model transformations, verification through model checking, invariant lifting, and fine-tuning steps.

predicates, called interpolants, from SAT solvers unsatisfiability proofs. Interpolants serve as overapproximations of reachable states, enabling the construction of inductive invariants without computing the full image of the transition relation. Such an approach is particularly effective for verifying safety properties in large-scale hardware designs, as it leverages efficient SAT-based bounded model checking while abstracting irrelevant details.

Definition 11: Given two Boolean formulas A and B, if they are inconsistent, i.e., if $A \wedge B$ is unsatisfiable, then there exists a Boolean formula I, called an *interpolant* between A and B, such that:

- $A \rightarrow I$ is valid
- $I \wedge B$ is unsatisfiable
- $\text{Vars}(I) \subseteq \text{Vars}(A) \cap \text{Vars}(B)$

In bounded model checking, a SAT solver refutes counterexamples up to depth k by proving $A \wedge B$ unsatisfiable, where A encodes initial states and traces to depth k, and B violates the property. An interpolant I satisfies $A \implies I$ and $I \wedge T \implies B$. Intuitively, I represents an abstraction of A with respect to B. The interpolant I captures and expresses, in the language common to both A and B, the reason for their mutual inconsistency.

In the following, we write $I = Itp(A, B)$ to denote a procedure that computes a Craig's interpolant from a pair of inconsistent formulas A and B.

Interpolants can be directly extracted from refutation proofs produced by SAT solvers on unsatisfiable runs. Given an unsatisfiable formula of the form $A \wedge B$, most modern SAT solvers can generate a refutation proof from which an interpolant $I = Itp(A, B)$ can be derived in polynomial time and space relative to the size of the proof. The interpolant is typically encoded as an AND/OR combinational circuit. In the context of model checking, if A represents a set of reachable states and B represents a set of bad states, the interpolant I can be viewed as a safe overapproximation of the reachable states. Such overapproximations can, in turn, be used to detect reachability fixpoints. McMillan [22] introduced the first complete algorithm for symbolic model checking that exploits Craig's interpolation. This algorithm, commonly referred to as *standard interpolation* or ITP, derives interpolants from refutation proofs of unsatisfiable BMC instances to compute overapproximations of the system's reachable states. Such an interpolation-based algorithm consists of an iterative process that builds inductive invariants incrementally, using SAT solvers and Craig interpolants. The algorithms starts from a weak invariant, such as the initial states, then repeatedly check if the invariant plus the transition relation can reach a bad state. At each iteration, the algorithm queries a SAT solver to determine if there exists a path of length up to a given bound from the current invariant to a property violation. The process recurses with an updated invariant, accounting for the computed interpolants, and an increased bound until no path to unsafe states is found, yielding a sound inductive invariant.

C. INTERPOLATION SEQUENCES

In [23], Vizel et al. propose a complete algorithm for symbolic model checking based on interpolation-sequences, which extend Craig's interpolation to formulas expressed as a conjunction of multiple factors.

Definition 12: Let $\Gamma = \{A_1, A_2, \ldots, A_n\}$ be a set of formulas such that $A_1 \wedge \cdots \wedge A_n$ is unsatisfiable, an *interpolation-sequence* for Γ is a set $\{I_0, I_1, \ldots, I_n\}$ such that:

- $I_0 \equiv \top$ and $I_n \equiv \bot$
- $I_i \wedge A_{i+1} \rightarrow I_{i+1}$ for each $0 \leq i < n$
- $\text{Vars}(I_i) \subseteq \text{Vars}(A_i) \cap \text{Vars}(A_{i+1})$ for each $0 < i < n$

We write $\{I_0, I_1, \ldots, I_n\} = ItpSeq(\Gamma)$ to denote a procedure that computes an interpolation sequence for Γ. In [23], the interpolation sequence is obtained from the refutation proof of a single SAT solver run, whereas in [24] it is constructed by iterating standard Craig's interpolation steps.

D. MODEL TRANSFORMATIONS

Let $\mathcal{S}$ be a transition system and let τ denote a transformation parametrized by a set of parameters θ. We denote the system obtained by applying the transformation τ to $\mathcal{S}$ with parameters θ as $\hat{\mathcal{S}}_{\tau,\theta}$. When a generic transformation is considered, or both the transformation and its parameters are clear from the context, we denote the transformed model simply as $\hat{\mathcal{S}}$. The same notation is applied to Boolean formulas before and after the transformation as well.

1) TEMPORAL DECOMPOSITION

Temporal decomposition [25] is a technique for simplifying a transition system by identifying and eliminating logic associated with initialization or reset sequences in a circuit. This logic typically governs operations restricted to the initial phases of execution and can therefore be ignored when analyzing the system's steady-state behaviour. The key idea is to determine the length of the reset sequence, denoted by the *transient duration* parameter k, measured in number of transitions. Once k is established, and it is verified that no bad states are reachable within k transitions from initial states, the initial states may be redefined as the set of states reachable

within k steps. This transformation produces a simplified transition system that more accurately captures the system's post-initialization behaviour and can be further reduced by detecting variable and constant equivalences that hold beyond the initial transient.

Given a transition system $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$, an invariant property P and a parameter k indicating the transient duration, applying *temporal decomposition* results in a new transition system $\hat{\mathcal{S}}_{td,k} = \langle \hat{X}_{td,k}, \hat{\mathcal{I}}_{td,k}, \hat{T}_{td,k} \rangle$ and property $\hat{P}_{td,k}$, such that:

- $\hat{\mathcal{I}}_{td,k}$ is the set of states reachable in exactly k steps from the initial states, i.e., $\hat{\mathcal{I}}_{td,k} \stackrel{\text{def}}{=} R_k^E$;
- $\hat{X}_{td,k}$, $\hat{T}_{td,k}$ and $\hat{P}_{td,k}$ are left unchanged with respect to the original system and property, i.e., $\hat{X}_{td,k} \stackrel{\text{def}}{=} X$, $\hat{T}_{td,k} \stackrel{\text{def}}{=} T$ and $\hat{P}_{td,k} \stackrel{\text{def}}{=} P$.

We assume that a model checker, when proving that a property $\hat{P}_{td,k}$ is an invariant of a system $\hat{\mathcal{S}}_{td,k}$ after temporal decomposition, produces as output both a formula $\hat{F}_{td,k}$ representing an inductive strengthening of $\hat{P}_{td,k}$ and a numerical value k corresponding to the transient duration parameter used in temporal decomposition.

2) PHASE ABSTRACTION

Phase abstraction [26], [27] is a simplification technique that operates by detecting and removing periodic signals that exhibit clock-like behaviours and partitioning the system according to the identified *number of phases* ϕ. Using phase abstraction, it is possible to remodel the system under verification in order to bypass hard-to-verify behaviours that arise from managing multi-phase systems.

Given a transition system $\mathcal{S}$, an invariant property P and a parameter ϕ indicating the number of phases, applying *phase abstraction* results in a new transition system $\hat{\mathcal{S}}_{pa,\phi} = \langle \hat{X}_{pa,\phi}, \hat{\mathcal{I}}_{pa,\phi}, \hat{T}_{pa,\phi} \rangle$ and property $\hat{P}_{pa,\phi}$, such that:

- $\hat{T}_{pa,\phi}$ is a composition of ϕ instances of the original transition relation T, i.e.,

$$\hat{T}_{pa,\phi}(X^0, X^\phi) \stackrel{\text{def}}{=} \exists X^1 \ldots \exists X^{\phi-1} \bigwedge_{i=0}^{\phi-1} T(X^i, X^{i+1})$$

- $\hat{X}_{pa,\phi}$ and $\hat{\mathcal{I}}_{pa,\phi}$ are left unchanged with respect to the original system and property, i.e., $\hat{X}_{pa,\phi} \stackrel{\text{def}}{=} X$, $\hat{\mathcal{I}}_{pa,\phi} \stackrel{\text{def}}{=} \mathcal{I}$;
- $\hat{P}_{pa,\phi}$ is expanded to include all states reaching a safe state (with respect to the original property P) in up to $\phi - 1$ transitions, i.e.,

$$\hat{P}_{pa,\phi} \stackrel{\text{def}}{=} P(X^0) \wedge \bigwedge_{i=1}^{\phi-1} \exists X^1, \ldots, X^i \left(\Pi_{0,i} \wedge P(X^i) \right)$$

We assume that a model checker, when proving that a property $\hat{P}_{pa,\phi}$ is an invariant of a system $\hat{\mathcal{S}}_{pa,\phi}$ after phase abstraction, produces as output both a formula $\hat{F}_{pa,\phi}$ representing an inductive strengthening of $\hat{P}_{pa,\phi}$ and a numerical value ϕ corresponding to the number of phases parameter used in phase abstraction.

3) EQUIVALENCE AND CONSTANT PROPAGATION

Several preprocessing techniques aim to simplify a model by identifying and propagating equivalences, either among state variables or between state variables and constants. Various methods can be used to determine the set of equivalence classes $\mathcal{C}$, including SAT-based refinement [28], [29] and ternary simulation [30]. Equivalences may also emerge as a by-product of other transformation techniques, such as temporal decomposition or phase abstraction. For example, in a two-phase system, abstracting one clock phase will render the latch associated with the clock signal constant.

Given a set of equivalence classes $\mathcal{C}$, a set of (non-constant) class representatives L, their representative mapping function ρ, a transition system $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ and an invariant property P, applying *variable and constant equivalence simplification* results in a new transition system $\hat{\mathcal{S}}_{eq,\mathcal{C}} = \langle \hat{X}_{eq,\mathcal{C}}, \hat{\mathcal{I}}_{eq,\mathcal{C}}, \hat{T}_{eq,\mathcal{C}} \rangle$ and a new property $\hat{P}_{eq,\mathcal{C}}$, such that:

- $\hat{X}_{eq,\mathcal{C}}$ is the reduced set of state variables composed of only equivalence class representatives, i.e., $\hat{X}_{eq,\mathcal{C}} \stackrel{\text{def}}{=} L$;
- $\hat{\mathcal{I}}_{eq,\mathcal{C}}$, $\hat{T}_{eq,\mathcal{C}}$ and $\hat{P}_{eq,\mathcal{C}}$ are obtained by replacing all variables X with their representatives $\rho(X)$, i.e.:

$$\begin{aligned} \hat{\mathcal{I}}_{eq,\mathcal{C}} &\stackrel{\text{def}}{=} \mathcal{I}[X \mapsto \rho(X)] \\ \hat{T}_{eq,\mathcal{C}} &\stackrel{\text{def}}{=} T[X \mapsto \rho(X), X' \to \rho(X')] \\ \hat{P}_{eq,\mathcal{C}} &\stackrel{\text{def}}{=} P[X \mapsto \rho(X)] \end{aligned}$$

  and simplifying the resulting formulas using Boolean rewriting.

Given that the transformation only removes variables from the original system, it is easy to prove the following lemmas.

Lemma 1: Given a transition system $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ and the corresponding transformed system $\hat{\mathcal{S}}_{eq,\mathcal{C}} = \langle \hat{X}_{eq,\mathcal{C}}, \hat{\mathcal{I}}_{eq,\mathcal{C}}, \hat{T}_{eq,\mathcal{C}} \rangle$ after variable and constant equivalence simplification, the following formulae are valid:

$$\begin{aligned} \mathcal{I} &\to \hat{\mathcal{I}}_{eq,\mathcal{C}} \\ \varepsilon \wedge \hat{P}_{eq,\mathcal{C}} &\to P \\ \varepsilon \wedge \hat{T}_{eq,\mathcal{C}} &\to T \end{aligned}$$

with ε defined as:

$$\varepsilon \stackrel{\text{def}}{=} \bigwedge_{x \in X} x \leftrightarrow \rho(x)$$

We assume that a model checker, when proving that a property $\hat{P}_{eq,\mathcal{C}}$ is an invariant of a system $\hat{\mathcal{S}}_{eq,\mathcal{C}}$ after variable and constant equivalence simplification, produces as output both a formula $\hat{F}_{eq,\mathcal{C}}$ representing an inductive strengthening of $\hat{P}_{eq,\mathcal{C}}$ and an invariant ψ such that $\psi \to \varepsilon$ and ψ are inductive for $\mathcal{S}$.

III. INVARIANTS LIFTING AND REFINEMENT

When transformations such as those described in Section II-D are applied to a model $\mathcal{S}$, the resulting transformed model $\hat{\mathcal{S}}$ may lose parts of the original state space and behaviour. These discrepancies often arise from changes in the variable set, for instance through the elimination of variables during equivalence simplification. Consequently, an inductive strengthening $\hat{F}$ derived from the transformed model usually does not hold for the original system.

In this section, we introduce a novel method, called *inductive invariant lifting*, which takes as input a safe inductive invariant $\hat{F}$ obtained from a transformed transition system $\hat{\mathcal{S}}$ together with additional parameters characterizing the transformation, and produces a corresponding inductive invariant F for the original system $\mathcal{S}$, which is safe with respect to the original property P. Our approach is based on an interpolation step that reconstructs the missing components of the invariant as a sequence of interpolants, which are then combined with $\hat{F}$ to complete the lifting process. For each transformation technique considered, we show how to construct such lifted invariants and prove that they are valid inductive strengthenings for the original property and system. Finally, we demonstrate how state-of-the-art weakening and strengthening techniques, originally developed for controlling the size of interpolants, can be applied to fine-tune both the size and the strength of the lifted invariants.

A. TEMPORAL DECOMPOSITION

When temporal decomposition is applied to a transition system $\mathcal{S}$, the resulting transformed model $\hat{\mathcal{S}}$ differs from the original in that it omits the first k transitions, where k denotes the transient duration. As a result, the set of reachable states of the transformed system, $\mathcal{R}(\hat{\mathcal{S}})$, may miss states of the original system that are visited only during the transient and never revisited afterward. In particular, given an inductive invariant $\hat{F}$ of $\hat{\mathcal{S}}$ the implication

$$R_{k-1} \rightarrow \hat{F}$$

is not guaranteed to hold, meaning that $\hat{F}$ cannot, in general, be regarded as an inductive invariant for $\mathcal{S}$. However, we show that it is possible to reconstruct an over-approximation of R_{k-1} using interpolation-sequences and that, by combining this over-approximation with $\hat{F}$, we can obtain a valid inductive invariant for the original system that remains safe with respect to the property P.

First, we define an over-approximation of the transition relation, denoted T_P, as:

$$T_P \stackrel{\text{def}}{=} T \vee \neg P.$$

Intuitively, this amounts to adding dummy transitions from unsafe states ($\neg P$) to all states. By expanding the transition relation in this way, we alter its behaviour only in states that are unreachable under the property P. Therefore, T and T_P are equivalent with respect to the reachable state space. Given this definition, we establish an intermediate result that will later serve to justify the use of interpolation-sequences.

Theorem 1: Given a transition system $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ and property P, let $\hat{\mathcal{S}}$ and $\hat{P}$ be the transformed transition system and property after applying temporal decomposition with transient duration k and let $\hat{F}$ be an inductive strengthening of $\hat{P}$ on $\hat{\mathcal{S}}$, then the formula:

$$\mathcal{I} \wedge \bigwedge_{i=0}^{k} T_P \wedge \neg \hat{F} \tag{2}$$

is unsatisfiable.

Proof: By definition T_P and T are equivalent on reachable states, therefore we have that:

$$\mathcal{I} \wedge \bigwedge_{i=0}^{k} T_P \rightarrow R_k^E \tag{3}$$

is valid, with R_k^E representing the set of state reachable in exactly k steps in $\mathcal{S}$. From the definition of $\hat{\mathcal{I}}$ in II-D1, the base step for the inductive invariant $\hat{F}$ is equivalent to:

$$R_k^E \rightarrow \hat{F} \tag{4}$$

From (3) and (4) we can derive:

$$\mathcal{I} \wedge \bigwedge_{i=0}^{k} T_P \rightarrow \hat{F}$$

and subsequently, by conjoining $\neg \hat{F}$ to either sides of the implication:

$$\mathcal{I} \wedge \bigwedge_{i=0}^{k} T_P \wedge \neg \hat{F} \rightarrow \bot$$

which proves the unsatisfiability of (2). □

Theorem 1 ensures that an interpolation-sequence can always be computed for $\Gamma = \{\mathcal{I} \wedge T_P, T_P, \ldots, T_P \wedge \neg \hat{F}\}$. We now show how this interpolation-sequence can be exploited to construct an inductive invariant F for the original system.

Theorem 2: Let $\hat{F}$ be an inductive strengthening of $\hat{P}$ on $\hat{\mathcal{S}}$, and let $\{I_0, I_1, \ldots, I_k\} = \mathit{ItpSeq}(\Gamma)$ be the interpolation sequence computed for $\Gamma \stackrel{\text{def}}{=} \{\mathcal{I} \wedge T_P, T_P, \ldots, T_P \wedge \neg \hat{F}\}$. Then, the formula

$$F \stackrel{\text{def}}{=} \hat{F} \vee \mathcal{I} \vee \bigvee_{i=1}^{k-1} I_i \tag{5}$$

is an inductive invariant of the original system $\mathcal{S}$.

Proof: The base step of induction for F follows trivially from its definition (5), where $\mathcal{I}$ appears as a disjunctive term. Focusing on the inductive step, we need to prove that:

$$F \wedge T \rightarrow F'$$

We do so by considering F as the disjunction of two terms $\hat{F}$ and $\mathcal{I} \wedge \bigvee_{i=0}^{k-1} I_i$ and proving the inductive step partially for each term before merging them together.

From the definition of F it follows that:

$$\hat{F} \rightarrow F \tag{6}$$

From the inductive step of $\hat{F}$ and (6) we can derive:

$$\hat{F} \wedge T \rightarrow F' \tag{7}$$

which proves the (partial) inductive step for the first term of F.

Given $\Gamma = \{\mathcal{I} \wedge T_P, T_P, \ldots, T_P \wedge \neg\hat{F}\}$, it follows from the definition of interpolation-sequence that, for each $i \in \{1, \ldots, k-2\}$:

$$\mathcal{I} \wedge T_P \rightarrow I_1' \tag{8}$$

$$I_i \wedge T_P \rightarrow I_{i+1}' \tag{9}$$

and

$$I_{k-1} \wedge T_P \rightarrow \hat{F}' \tag{10}$$

From the definition of T_P, it follows that $T \rightarrow T_P$, therefore we can be safely replace T_P with T in (8), (9), and (10). By disjoining the antecedents and consequents of the resulting implications and factoring T we obtain:

$$\Big(\mathcal{I} \vee \bigvee_{i=1}^{k-1} I_i\Big) \wedge T \rightarrow \bigvee_{i=1}^{k-1} I_i' \vee \hat{F}' \tag{11}$$

It stems from the definition of F that:

$$\bigvee_{i=1}^{k-1} I_i \vee \hat{F} \rightarrow F \tag{12}$$

Therefore, from (11) and (12) we can derive:

$$\Big(\mathcal{I} \vee \bigvee_{i=1}^{k-1} I_i\Big) \wedge T \rightarrow F' \tag{13}$$

which proves the (partial) inductive step for the second term of F. Finally, the inductive step F for $\mathcal{S}$ follows by disjoining the antecedents and consequents of (7) and (13) and factoring T. □

Finally, we need to demonstrate that the lifted inductive invariant F is safe with respect to the property P, i.e., $F \rightarrow P$. Note that, when applying temporal decomposition with transient duration k, the property P is first checked, using BMC or equivalent techniques, to hold throughout the initial k transitions. As a result, by assumption we know that:

$$R_{k-1} \rightarrow P \tag{14}$$

Note that this assumption must be considered as an additional proof-obligation to be discharged by the proof-checker when F is used to prove safety with respect to P in the original system.

Theorem 3: Let $\hat{F}$ be an inductive strengthening of $\hat{P}$ for $\hat{\mathcal{S}}$, and let F be the lifted inductive invariant for $\mathcal{S}$ derived from $\hat{F}$ as defined in Theorem 2. Then F is safe with respect to the original property P and, consequently, constitutes an inductive strengthening of P for $\mathcal{S}$.

Proof: Since $\hat{F}$ is an inductive strengthening of $\hat{P}$, and temporal decomposition does not alter the property definition ($P = \hat{P}$, see II-D1), it follows that

$$\hat{F} \rightarrow P \tag{15}$$

From the assumption in (14), it follows that:

$$\mathcal{I} \rightarrow P \tag{16}$$

By the construction of the interpolation-sequence, the definition of Γ and the definition of T_P, for each $i \in \{1, \ldots, k-1\}$, we have that:

$$I_i \rightarrow P \tag{17}$$

Intuitively, this follows from the fact that each I_i, being an interpolant, renders the formula $I_i \wedge B$ unsatisfiable, where $B \stackrel{\text{def}}{=} \bigwedge_{i=0}^{k-1} T_P \wedge \neg\hat{F}$. If (17) were not valid for some i, i.e., if there existed at least one unsafe I_i-state, then the formula $I_i \wedge B$ would be satisfiable, since T_P would allow any state to be reached from an unsafe state after an arbitrary number of transitions. This would contradict the fact that I_i is an interpolant.

The safety of F with respect to P then follows from the fact that each of its disjunctive terms is safe with respect to P, as established by (15), (16), and (17). □

Algorithm 1 Top-Level Procedure for Invariant Lifting With Temporal Decomposition

Input: $\mathcal{S} = \langle X, \mathcal{I}, T\rangle$ the original transition system; P the original property; $\hat{\mathcal{S}} = \langle \hat{X}, \hat{\mathcal{I}}, \hat{T}\rangle$ the transformed transition system; $\hat{P}$ the transformed property; $\hat{F}$ an inductive strengthening of $\hat{P}$; k the transient duration.

Output: F a lifted inductive strengthening of P for $\mathcal{S}$.

```
1: procedure liftTdInvar(S, P, Ŝ, P̂, F̂, k)
2:     I_0 ← I
3:     T_P ← T ∨ ¬P
4:     for i = 1 to k − 1 do
5:         I_i ← Itp(I_{i−1} ∧ T_P, ⋀_{j=i}^{k−1} T_P ∧ ¬F̂)
6:     F ← ⋁_{i=0}^{k−1} I_i ∨ F̂
7:     return F
```

Algorithm 1 illustrates the steps required to lift an inductive strengthening obtained from a model after temporal decomposition.

B. PHASE ABSTRACTION

When phase abstraction with ϕ phases is applied to a transition system $\mathcal{S}$, the transformed model $\hat{\mathcal{S}}$ differs from the original in that its reachable state space $\mathcal{R}(\hat{\mathcal{S}})$ includes only those states reached at periodic time-frame boundaries of the original system, with period ϕ. Consequently, an inductive invariant $\hat{F}$ of $\hat{\mathcal{S}}$ is guaranteed to hold at every multiple of ϕ transitions, but there is no guarantee that $\hat{F}$ also holds at intermediate time frames. In particular, we cannot assume the validity of

$$R^E_{i+k\phi} \rightarrow \hat{F}$$

for arbitrary $i \in \{0, \ldots, \phi - 1\}$ and $k \in \mathbb{N}$. As a result, $\hat{F}$ cannot, in general, be regarded as an inductive invariant for $\mathcal{S}$. However, we show that it is possible to reconstruct an over-approximation of the intermediate states using interpolation sequences. By combining this over-approximation with $\hat{F}$, we obtain a valid inductive invariant for the original system that is still safe with respect to property P.

Using the same definition of T_P provided in III-A, first we establish an intermediate result that will allow us to compute the required interpolation-sequence.

Theorem 4: Given a transition system $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ and property P, let $\hat{\mathcal{S}}$ and $\hat{P}$ be the transformed transition system and property after applying phase abstraction with number of phases ϕ and let $\hat{F}$ be an inductive strengthening of $\hat{P}$ on $\hat{\mathcal{S}}$, then the formula:

$$\hat{F} \wedge \bigwedge_{i=0}^{\phi} T_P \wedge \neg \hat{F} \tag{18}$$

is unsatisfiable.

Proof: Unsatisfiability of (18) follows directly from the fact that it requires both $\hat{F}$ and $\neg\hat{F}$ to hold simultaneously, which is contradiction. □

Theorem 4 ensures that an interpolation-sequence can always be computed for $\Gamma = \{\hat{F} \wedge T_P, T_P, \ldots, T_P \wedge \neg\hat{F}\}$. We now show how this interpolation-sequence can be exploited to construct an inductive invariant F for the original system.

Theorem 5: Let $\hat{F}$ be an inductive strengthening of $\hat{P}$ on $\hat{\mathcal{S}}$, and let $\{I_0, I_1, \ldots, I_\phi\} = ItpSeq(\Gamma)$ be the interpolation sequence computed for $\Gamma \stackrel{\text{def}}{=} \{\hat{F} \wedge T_P, T_P, \ldots, T_P \wedge \neg\hat{F}\}$. Then, the formula

$$F \stackrel{\text{def}}{=} \hat{F} \vee \bigvee_{i=1}^{\phi-1} I_i \tag{19}$$

is an inductive invariant of the original system $\mathcal{S}$.

Proof: The base step follows from the definition of F and the base step of $\hat{F}$. Focusing on the inductive step, we need to prove that:

$$F \wedge T \rightarrow F' \tag{20}$$

Given $\Gamma = \{\hat{F} \wedge T_P, T_P, \ldots, T_P \wedge \neg\hat{F}\}$, it follows from the definition of interpolation-sequence that, for each $i \in \{1, \ldots, \phi - 2\}$:

$$\hat{F} \wedge T_P \rightarrow I_1' \tag{21}$$

$$I_i \wedge T_P \rightarrow I_{i+1}' \tag{22}$$

and

$$I_{\phi-1} \wedge T_P \rightarrow \hat{F}' \tag{23}$$

From (21), (23), and (22), by safely replacing T_P with T, disjoining the antecedents and consequents of the resulting implications and factoring T we have:

$$\left(\hat{F} \vee \bigvee_{i=1}^{\phi-1} I_i\right) \wedge T \rightarrow \hat{F}' \vee \bigvee_{i=1}^{\phi-1} I_i' \tag{24}$$

which proves the inductive step of F for $\mathcal{S}$. □

Finally, we need to demonstrate that the lifted inductive invariant F is safe with respect to the property P, i.e., $F \rightarrow P$.

Theorem 6: Let $\hat{F}$ be an inductive strengthening of $\hat{P}$ for $\hat{\mathcal{S}}$, and let F be the lifted inductive invariant for $\mathcal{S}$ derived from $\hat{F}$ as defined in Theorem 5. Then F is safe with respect to the original property P and, consequently, constitutes an inductive strengthening of P for $\mathcal{S}$.

Proof: Since $\hat{F}$ is an inductive strengthening of $\hat{P}$, and the transformed property is stronger than the original we have that:

$$\hat{F} \rightarrow P \tag{25}$$

By the construction of the interpolation-sequence and the definition of T_P, for each $i \in \{1, \ldots, \phi - 1\}$, we have:

$$I_i \rightarrow P \tag{26}$$

This follows intuitively from the same argument provided in the proof of Theorem 3.

The safety of F with respect to P then follows from the fact that each of its disjunctive terms is safe with respect to P, as established by (25) and (26). □

Algorithm 2 Top-Level Procedure for Invariant Lifting With Phase Abstraction

Input: $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ the original transition system; P the original property; $\hat{\mathcal{S}} = \langle \hat{X}, \hat{\mathcal{I}}, \hat{T} \rangle$ the transformed transition system; $\hat{P}$ the transformed property; $\hat{F}$ an inductive strengthening of $\hat{P}$; ϕ the number of phases.

Output: F a reconstructed invariant for the original model.

```
1: procedure liftPaInvar(S, P, Ŝ, P̂, F̂, φ)
2:     I_0 ← F̂
3:     T_P ← T ∨ ¬P
4:     for i = 1 to φ − 1 do
5:         I_i ← Itp(I_{i−1} ∧ T_P, ⋀_{j=i}^{φ−1} T_P ∧ ¬F̂)
6:     F ← ⋁_{i=0}^{φ−1} I_i
7:     return F
```

Algorithm 2 illustrates the steps required to lift an inductive strengthening obtained from a model after phase abstraction.

C. VARIABLE AND CONSTANT EQUIVALENCE SIMPLIFICATION

In this section, we demonstrate how an inductive strengthening obtained from a model and property simplified by propagating variable and constant equivalences, can be lifted back to the original model and property. Let $\mathcal{S}$ and P be the original transition system and invariant property. Let $\hat{\mathcal{S}}$ and $\hat{P}$ be the corresponding transformed transition system and property after simplification via variable and constant

equivalences propagation as defined in II-D3, with $\mathcal{C}$ being the set of equivalence classes, L being the set of representative variables for each class and ε being an invariant formula representing the set of equivalences between variables in X.

Theorem 7: Given $\hat{F}$ an inductive invariant of $\hat{\mathcal{S}}$ and ψ an inductive invariant of $\mathcal{S}$ such that $\psi \rightarrow \varepsilon$, then the formula:

$$F \stackrel{\text{def}}{=} \psi \wedge \hat{F} \tag{27}$$

is an inductive invariant of $\mathcal{S}$.

Proof: First we prove the base step. By conjoining the antecedents and consequents of the base steps of the two inductive invariants $\hat{F}$ and ψ, we have:

$$\mathcal{I} \wedge \hat{\mathcal{I}} \rightarrow \psi \wedge \hat{F} \tag{28}$$

The base step for F with respect to $\mathcal{S}$ follows from (28) and Lemma 1:

$$\mathcal{I} \rightarrow \psi \wedge \hat{F}$$

Finally we prove the inductive step. As before, we start by conjoining the antecedents and consequents of the inductive steps of $\hat{F}$ and ψ:

$$\psi \wedge \hat{F} \wedge T \wedge \hat{T} \rightarrow \psi' \wedge \hat{F}' \tag{29}$$

The inductive step for F with respect to $\mathcal{S}$ follows from (29), Lemma 1 and $\psi \rightarrow \varepsilon$:

$$\psi \wedge \hat{F} \wedge T \rightarrow \psi' \wedge \hat{F}'$$

□

Theorem 8: Let $\hat{F}$ be an inductive strengthening of $\hat{P}$ for $\hat{\mathcal{S}}$, and let F be the lifted inductive invariant for $\mathcal{S}$ derived from $\hat{F}$ as defined in Theorem 7. Then F is safe with respect to the original property P and, consequently, constitutes an inductive strengthening of P for $\mathcal{S}$.

Proof: Since $\hat{F}$ is an inductive strengthening of $\hat{P}$, it holds that

$$\hat{F} \rightarrow \hat{P}\,.$$

By conjoining ψ to both sides of the implication, it follows from Lemma 1 and $\psi \rightarrow \varepsilon$ that:

$$\psi \wedge \hat{F} \rightarrow P$$

This establishes the safety of F with respect to P. □

D. FINE-TUNING OF LIFTED INVARIANTS

The lifting techniques presented above successfully produce safe inductive invariants for the original system from invariants obtained after applying various model transformations. However, the interpolation step often makes it difficult to control the size of the resulting invariant. To address this limitation, we complement our lifting approach with a *fine-tuning step*, in which weakening and strengthening techniques, originally proposed in [12] for interpolants, are applied directly to the lifted invariant to achieve a more balanced trade-off between size and strength.

In [12], the authors propose a SAT-based technique that, given an unsatisfiable formula of the form $A \wedge B$, constructs an interpolant I_s such that:

- $A \rightarrow I_s$
- $I_s \rightarrow \neg B$
- I_s is empirically smaller in size than $\neg B$.

We denote this technique as $ItpStrengthen(A, B)$. In this section, we demonstrate how it can be applied to a lifted safe inductive invariant F to derive a stronger invariant F_s that remains both inductive and safe with respect to the original system and property.

Let F be an inductive invariant of the original transition system $\mathcal{S}$. By the inductive step of F, it follows that $F \wedge T \wedge \neg F'$ is unsatisfiable. This observation leads to the following theorem.

Theorem 9: Given F and inductive invariant of the original transition system $\mathcal{S}$ that is safe with respect to the original property P, let I_s being the formula obtained by applying SAT-based strengthening to $A \wedge B$, with $A \stackrel{\text{def}}{=} F \wedge T$ and $B \stackrel{\text{def}}{=} \neg F'$, i.e.,

$$I_s' = ItpStrengthen(F \wedge T, \neg F')$$

then, the formula:

$$F_s \stackrel{\text{def}}{=} I_s \vee \mathcal{I} \tag{30}$$

is still a safe inductive invariant for the original model and property.

Proof: The base step of induction for F_s follows trivially from its definition, as $\mathcal{I}$ appears as a disjunctive term in (30). To prove the inductive step, we first show that the new invariant F_s is stronger than F. From the definition of B, and since I_s is stronger than $\neg B$, it follows that:

$$I_s' \rightarrow F' \tag{31}$$

From the base step of F and (31) after substituting for current state variables, it follows:

$$I_s \vee \mathcal{I} \rightarrow F$$

which proves $F_s \rightarrow F$. Since, by the definition of the strengthening technique, $A \rightarrow I_s'$, it follows from the definition of A that:

$$F \wedge T \rightarrow I_s' \tag{32}$$

Given that F_s is stronger than F, from (32) we can derive:

$$F_s \wedge T \rightarrow I_s' \tag{33}$$

and consequently:

$$F_s \wedge T \rightarrow I_s' \vee \mathcal{I}' \tag{34}$$

which proves the inductive step of F_s. The safety of F_s follows trivially from the fact that F_s is stronger than F. □

IV. EXPERIMENTAL EVALUATION

In this section, we present an experimental evaluation of our methodology, assessing its effectiveness, efficiency, and robustness using a series of tests on publicly available benchmark sets. The goal of this experimental evaluation is to evaluate whether: (1) the size of the reconstructed invariants is in the same order of magnitude of that of the original invariants, (2) the time required for reconstruction is reasonable, and (3) the fine-tuning step effectively reduces the size of the lifted invariants.

A. EXPERIMENTAL SETUP

We have implemented our invariant lifting and certification procedure on top of PdTRAV [31], a state-of-the-art model checking suite. The code is open source and publicly available [32]. PdTRAV takes as input a model in Aiger [33] format, and produces inductive invariants as AIG combinational circuits expressed over the state variables of the model.[1] The experiments were carried out on an Intel Core i7-1370, with 16 CPUs running at 5GHz, 16 GBytes of main memory DDR III, and hosting a Ubuntu 22.04 LTS Linux distribution. All experiments were carried out taking into account a time limit of 7200 seconds and a memory limit of 16 GB.

B. BENCHMARK SET

For our evaluation, we have collected a total of 150 problem instances from different benchmark families stemming from past Hardware Model Checking Competitions (HWMCC) [8]. To enable accurate and unambiguous tracking of all elements within the circuits under verification, we have incorporated an explicit symbol table into all selected base model instances. Each symbol table labels the inputs, outputs, and state elements of its corresponding circuit. The addition of this explicit symbol table was accomplished through a straightforward rewriting process that leverages ABC [34] as an auxiliary tool within the workflow. The set of selected benchmarks was determined by choosing UNSAT instances that could be verified and for which an invariant could subsequently be derived within a one-hour time limit. All benchmarks, inductive invariants, and run logs have been documented and made available for reference in [35].

C. EXPERIMENTAL RESULTS

For each benchmark model, we perform two runs: one using temporal decomposition and the other using phase abstraction. In the case of temporal decomposition, our implementation automatically infers a transient duration k for each model. For phase abstraction, a fixed number of phases $\phi = 2$ is used. Each run consists of three stages: first, a verification stage in which an inductive invariant is derived and the applied transformations are logged; second, a lifting stage, which produces a corresponding invariant for the original model and, third, a fine-tuning stage, in which the lifted invariant is compacted. In order to keep the verification and certification steps streamlined, we select a single model checking strategy to perform the verification phase. Specifically, we use PdTRAV's IC3 engine [36] without introducing any additional simplification steps. It is worth noticing that we do not exploit a portfolio of different engines, nor different tunings/variants of the IC3 engine. Also note that the transformation (temporal decomposition, phase abstraction) is not necessarily expected to provide an improvement vs. the original version of the circuit, so experimental results are not competing in terms of absolute performance.

Whenever a new inductive invariant is generated, whether during the verification or lifting phase, we also perform a certification step to ensure its correctness with respect to the target model and property.

For each run, we report the verification time as well as the time required for the lifting process and the subsequent fine-tuning step. For all steps, we also provide statistics on the size of the resulting inductive invariant. Sizes are expressed in terms of the number of AND gates necessary to represent the invariants in AIGER format.

Figures 2-4 compare the sizes of the generated and processed invariants, against each other and the original model, at various stages of our workflow: after verification, after lifting, and after fine-tuning. In each plot, the horizontal axis represents the size of the input (invariant or original model) for that stage, while the vertical axis represents the size of the resulting output invariant. In particular, Figure 2 compares the size of each benchmark instance with the invariant generated when tackling the corresponding model checking task. With very few exceptions, invariants are generally smaller than the related input model. A detailed analysis of the outliers (invariants larger than the circuit size) are typically hard to verify instances, where IC3 converges after generating a large amount of clauses: so either the instance is really a difficult one for all engines, or a smaller invariant could be generated by a better performing engine. We do not distinguish between the two cases here.

Figure 3 compares the size of the invariant produced by the model checker with that of the invariant generated during the lifting stage. As expected, as a result of the use of interpolation in the lifting process, we observe an increase in size. However, it is worth noting that this increase exceeds two orders of magnitude only for small invariants, with an initial size of up to a few thousand. For larger invariants, the increase is either negligible or limited to a factor of at most $10\times$. Figure 4 compares the size of the lifted invariant with that of the same invariant after the fine-tuning step described in III-D. The results show a size reduction in the range from 1 to 10, with just a few cases of more reduction. Finally, Figure 5 compares the size of the lifted and fine-tuned invariant with that of the original invariant produced by the model checker, illustrating the combined effect of invariant lifting and fine-tuning stages. A visual

[1] If model transformations are applied, such invariants are defined over the state variables *after* the transformations have been performed.

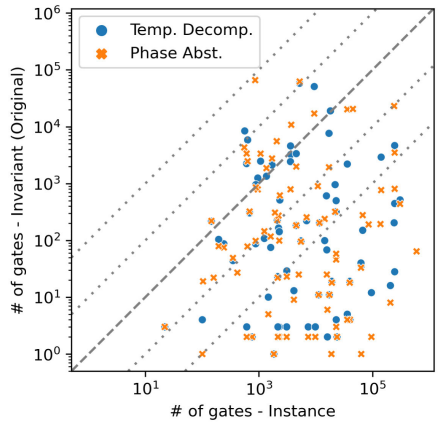


FIGURE 2. Size of each model checking instance against the size of the invariants generated by the model checker.

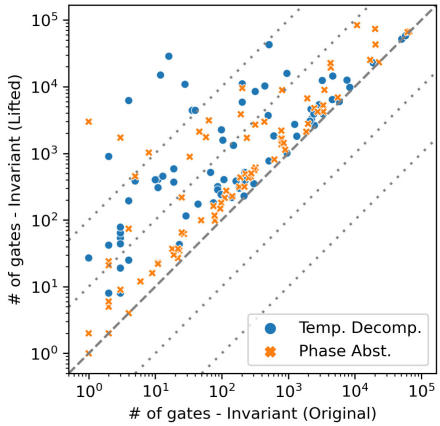


FIGURE 3. Size of each invariant generated during verification against the size of the corresponding lifted invariant.

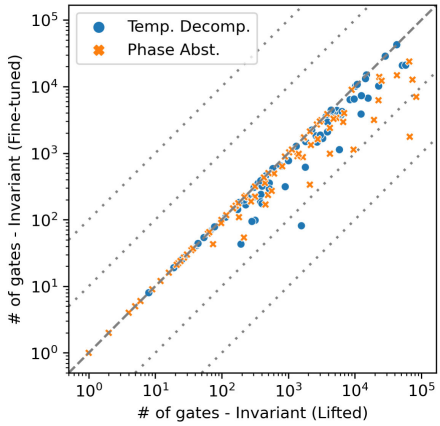


FIGURE 4. Size of each lifted invariant compared to the size of the corresponding lifted invariant after fine-tuning.

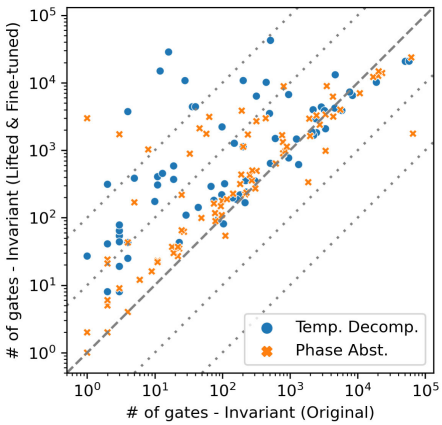


FIGURE 5. Size of each lifted and fine-tuned invariant against the size of the invariants generated by the model checker.

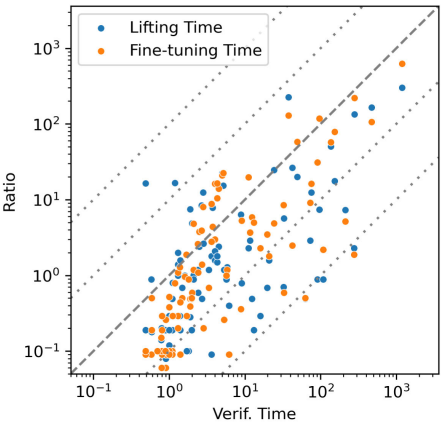


FIGURE 6. Comparison of verification times and combined lifting and fine-tuning times for each model instance and invariant pair.

comparison with Figure 3 clearly highlights the average gains achieved. it is also evident that for some benchmarks nothing changes, meaning that the invariant has no redundancy and no room for compaction is available.

In Figure 6, we compare the verification time with the combined time required for invariant lifting and fine-tuning. Verification times are shown on the horizontal axis, whereas the combined times for lifting and fine-tuning are shown on the vertical axis. This visual comparison allows us to identify patterns or trends, such as whether longer verification times correspond to extended lifting and fine-tuning times, or whether these stages operate independently in terms of time consumption. Ideally, the post-processing steps should require less time than the initial verification phase. However, notable exceptions exist: in some cases, the duration of reconstruction or simplification approaches, or even exceeds, that of verification. Focusing on times in the range of hundreds of seconds, it is clear that invariant manipulation times are generally less than or equal to the model checking times.

Finally, Figure 7 shows the cumulative distribution of times across all the chosen benchmarks needed to run the model checking tasks, the lifting stage, and the final fine-tuning stage. The fine-tuning curve turns out to be a bit more expensive than lifting, and the gap vs. verification times tend to narrow at difficult instances. A deeper analysis of the data showed that the hard instances in terms of time (both for lifting and fine-tuning) are the ones starting with large invariants, with almost no compaction, which are typically related to difficult SAT solver runs and large interpolants. As already noted, they could be really

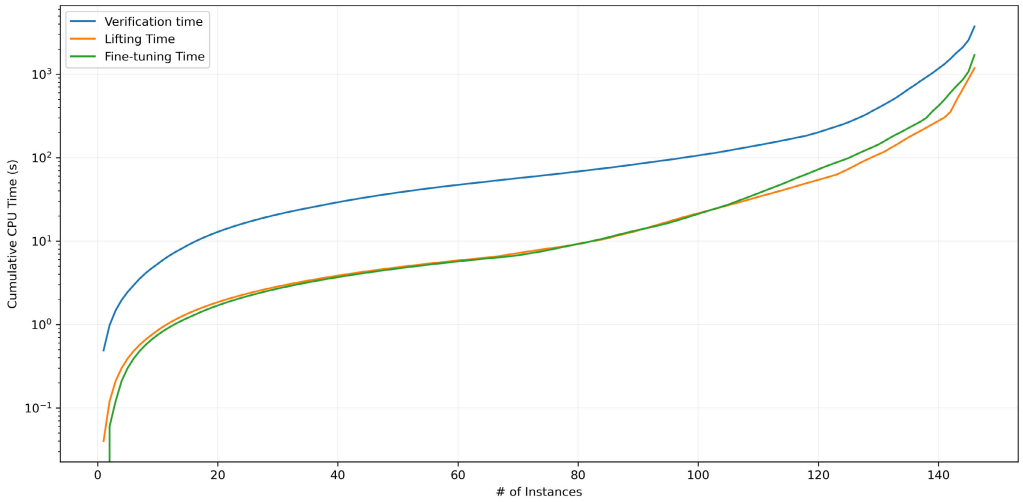


FIGURE 7. Cumulative time distribution of runs for verification, lifting and fine-tuning stages, across all benchmarks.

hard problems, whichever the engine end/or the adopted certification strategy.

1) SCALABILITY

Although scalability is obviously desirable, it is clear that certain large invariants, combined with large and hard-to-verify models, could be out of the scope of our techniques. In this respect, an extensive experimentation on scalability was out of the scope of this work, for the following reasons:

- The main purpose of this paper is to show the theoretical soundness of a novel approach, as well as its applicability to small-medium sized problems
- We expect that the potential usability of inductive invariants for other tasks (beyond pure certification) is likely limited to small-medium sized invariants, due to the limited overhead they generate
- We are not aware of any extensive experimentation on certification in the presence of transformations, with a specific focus on scalability. Even recent work [37], although very important in definitely showing the feasibility of certification, does not provide hints on transformations operated by model checkers, which hide this kind of information.

Covering scalability, with a portfolio of multiple engines, likely to output better (on average) inductive invariants for given hard-to-verify models, is one of the expected future work. Future work also include other transformations, such as retiming, signal correspondence, reparameterization, and sequences of transformations.

V. CONCLUSIONS AND FUTURE WORK

In this work, we tackle a critical challenge in the certification of hardware model checking results: leveraging proof certificates when the original model undergoes complex pre-processing and transformations. Unlike previous approaches that rely on generating witness models, we propose a method to fully lift inductive invariants back to the original model, simplifying the model checker's role to generate the final invariant along with a transformation log. To handle the existential quantification inherent in lifting, we employ Craig's interpolation to generate new inductive invariants and introduce a SAT-based fine-tuning step to reduce their size. Experimental results on a diverse set of publicly available benchmarks demonstrate the practicality and benefits of our approach, enabling the reuse of invariants for tasks such as logic synthesis optimizations and verifying additional properties.

In future work, our objective is to extend our experimental evaluation and the set of techniques covered to support additional engines, such as interpolation and IGR [11], and to include a larger set of more challenging instances. Furthermore, we plan to apply this approach to additional model transformation techniques.

REFERENCES

[1] E. Yu, A. Biere, and K. Heljanko, "Progress in certifying hardware model checking results," in *Proc. 33rd Int. Conf. Comput. Aided Verification*. Cham, Switzerland: Springer, 2021, pp. 363–386.
[2] E. Yu, N. Froleyks, A. Biere, and K. Heljanko, "Stratified certification for K-induction," in *Proc. Formal Methods Computer-Aided Design (FMCAD)*, Oct. 2022, pp. 59–64.
[3] E. Yu, N. Froleyks, A. Biere, and K. Heljanko, "Towards compositional hardware model checking certification," in *Proc. Formal Methods Comput.-Aided Design (FMCAD)*, Mar. 2023, pp. 1–11.
[4] N. Froleyks, E. Yu, A. Biere, and K. Heljanko, "Certifying phase abstraction," in *Proc. Int. Joint Conf. Automated Reasoning*. Cham, Switzerland: Springer, 2024, pp. 284–303.
[5] A. R. Bradley, F. Somenzi, Z. Hassan, and Y. Zhang, "An incremental approach to model checking progress properties," in *Proc. Formal Methods Computer-Aided Design (FMCAD)*, Oct. 2011, pp. 144–153.
[6] A. Biere, C. Artho, and V. Schuppan, "Liveness checking as safety checking," *Electron. Notes Theor. Comput. Sci.*, vol. 66, no. 2, pp. 160–177, Dec. 2002.
[7] N. Froleyks. (2024). *Certifaiger*. [Online]. Available: https://github.com/Froleyks/certifaiger
[8] A. Biere and T. Jussila. *The Model Checking Competition Web Page*. Accessed: Apr. 25, 2026. [Online]. Available: http://fmv.jku.at/hwmcc
[9] A. Griggio, M. Roveri, and S. Tonetta, "Certifying proofs for LTL model checking," in *Proc. Formal Methods Comput. Aided Design (FMCAD)*, Oct. 2018, pp. 1–9.

[10] G. Sindoni, P. Pasini, G. Cabodi, P. E. Camurati, A. Griggio, M. Palena, M. Roveri, and S. Tonetta, "A theorem prover based approach for sat-based model checking certification," in *Proc. 30th Int. Conf. Automated Deduction*, Stuttgart, Germany, 2025, pp. 449–467, doi: 10.1007/978-3-031-99984-0_24.

[11] G. Cabodi, P. E. Camurati, M. Palena, and P. Pasini, "Interpolation with guided refinement: Revisiting incrementality in SAT-based unbounded model checking," *Formal Methods Syst. Design*, vol. 60, no. 2, pp. 117–146, Dec. 2022, doi: 10.1007/s10703-022-00406-7.

[12] G. Cabodi, P. E. Camurati, M. Palena, P. Pasini, and D. Vendraminetto, "Reducing interpolant circuit size through SAT-based weakening," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 7, pp. 1524–1531, Jul. 2020.

[13] G. Cabodi, P. E. Camurati, M. Palena, P. Pasini, and D. Vendraminetto, "Logic synthesis for interpolant circuit compaction," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 2, pp. 380–384, Feb. 2019.

[14] M. Awedh and F. Somenzi, "Automatic invariant strengthening to prove properties in bounded model checking," in *Proc. 43rd Annu. Conf. Design Autom. - DAC*, 2006, pp. 1073–1076.

[15] G. Cabodi, P. Enrico Camurati, M. Palena, and P. Pasini, "Improving bounded model checking exploiting interpolation-based learning and strengthening," *IEEE Access*, vol. 12, pp. 119341–119349, 2024.

[16] M. Brain, S. Joshi, D. Kroening, and P. Schrammel, "Safety verification and refutation by K-invariants and K-induction," in *Proc. Int. Static Anal. Symp.* Cham, Switzerland: Springer, 2015, pp. 145–161.

[17] H. Rocha, H. Ismail, L. Cordeiro, and R. Barreto, "Model checking embedded c software using K-induction and invariants," in *Embedded Software Verification and Debugging*, New York, NY, USA: Springer, 2017, pp. 159–182.

[18] H. Jain, F. Ivančić, A. Gupta, I. Shlyakhter, and C. Wang, "Using statically computed invariants inside the predicate abstraction and refinement loop," in *Proc. 18th Int. Conf., Comput. Aided Verification*, Seattle, WA, USA. Cham, Switzerland: Springer, 2006, pp. 137–151.

[19] D. Beyer, P.-C. Chien, and N.-Z. Lee, "Augmenting interpolation-based model checking with auxiliary invariants," in *Proc. Int. Symp. Model Checking Softw.* Cham, Switzerland: Springer, 2024, pp. 227–247.

[20] A. Gurfinkel, T. Kahsai, A. Komuravelli, and J. A. Navas, "The seahorn verification framework," in *Proc. Int. Conf. Comput. Aided Verification*. Cham, Switzerland: Springer, 2015, pp. 343–361.

[21] W. Craig, "Three uses of the herbrand-gentzen theorem in relating model theory and proof theory," *J. Symbolic Log.*, vol. 22, no. 3, pp. 269–285, Sep. 1957.

[22] K. L. McMillan, "Interpolation and SAT-based model checking," in *Proc. Comput. Aided Verification*. Boulder, CO, USA: Springer, 2003, pp. 1–13.

[23] Y. Vizel and O. Grumberg, "Interpolation-sequence based model checking," in *Proc. Formal Methods Computer-Aided Design*. USA: Springer, Nov. 2009, pp. 1–8.

[24] G. Cabodi, S. Nocco, and S. Quer, "Interpolation sequences revisited," in *Proc. Design, Autom. Test Eur.*, Mar. 2011, pp. 1–6.

[25] M. L. Case, H. Mony, J. Baumgartner, and R. Kanzelman, "Enhanced verification by temporal decomposition," in *Proc. Formal Methods Computer-Aided Design*, Nov. 2009, pp. 17–24.

[26] J. Baumgartner, T. Heyman, V. Singhal, and A. Aziz, "An abstraction algorithm for the verification of level-sensitive latch-based netlists," *Formal Methods Syst. Design*, vol. 23, no. 1, pp. 39–65, Jul. 2003, doi: 10.1023/a:1024485130001.

[27] P. Bjesse and J. Kukula, "Automatic generalized phase abstraction for formal verification," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, Feb. 2005, pp. 1076–1082.

[28] M. Sheeran, S. Singh, and G. Stålmarck, "Checking safety properties using induction and a sat-solver," in *Formal Methods Computer-Aided Design*. Berlin, Germany: Springer, 2000, pp. 127–144.

[29] M. L. Case, A. Mishchenko, and R. K. Brayton, "Inductively finding a reachable state space over-approximation," in *Proc. Int. Workshop Log. Synth.*, May 2006, pp. 1–18.

[30] C.-J.-H. Seger and R. E. Bryant, "Formal verification by symbolic evaluation of partially-ordered trajectories," *Formal Methods Syst. Design*, vol. 6, no. 2, pp. 147–189, Mar. 1995, doi: 10.1007/bf01383966.

[31] G. Cabodi, S. Nocco, and S. Quer, "Benchmarking a model checker for algorithmic improvements and tuning for performance," *Formal Methods Syst. Design*, vol. 39, no. 2, pp. 205–227, Oct. 2011, doi: 10.1007/s10703-011-0123-3.

[32] G. Cabodi, P. E. Camurati, M. Palena, and P. Pasini. *PDtools*. Accessed: Oct. 29, 2025. [Online]. Available: https://github.com/polito-fmgroup/pdtools

[33] A. Biere, K. Heljanko, and S. Wieringa, "AIGER 1.9 and beyond," Inst. Formal Models Verification, Johannes Kepler University, Linz, Austria, Tech. Rep. 11/2, 4040.

[34] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Proc. 22nd Int. Conf. Comput. Aided Verification*, ser. CAV'10Berlin, Heidelberg: Springer-Verlag, 2010, pp. 24–40, doi: 10.1007/978-3-642-14295-6_5.

[35] G. Cabodi, P. E. Camurati, M. Palena, and P. Pasini. *PDT-INV-ITP-25*. [Online]. Available: https://github.com/polito-fmgroup/invitp25

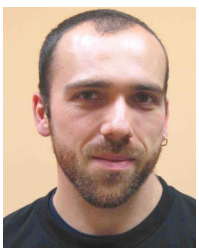
[36] G. Cabodi, P. E. Camurati, A. Mishchenko, M. Palena, and P. Pasini, "SAT solver management strategies in IC3: An experimental approach," *Formal Methods Syst. Design*, vol. 50, no. 1, pp. 39–74, Mar. 2017, doi: 10.1007/s10703-017-0272-0.

[37] N. Froleyks, E. Yu, M. Preiner, A. Biere, and K. Heljanko, "Introducing certificates to the hardware model checking competition," in *Proc. 37th Int. Conf. Comput. Aided Verification*, Zagreb, Croatia. Cham, Switzerland: Springer, 2025, pp. 281–295.

GIANPIERO CABODI received the degree in electronic engineering, in 1984, and the Ph.D. degree in computer and system engineering, in 1988. He is currently an Associate Professor with the Dipartimento di Automatica e Informatica, Politecnico di Torino, Italy. His current research interests include formal methods and verification, model checking engines, and cybersecurity and classification.

PAOLO ENRICO CAMURATI received the degree in electronic engineering, in 1984, and the Ph.D. degree in computer and system engineering, in 1988. He is currently a Full Professor with the Dipartimento di Automatica e Informatica, Politecnico di Torino, Italy. His current research interests include formal verification of hardware correctness and verification problems in cybersecurity.

MARCO PALENA received the M.S. degree in computer engineering and the Ph.D. degree in computer and control engineering from the Politecnico di Torino, in 2012 and 2017, respectively. He is currently a Postdoctoral Researcher with the National Inter-University Consortium for Telecommunications (CNIT). His research interests include SAT-solving, automated reasoning, and formal methods.



PAOLO PASINI received the M.S. degree in computer engineering and the Ph.D. degree in computer and control engineering from the Politecnico di Torino, in 2012 and 2017, respectively. He is currently a Postdoctoral Researcher with the Department of Electronics and Telecommunications, Politecnico di Torino. He is active in the field of formal verification, with a specific focus on hardware model checking.