Improving Bounded Model Checking exploiting Interpolation-based learning and strengthening

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 19 July 2024, accepted 10 August 2024, date of publication 21 August 2024, date of current version 4 September 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3446802

RESEARCH ARTICLE

Improving Bounded Model Checking Exploiting Interpolation-Based Learning and Strengthening

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

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

Corresponding author: Gianpiero Cabodi (gianpiero.cabodi@polito.it)

This work was supported in part by Partenariato Esteso–RESTART ''RESearch and innovation on future Telecommunications systems and networks, to make Italy more smART'' under Grant PE00000001.

ABSTRACT Bounded Model Checking (BMC) is one of the most prominent approaches used as a falsification engine, capable of identifying counterexamples of bounded length, in a scalable and sustainable way. Nevertheless, in the context of a portfolio-based verification suite, BMC can benefit from potential interaction with other engines, exploiting their capabilities and partial results as a form of application-dependant learning. In the past, previous works tackled the issue of using over-approximated state sets generated via Binary Decision Diagrams (BDD) based traversals. In a sense, BDD engines can be considered as external tools, whereas interpolants are directly related to BMC problems. Since interpolants come from Boolean satisfiability (SAT) refutation proofs, their role as a SAT-based learning can be potentially higher. Furthermore, their integration is more tightly linked to the BMC problem at hand. In this paper we aim at improving the efficiency of SAT calls in BMC problems, exploiting interpolation-based invariants computed over different cut points, as additional constraints for BMC problems. We experimentally evaluate costs and benefits of our proposed approach on a set of publicly available model checking problems.

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

I. INTRODUCTION

Ranging from personal and commodity devices up to business and safety critical environments, digital systems have become ubiquitous into our daily lives. Furthermore, thanks to the advent of high-level synthesis, contemporary hardware designs comprise an extremely high level of complexity, and thus design verification has become one of the most relevant aspects of the design and production flow. In such a scenario, assessing and guaranteeing the correct behavior of digital systems, with respect to their specification, is becoming an ever more pressing need for manufacturers [1].

Traditionally, hardware designs are verified through *simulation*, in which a model of the system is solicited with a set of stimuli and the resulting behavior is checked against a golden reference. Simulation is scalable and applicable to designs of virtually any size, but at the same time it is also costly and incomplete. It is often unfeasible to completely cover the behaviors of a realistic design through simulation. Since the late '80s, *formal hardware verification* has become an increasingly more relevant approach to deal with the coverage limitations of simulation. Formal verification is the use of the formal methods of mathematics and logic to (dis)prove the correctness of a design considering a formal specification of its expected behavior. Currently, model checking is one of the most widely used formal verification approaches when taking into account sequential hardware designs. Given a *model* of the design and a representation of its specification, via algorithmic procedures it is possible to check whether the model meets the formal specification. Such procedures exhaustively traverse the modeled behaviors of the system to either confirm theirs adherence to specifications or to

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

generate a trace demonstrating a violating behavior, called *counterexample*. Designs are usually modeled as *transition systems*, comprising states and transitions between states, and their specification is formalized using *temporal logic properties*. Invariant verification is a model checking task in which system specifications are described as invariant properties that must hold true in every reachable state of the models [2].

Bounded Model Checking (BMC) [3] is a state-of-the-art formal verification method with widespread industry-level application in various domains. Such a technique has been established as a relevant falsification engine, in order to identify bugs and counterexamples up to a given unroll bound k. Despite it being an incomplete approach, as it produces bounded proofs, it plays an important role in a verification context due to its simple and scalable SAT-based approach. Furthermore, the bounded proofs BMC produces can be significant in scenarios where it is impossible to fully verify the design under test.

Complete SAT-based verification approaches based on BMC have been studied in order to address the completeness issue: inductive techniques [4] and Craig interpolation based model checking [5] are among the most notable and successful ones. More recently, a novel algorithm called IC3 [6] has emerged as an approach unrelated to BMC, as it does not require transition relation unfoldings. Variants of standard interpolation-based techniques have been introduced as well [7].

Previous research addressed the issue of using over-approximated state sets generated by BDD-based [8] traversals. This paper focuses on Craig interpolation as an operator able to provide a form of learning, derived from a BMC-like problem, to be exploited in order to speed-up subsequent BMC queries. Our target is to accelerate BMC runs with the help of interpolation-based learning.

In the context of Model Checking, Craig interpolation can be interpreted as an operator capable of computing over-approximated images of reachable state sets. Using interpolation in such a scenario, it can be viewed as a form of iterative refinement of proof-based abstractions whose purpose is of narrowing down proofs to just the relevant facts. In order to derive the aforementioned over-approximations of the reachable states, they can be computed taking into account refutation proofs arising from unsatisfied BMC formulas.

A. RELATED WORKS

We build upon works taking into account advances in both BMC- and interpolation-based settings. Given the context of application, our approach is directly related to various recent works on SAT-based Model Checking, such as those presented in [9] and [10]. More in details, in [9] the authors propose a method to improve standard BMC using BDDs representing over-approximations of reachable states. The overestimated reachable state sets are used to restrict the search space of BMC runs. Such BDDs can thus be seen as explicit constraints for the SAT solver. In [10] the authors propose various approaches to strengthen and/or weaken interpolants, in order to exploit their representativeness while maintaining their size under control.

Although this paper focuses explicitly on bit-level SAT-based hardware model checking, BMC has seen a growing application in other branches of formal verification, such as software model checking. In such a context, recent works aimed at improving efficiency and applicability of BMC have arisen, such as [11], that addresses scalability through underapproximations widening driven by proofs of unsatisfiability, and [12] that exploits partial proofs to reuse within BMC runs.

B. CONTRIBUTION

In this paper we introduce a technique to improve the efficiency of SAT calls for BMC problems. The proposed technique is based on exploiting interpolation-based invariants, as additional constraints. These invariants are computed over different cut points alongside a given unrolling and injected as a constraining factor at various bounds within the BMC problem itself. Our work is focused on trying to understand whether and how much interpolants can be exploited to speed up BMC checks, being used to represent constraints on forward and backward reachable states at given unrolling boundaries.

In this paper we propose the following contributions:

1) A technique to generate redundant constraints derived from:
   - BMC problems, taking into account just the *easy*, i.e., quick to solve, bounds;
   - Interpolation-based runs;
2) A technique to reuse and exploit generated constraints in order to improve BMC checks performance for *hard* bounds;
3) An experimental evaluation of the novel approaches here presented.

C. OUTLINE

The paper is organized as follows. Section II introduces the notation used throughout the paper as well as some preliminary concepts on SAT-based techniques for verification, reachability analysis, and interpolation. Section III introduces our novel approach at exploiting interpolation-based learning as a way to speed-up BMC checks. Section IV presents the experimental evaluation of the proposed techniques over state-of-the-art hardware model checking benchmarks. Section V draws conclusion concerning the proposed techniques, provides some insight and outlines possible future works.

II. PRELIMINARIES

A. MODEL, NOTATION AND PROPERTY DEFINITION

We take into account systems modelled by labelled state transition structures, implicitly represented using Boolean formulas.

Definition 1: A *transition system* $\mathcal{S}$ is a triple $\langle X, \mathcal{I}, T \rangle$, where X is a set of Boolean variables which represent the system states, $\mathcal{I}$ is a Boolean formula, defined over X, which represents the set of *initial states* and T is a Boolean formula, defined over $X \times X'$, which represents the system *transition relation*.

Variables of X take the role of *state variables* for $\mathcal{S}$. A state of $\mathcal{S}$ can be thus represented taking into account a complete truth assignment s for its state variables. Sets of system states can be represented using Boolean formulas over X. With Space($\mathcal{S}$) we denote the state space of $\mathcal{S}$. Given both a Boolean formula F over X and s, a complete truth assignment, such that $s \models F$, we have that s is a state within the set represented by F and can be in turn be called an *F-state*. Future states of $\mathcal{S}$, i.e., states reached after a transition, are represented using primed state variables X'. Accordingly, sets of future states are then represented using Boolean formulas over X'. With the notation s, s' we identify a complete truth assignment to $X \times X'$ derived by joining a complete truth assignment s to X with a complete truth assignment s' to X'.

In our context, transition systems are used to represent the behaviour and model sequential hardware circuits. In such a scenario, every state variable $x_i \in X$ corresponds to a latch. The set I, i.e., the initial states of the system, is defined by reset values of latches. Transition relation T is the conjunction $\bigwedge_i (x_i' \leftrightarrow \exists PI . \delta_i(X, PI) = \top)$ where $\delta_i(X, PI)$ are formulas representing the next-state function of each latch. Note that, in the modelled transition system, the primary inputs PI of the circuit are abstracted away.

Definition 2: A *literal* is a Boolean variable or its negation. A *clause* (*cube*) is a possibly empty disjunction (conjunction) of literals. A Boolean formula is in *Conjunctive Normal Form* (CNF) or clausal normal form if it is a conjunction of one or more clauses.

Definition 3: Given a Boolean formula F, a truth assignment for F defined over variables X is a function $\tau : Y \subseteq X \rightarrow \{\top, \bot\}$ mappings Y variables to truth values. τ can either be *complete*, iff $Y \equiv X$, or *partial* otherwise.

Definition 4: Given a truth assignment τ and a literal x, $\tau satisfies x$, expressed as $\tau \models x$, iff $\tau(x) = \top$. Conversely, τ satisfies $\neg x$ iff $\tau(x) = \bot$. A clause C is satisfied by a truth assignment τ, written as $\tau \models C$, iff at least one literal in C is satisfied by τ. Given a truth assignment τ and a CNF formula F, τ satisfies F, written as $\tau \models F$, iff all clauses in F are satisfied by τ.

Definition 5: Given a Boolean formula F, F is said to be *satisfiable* iff a truth assignment τ such that $\tau \models F$ exists. F is said to be *unsatisfiable* otherwise. Given two Boolean formulas F and G, if they are either both satisfiable or both unsatisfiable, they are said to be *equi-satisfiable*.

Abusing notation, a truth assignment can be represented in the form of a set of literals of different variables. A truth assignment τ expressed in this way assigns each variable to the appropriate truth value satisfying the corresponding literal in the set. We also represent a clause, or a cube, as a set of literals, leaving the disjunction, or conjunction, implicit when unambiguous given the context.

Given a Boolean formula F, whenever it is necessary to explicitly indicate its *before/after* version with respect to an evaluation, such a refinement step for instance, in order to express its *before* version we use a -1 superscript, i.e., F^{-1}. We will conventionally use letters in boldface for arrays of functions, such as $\boldsymbol{F} = (F_0, F_1, \ldots)$.

Clauses and their representations are quite relevant, since most modern SAT solvers [13], [14] use them as their principal representation and manipulation formalism to express Boolean functions.

Definition 6: Given a transition system $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$, and a complete truth assignment s, s' to $X \times X'$, if we have that $s, s' \models T$ then s is a *predecessor* of s' and s' is a *successor* of s. A states sequence $\pi^{0,n} = (s_0, \ldots, s_n)$ is a *path* in $\mathcal{S}$ iff $s_i, s'_{i+1} \models T$ for every pair of adjacent states in the sequence (s_i, s_{i+1}), $0 \leq i < n$.

Definition 7: A state $s \in$ Space($\mathcal{S}$) is said to be *reachable exactly in k steps* in $\mathcal{S}$ iff there exists a finite initial path of length k $\pi = (s_0, \ldots, s_k)$ such that $s_k = s$.

Definition 8: A state $s \in$ Space($\mathcal{S}$) is *reachable within k steps* or, alternatively said, *reachable bounded by k* in $\mathcal{S}$ iff there exists $i \leq k$ such that s is reachable exactly in i steps in $\mathcal{S}$.

Definition 9: A state $s \in$ Space($\mathcal{S}$) is *reachable* in $\mathcal{S}$ if it can be reached within a arbitrary, but finite, number of steps in $\mathcal{S}$.

Definition 10: $\mathcal{R}_i^E(\mathcal{S})$ denotes the *set of states reachable in exactly i steps* in $\mathcal{S}$.

Definition 11: $\mathcal{R}_i(\mathcal{S})$ denotes the *set of states reachable within i steps* in $\mathcal{S}$, i.e.,

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

Definition 12: The *reachability diameter* of $\mathcal{S}$ is the minimal number $d \in \mathbb{N}$ of steps needed to reach all reachable states in $\mathcal{S}$:

$$d \stackrel{\text{def}}{=} \arg\min_{i \in \mathbb{N}} \{ i \mid \mathcal{R}_i(\mathcal{S}) = \mathcal{R}_{i+1}(\mathcal{S}) \}$$

Definition 13: $\mathcal{R}(\mathcal{S})$ denotes the *set of states reachable* in $\mathcal{S}$, i.e.,

$$\mathcal{R}(\mathcal{S}) \stackrel{\text{def}}{=} \bigcup_{0 \leq j < d} \mathcal{R}_j(\mathcal{S})$$

We use a superscript notation whenever there is more than one timeframe involved in a formula, for instance in the case of circuit unrollings. We use X^i for the variables instantiated at the i-th time frame. When support variables can be easily inferred from the context, they are omitted for the sake of simplicity and compactness.

Definition 14: A *path formula* of length $k = j - i$ from two timeframes, i to j, is the propositional formula $\Pi(i, j)$

over $X^i \cup \cdots \cup X^j$:

$$\Pi(i,j) \stackrel{\text{def}}{=} \bigwedge_{h=i}^{j-1} T(X^h, X^{h+1})$$

Definition 15: An *initial path formula* of length k is a propositional formula:

$$\Pi_0(k) \stackrel{\text{def}}{=} \mathcal{I}(X^0) \wedge \Pi(0,k)$$

Given $\Pi(i,j)$, such a path formula encodes all paths of length $k = j - i$ that start at timeframe i in $\mathcal{S}$. On the other hand, an initial path formula $\Pi_0(k)$ represents all paths of length k that start from the initial states of the system. An initial path formula $\Pi_0(k)$ can thus be used to represent the set of reachable states in exactly k steps, starting from the initial states in $\mathcal{S}$.

Definition 16: Given a transition system $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$, 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 17: Given a transition system $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ and given an invariant property P defined over X, a propositional formula F over X is *safe* with respect to P iff F is stronger than P, i.e., $F \rightarrow P$.

Definition 18: A *bad cone* of length $k = j - i$ from timeframes i to j is the propositional formula $Cone(i,j)$ over $X^i \cup \cdots \cup X^j$:

$$Cone(i,j) \stackrel{\text{def}}{=} \Pi(i,j) \wedge \bigvee_{h=i}^{j} \neg P(X^h)$$

A bad cone $Cone(i,j)$ encodes all paths starting at timeframe i reaching the target in up to $k = j - i$ steps. A bad cone thus represents the set of backward reachable, in at most k steps, bad states from the target.

Definition 19: Given a transition system $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ and given a propositional formula F over X, F is an *inductive invariant* of $\mathcal{S}$ if the following conditions hold:

$$\mathcal{I} \rightarrow F \qquad (Initiation)$$
$$F \wedge T \rightarrow F' \qquad (Consecution)$$

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

Definition 20: Given a transition system $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ and given an invariant property P over X, an inductive invariant F of $\mathcal{S}$ is an *inductive strengthening* of P iff it is safe with respect to P.

B. BOUNDED MODEL CHECKING

Given a transition system $\mathcal{S} \stackrel{\text{def}}{=} \langle X, \mathcal{I}, T \rangle$ and an invariant property P, Bounded Model Checking [15] is an iterative process aimed at checking whether a counterexample to P of length at most k in $\mathcal{S}$ exists, or at proving its absence. In order to do so, BMC just needs to performs a SAT check on BMC formulas.

Definition 21: A *BMC formula* of length k for P in $\mathcal{S}$ is the propositional formula $bmc(k)$ over $X^0 \cup \cdots \cup X^k$:

$$bmc(k) \stackrel{\text{def}}{=} \Pi_0(k) \wedge \bigvee_{i=0}^{k} \neg P(X^i) = \mathcal{I} \wedge Cone(0,k)$$

A BMC formula of length k can be intuitively seen as representing all initial paths, of length at most k, in $\mathcal{S}$ that reach a bad state in $\neg P$. If the formula is SAT, a counterexample to P of length at most k in $\mathcal{S}$ can be found. On the contrary, no such a counterexample exists.

BMC-based tools iteratively solve BMC formulas at increasingly deeper bounds, until either a counterexample is found or an halting condition is met, e.g., some maximum bound is reached. BMC is suitable for finding counterexamples, but on the other hand it is not capable of detecting whether P holds in $\mathcal{S}$. In order to face such a limitation, ad-hoc approaches are necessary to support Unbounded Model Checking (UMC). The capability of evaluating reachability fix-points and finding inductive invariants is thus the main difference between BMC and UMC.

C. CRAIG INTERPOLANTS

Craig's interpolation theorem is a very influential result in mathematical logic concerning the relationship between model and proof theory. Craig [16] provided its original formulation of the theorem in the context of first-order logic. Additional variants of the original theorem valid for other logical systems, such as propositional logic, have been formulated as well. We provide here the formulation of the theorem in propositional logic, since it is the one most relevant in our scenario and the one which is typically encountered in model checking contexts.

Theorem 1: Given two propositional formulas A and B, if they are inconsistent, i.e., if $A \wedge B$ is unsatisfiable, then there is a propositional formula I, which can be called *interpolant* between A and B, such that:

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

Intuitively, I represents an abstraction of A from the standpoint of B. The interpolant I summarizes and translates, in the common language between A and B, the motive behind A and B mutual inconsistency. In the following, we denote with $I = \text{ITP}(A,B)$ the procedure capable of computing a Craig's interpolant starting from a pair of inconsistent formulas A and B.

Interpolants can be directly inferred from refutation proofs of unsatisfiable SAT solving runs. Taking into account an unsatisfiable formula in the form $A \wedge B$, most modern SAT solvers can generate a refutation proof in either resolution-based or clausal form. In the former case, i.e., given a refutation proof, an interpolant formula $I = \text{ITP}(A,B)$

can be derived in polynomial time and space with respect to the size of the proof itself, assuming the form of an AND/OR combinational circuit. Specifically to the context of model checking, if we interpret A as representing a set of reachable states and B as representing a set of bad states, then $I = \text{ITP}(A, B)$ can be seen as a safe overapproximation of A with respect to B. As a consequence, overapproximations generated this way can be used to detect reachability fix-points. McMillan [5] introduced the first complete algorithm for symbolic model checking exploiting Craig's interpolation. Such an algorithm, called *standard interpolation* or ITP for short, computes Craig's interpolants, starting from refutation proofs of unsatisfiable BMC runs, to overapproximate reachable states of the system.

III. COMBINING SAT-BASED BMC AND INTERPOLATION-BASED LEARNING

In this section we describe the novel approach we propose. As it stands, it can be viewed as a way to partition a verification task between an interpolation-based and a SAT-based engine. Alternatively, it can be seen as an optimization of a BMC run, by means of redundant information learned in the form of interpolants.

Starting from an UNSATBMC run of length k with a given bound, the proposed approach identifies two cut points k' and k'' in order to split the BMC formula in three parts: a prefix, identified as an initial path formula $\Pi_0(k')$ up to bound k', a transition relation unrolling, identified as a path formula $\Pi(k', k'')$ from bound k' up to bound k'', and a backward cone $Cone(k'', k)$ from the target, i.e.:

$$bmc(k) = \Pi_0(k') \wedge \Pi(k', k'') \wedge Cone(k'', k)$$

For instance, let's assume the procedure starts with an initial unrolling of depth $k = 40$, choosing $k' = 2$ and $k'' = 26$, we would have a backward cone of length 14 and a transition relation unrolling of length equal to $l = k'' - k' = 24$.

The selection of the k' and k'' cut points directly impacts the way in which the procedure implicitly uses them: such a choice may help the solver propagate activity from the initial states forward, from the target states backwards or inbetween cuts, in intermediate frames.

Given the two cut points, the procedure tries to identify a proper abstraction for the transition relation unrolling in between k' and k'', performing an interpolation step $I = \text{ITP}(A, B)$ with $A = \Pi(k', k'')$ and $B = \Pi_0(k') \wedge Cone(k'', k)$. The result is an interpolant defined over the state variables of the two cut points, which represents an over-approximation of the transition relation unrolling of a given depth l, that it is guaranteed not to intersect B. Note that such an interpolant can be used as a redundant constraint for any transition relation unrolling of length l, modulo variables relabelling. We denote this interpolant as an *abstract transition unrolling*.

The resulting interpolant only depends on the cut points variables and, intuitively, represent an existential quantification for the variables of the cuts themselves.

Fig. 1 shows a visual representation of how the BMC formula is partitioned, according to the description provided.

Interpolant generated as described above can be then used as a form of redundant learning in the context of BMC problems. In our current setup, the interpolant is mostly used as-is, without further processing. At the cost of additional computation, however, these interpolants can be optimized using techniques such as [10] and [17] in order to reduce their overhead.

Abstract transition unrolling can be conjoined to the formula needed to solve a BMC problem at a specific bound, with the specific aim of helping the SAT solver to reach a decision faster. The main idea lies in the notion of providing extra information with the purpose of limiting the activity of the SAT solver, in terms of decision-propagation-conflict cycles, as well as a form of external constraining for the addressable state space and its size.

Usually, BMC runs use an *exact bound* strategy, in which only a specific bound k is checked for counterexamples to the property, and the verification procedure is run incrementally for all the bounds of interests. Our approach can be applied to BMC runs that uses either an *exact bound* or an *up to bound* approach. The only relevant aspect to be taken into account is the fact that the backward cone must not propagate beyond the k'' cut point in the *up to bound* case, otherwise it would overlap with the transition relation unrolling previously mentioned. In our current setup, we rely on an *exact bound* approach to keep the computation lightweight.

Since computing these interpolants involves some overhead, it is important to evaluate if the performance gain outweighs the added overhead. In order to investigate this performance trade-off, we consider interpolant-enhanced runs that can be customized with two additional parameters, namely an *init step* and a *period*. Such parameters describe the starting bound from which and how often to inject the additional constraint in the SAT solver, respectively. The former allows us to skip the shallowest part of the BMC problem, thus avoiding encumbering easier-to-solve bounds for which little performance gains could be attained. The latter is responsible for leveraging the benefits of the additional constraints while still allowing us to keep the introduced overhead under control. Sizing both parameters strictly depends on the actual structure and behaviour of the underlying circuit, thus their values have to be carefully evaluated taking into account the problem we want to solve.

The main contribution of the proposed technique is being capable of abstracting a portion of the transition relation, with the aim of helping problems in which the SAT solver has to (potentially) analyze several branching conditions, especially if bound to choices depending on primary inputs, and subsequently may incur in a significant number of backtracking steps. The proposed technique implicitly acts as a form of primary input existential quantification, thus limiting the number of traces that may lead to a specific state and/or subsuming multiple behaviours at once.

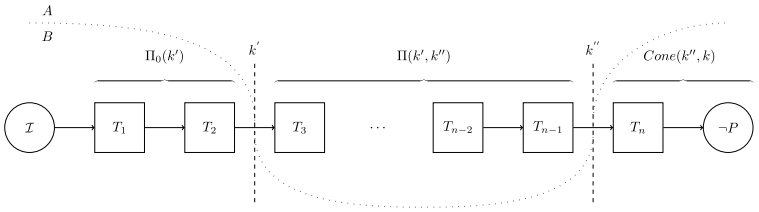


FIGURE 1. Standard combinational unrolling for BMC with two sample cut points, k' and k'', and boundaries between A and B.

As mentioned before, the selection of proper cut points is at the heart of the proposed technique. It is desirable for the abstracted part to encompass a significant portion of the sub-behaviours of the design under verification, thus the two cut points cannot be too close together. At the same time, too wide of a gap between the two cut points may result in a large formula to be injected into the SAT solver, with significant overhead to account for. One aspect to take into account, is that it is not just a matter of numerical distance between the two cut points, since different designs will manifest different behaviours. For very complex designs, smaller gaps will still encompass a relevant amount of behaviours. For easier design, a larger gap may be beneficial, in order to include a big enough abstraction for it to be useful, i.e., being representative of a significant part of the underlying behaviours. Without a priori knowledge about the design, one can estimate the number of bounds that may be suitable to include in the abstraction taking into account the time needed to solve single bounds, thus opting to either include many easier-to-solve or just fewer harder-to-solve ones.

Concerning the placement of the first cut point, i.e., the one closest to the initial states, it should be placed after a possible initial transient behaviour for the circuit at hand, in order to skip, for instance, setup behaviours. The desired effect is to keep only relevant behaviours in the abstracted part, once again not to incur in unnecessary overhead. Intuitively, if the additional learning injected in the SAT solvers concerns a behaviour which is relevant only to the initial setup of the circuit, but it is not related to the actual behaviour of the design under verification, during its operation, such an abstraction would be of little help for the following BMC problems.

All in all, cut points selection is a matter of balancing the extent of the behaviours included, while keeping initial transients out, and limiting the size of the corresponding interpolant.

Algorithm 1 reports a modified BMC procedure up to a certain bound k_{max}, capable of exploiting external learning in the form of an abstract transition relation. First the algorithm checks whether the property P is satisfied by the initial states (lines 2–3). If an initial state fails to satisfy P, the procedure terminates by returning FAIL alongside a trivial counterexample. Otherwise, the procedure starts iterating over increasing values of k up to k_{max}. The actual BMC problem taken into account differs according to whether the procedure exploits or not the abstract transition relation. In case the condition (line 6) holds, i.e., iteration is past the initial step for which we are willing to use the abstraction and the current bound respects the injection period, the BMC formula (Definition 21) is slightly altered to inject also the abstract transition relation, denoted as $T^*_{k',k''}$. At each iteration, the procedure checks whether there is at least a path satisfying the BMC formula up to bound k (lines 10–11). If a satisfying path $\pi^{0,k}$ is found, the procedure returns FAIL along with the path as a counterexample. If the procedure does not find any counterexample of length up to k_{max}, it returns a SUCESS, indicating that P holds up to the given bound.

Algorithm 1 Modified BMC Traversal

Input: $\mathcal{S} = \langle X, \mathcal{I}, T\rangle$ a transition system; P a property over X; k_{max} maximum bound; $T^*_{k',k''}$ abstract transition relation between cut points k' and k''; i initial injection bound; p period of injection.

Output: $\langle res, cex\rangle$ with $res \in$ {SUCESS, FAIL}; cex a (possibly empty) initial path representing a counterexample.

1: **procedure** BoundedModelChecking($\mathcal{S}$, P, k_{max}, $T^*_{k',k''}$, $init\_step$, $period$)
2: **if** $\exists s_0 \models \mathcal{I}(X) \wedge \neg P(X)$ **then**
3: **return** $\langle$FAIL, $(s_0)\rangle$
4: $k \leftarrow 1$
5: **while** $k \leq k_{max}$ **do**
6: **if** $k \geq i$ **and** $(k-i) \bmod p = 0$ **then**
7: $bmc(k) \leftarrow \Pi_0(k) \wedge T^*_{k',k''} \wedge \bigvee_{i=0}^{k} \neg P(X^i)$
8: **else**
9: $bmc(k) \leftarrow \Pi_0(k) \wedge \bigvee_{i=0}^{k} \neg P(X^i)$
10: **if** $\exists \pi^{0,k} \models bmc(k)$ **then**
11: **return** $\langle$FAIL, $\pi^{0,k}\rangle$
12: $k \leftarrow k+1$
13: **return** $\langle$SUCESS, $-\rangle$

We leave as future work to investigate how to optimize abstract transition unrolling as well strategies to devise proper heuristics to better support parameters selections (i.e., cut points, depth, initialization and period of injection). Furthermore, we are interested in taking into account both single an multiple properties designs [18], to better characterize the role of learning from related, but different, subset of properties.

IV. EXPERIMENTAL RESULTS

In order to evaluate our proposed approach, we implemented the described methodology within the PdTRAV tool [19], a state-of-the-art model checking academic tool. Experiments were run 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 the experiment were run taking into account a time limit of 3600 seconds and a memory limit of 16 GB.

The experimental data here presented provide an evaluation of the proposed approach, compared to standard techniques that we consider as our baseline.

The benchmarks set considered was derived from a subset of past Hardware Model Checking Competitions (HWMCC) suites [20], with a primary focus on instances belonging to the deep bound track, considered hard-to-solve, and difficult SAT instances from the other tracks. The set includes hardware as well as software verification problems. We focused on problems characterized by increasingly expensive BMC bounds. We thus discarded problems involving a single hard-to-solve bound following a sequence of very easy-to-check bounds, since they are of rather limited interest in relation with our proposed approach. We also discarded very deep benchmarks characterized by just long series of easy-to-solve bounds.

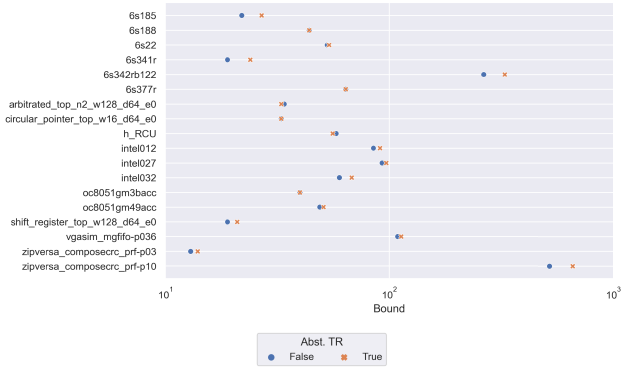


FIGURE 2. Bounds distribution of baseline BMC runs with respect to ones using abstract transition unrollings.

Fig. 2 compares the bound reached taking into account baseline BMC runs and runs exploiting the injection of pre-computed abstract transition unrollings. All the abstract transition unrollings have been pre-computed offline with a cutoff time of around 100 seconds. Overall we can notice that, upon properly selecting bounds for the abstractions and adopting a suitable injection period, we can support the SAT solver activity by providing redundant information which in turn reduces the decision effort required at each bound, as well as providing a form of space constraining, which limit the state space the procedure has to explore while addressing each bound.

Fig. 3 compares, in percentage, the activity of the underlying SAT solvers taking into account number of clauses, variables, decisions, propagations and conflicts. The ratio

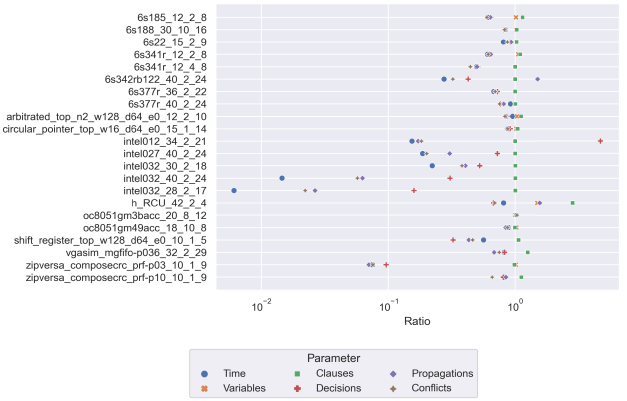


FIGURE 3. Ratio distribution of SAT solver activities for runs using abstract transition unrollings with respect to baseline.

is computed evaluating each parameter obtained from runs exploiting the injection of pre-computed abstract transition unrollings against the baseline. We can see that the ratio for all the parameters tends to lie below the unit threshold (i.e., same or very close value for both baseline and enhanced runs). Even for instances where there is more effort required from the SAT solver standpoint, not all parameters are impacted negatively at the same time, as both improvements and degradations are observed.

Table 1 summarizes data plotted in Fig. 2 and Fig. 3 to provide a better understanding of actual data distributions. For each parameter we provide aggregate data, namely average, median, standard deviation, minimum and maximum values, taking into account both the baseline, i.e., no use of abstract transition relation, and runs exploiting it. Concerning data relative to Fig. 3, the table presents actual values for the two series rather than the ratio, to give a hint about the scales of the values at hand.

In order to highlight the impact of using external informations to speed-up baseline BMC runs, we took into account a specific benchmark, namely `intel032` to compare more explicitly the time requirement needed to reach a specific bound in various configurations, and the underlying SAT solver effort. Fig. 4 compares the impact of different abstract transition unrollings on the candidate benchmark, both in terms of time required to reach a given bound and in terms of the parameters characterizing SAT solver activity at a given bound. From the plots we can see that the curves present the same overall patterns for all runs, consisting of have easier bounds, the plateaus, interleaved with harder to solve ones. All the enhanced runs manage to reach deeper bounds and tend to lie beneath the baseline, i.e., they take less time to reach the same bound.

In order to better highlight the difference among runs, the y axis is charted in logarithmic scale. It is possible to notice how there might be up to an order of magnitude, if not more, in term of timing difference among runs. For all charts, the first data sample plotted along the x axis is the first instance for which the relative parameter presents a non-zero value.

TABLE 1. Values distributions and summaries for Fig. 2 and Fig. 3 data.

Parameter	Abst. TR	Avg	Mdn	St.Dev	Min	Max
Bound	False	87.67	51	121.73	13	519
	True	100.94	52.5	156.04	14	659
Time	False	2588.11	2723.87	830.21	624.37	3535.21
	True	1675.74	1852.11	1148.62	10.54	3586.97
Clauses	False	1.58E+06	5.25E+05	2.69E+06	3.21E+04	1.09E+07
	True	1.50E+06	5.41E+05	2.44E+06	3.17E+04	1.09E+07
Propagations	False	7.82E+09	7.96E+09	4.91E+09	1.47E+09	1.88E+10
	True	5.41E+09	5.54E+09	4.02E+09	3.94E+07	1.38E+10
Variables	False	9.32E+05	1.10E+05	1.88E+06	4.21E+03	7.32E+06
	True	8.60E+05	1.64E+05	1.71E+06	4.26E+03	7.32E+06
Decisions	False	3.80E+07	1.24E+07	8.47E+07	1.13E+06	3.41E+08
	True	1.79E+07	8.11E+06	3.51E+07	4.34E+05	1.38E+08
Conflicts	False	7.03E+06	6.42E+06	5.42E+06	3.15E+05	1.91E+07
	True	4.21E+06	2.93E+06	4.08E+06	2.89E+04	1.32E+07

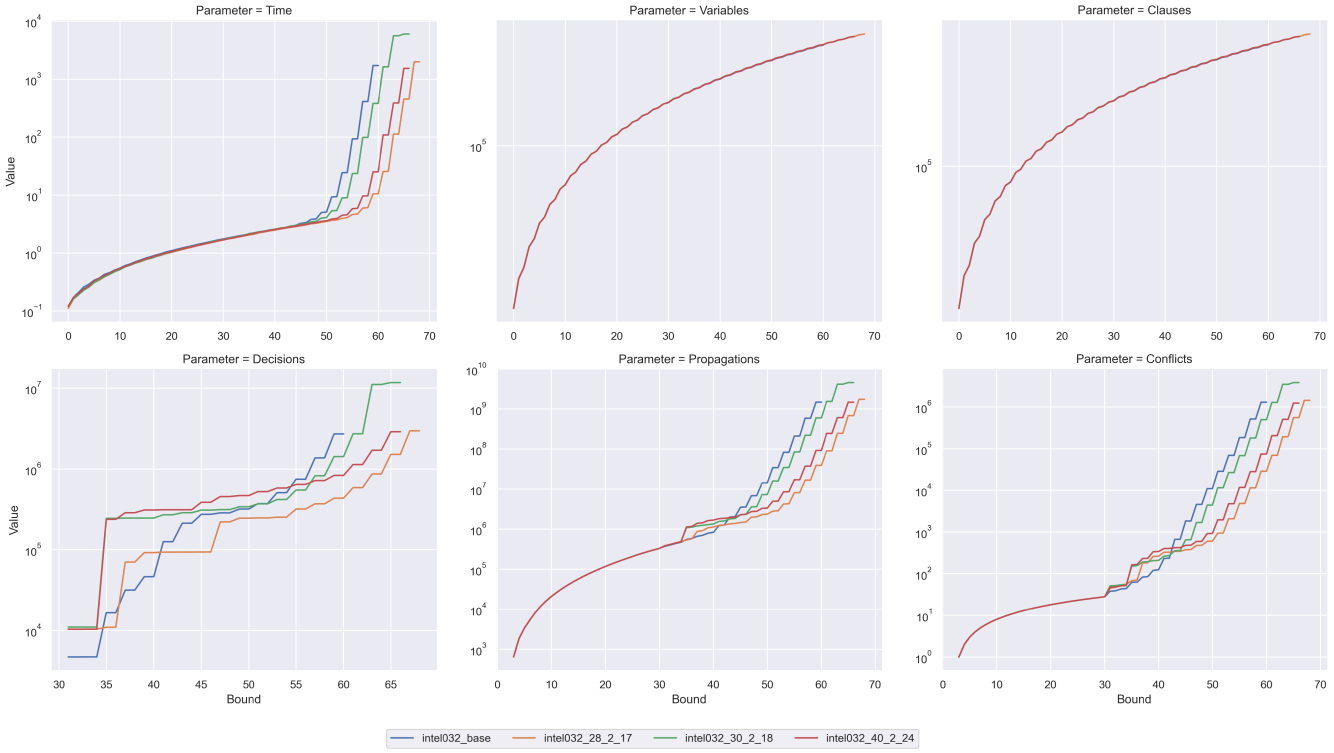


FIGURE 4. Comparison of time required to reach different bounds and respective SAT solver activity over the same candidate benchmark `intel032` with different run setups.

V. CONCLUSION

In this paper we addressed the problem of understanding whether and how interpolants could be used to speed up BMC checks, exploiting them as a form of redundant learning, representing constraints on forward and backward reachable states at given unrolling boundaries. We proposed a technique exploiting interpolation-based invariants, computed over different cut points alongside a given transition unrolling, that can be injected as an external constraining factor within a BMC problem. We experimentally evaluated costs, benefits, as well as invariant selection options, on a set of publicly available model checking problems. Experimental results support our take that external constraints can be beneficial in order to speed up BMC runs. Future works will investigate improvements on appropriate cut and application points selections, since they are strictly related to the benchmark and/or verification problem at hand, and directly impact the proposed approach effectiveness.

REFERENCES

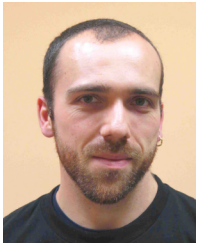
[1] H. Foster, "2022 Wilson research group FPGA functional verification trends," Siemens Digit. Industries Softw., Wilson Res. Group, Sacramento, CA, USA, White Paper, 2022.
[2] G. Cabodi, P. E. Camurati, M. Palena, and P. Pasini, "Hardware model checking algorithms and techniques," *Algorithms*, vol. 17, no. 6, p. 253, Jun. 2024. [Online]. Available: https://www.mdpi.com/1999-4893/17/6/253
[3] E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving," *Formal Methods Syst. Des.*, vol. 19, no. 1, pp. 7–34, 2001. [Online]. Available: http://dblp.uni-trier.de/db/journals/fmsd/fmsd19.html#ClarkeBRZ01

[4] M. Sheeran, S. Singh, and G. Stålmarck, "Checking safety properties using induction and a SAT-solver," in *Formal Methods in Computer-Aided Design*. Heidelberg, Germany: Springer, 2000, pp. 127–144.
[5] K. L. Mcmillan, "Interpolation and SAT-based model checking," in *Computer Aided Verification* (Lecture Notes in Computer Science), vol. 2725. Boulder, CO, USA: Springer, 2003, pp. 1–13.
[6] A. R. Bradley, "SAT-based model checking without unrolling," in *Proc. Int. Workshop Verification Model Checking Abstract Interpretation*, Austin, TX, USA, 2011, pp. 70–87.
[7] 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.
[8] B. Yang, R. E. Bryant, D. R. O'Hallaron, A. Biere, O. Coudert, G. Janssen, R. K. Ranjan, and F. Somenzi, "A performance study of BDD-based model checking," in *Formal Methods in Computer-Aided Design*. Alto, CA, USA: Springer, Nov. 1998, pp. 255–289.
[9] G. Cabodi, S. Nocco, and S. Quer, "Improving SAT-based bounded model checking by means of BDD-based approximate traversals," *J. Universal Comput. Sci.*, vol. 10, no. 12, pp. 1693–1730, 2004.
[10] 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.
[11] P. Chatterjee, J. Meda, A. Lal, and S. Roy, "Proof-guided underapproximation widening for bounded model checking," in *Computer Aided Verification*, S. Shoham and Y. Vizel, Eds., Cham, Switzerland: Springer, 2022, pp. 304–324.
[12] M. Solanki, P. Chatterjee, A. Lal, and S. Roy, "Accelerated bounded model checking using interpolation based summaries," in *Tools and Algorithms for the Construction and Analysis of Systems*, B. Finkbeiner and L. Kovács, Eds., Cham, Switzerland: Springer, 2024, pp. 155–174.
[13] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proc. 38th Design Autom. Conf.*, Jun. 2001, pp. 530–535.
[14] N. Eén and N. Sörensson. (Apr. 2009). *The Minisat SAT Solver*. [Online]. Available: http://minisat.se
[15] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs," in *Proc. Design Autom. Conf.*, New Orleans, LA, USA, Jun. 1999, pp. 317–320.
[16] 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.
[17] 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.
[18] G. Cabodi, P. E. Camurati, C. Loiacono, M. Palena, P. Pasini, D. Patti, and S. Quer, "To split or to group: From divide-and-conquer to sub-task sharing for verifying multiple properties in model checking," *Int. J. Softw. Tools Technol. Transf.*, vol. 20, no. 3, pp. 313–325, Jun. 2018, doi: 10.1007/s10009-017-0451-8.
[19] 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.
[20] A. Biere and T. Jussila. *The Model Checking Competition Web Page*. Accessed: 12 Jun. 2024. [Online]. Available: http://fmv.jku.at/hwmcc

GIANPIERO CABODI received the degree in electronic engineering, in 1984, and the Ph.D. degree in computer and system engineering, in 1988. He is 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 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 Consorzio Nazionale Interuniversitario per le Telecomunicazioni (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.

Open Access funding provided by 'Politecnico di Torino' within the CRUI CARE Agreement