Formal Methods in System Design (2022) 60:117–146
https://doi.org/10.1007/s10703-022-00406-7

Interpolation with guided refinement: revisiting incrementality in SAT-based unbounded model checking

G. Cabodi[1] · **P. E. Camurati**[1] · **M. Palena**[1] · **P. Pasini**[1]

Received: 30 July 2019 / Accepted: 2 November 2022 / Published online: 8 December 2022

Abstract
This paper addresses model checking based on SAT solvers and Craig interpolants. We tackle major scalability problems of state-of-the-art interpolation-based approaches, and we achieve two main results: (1) A novel model checking algorithm; (2) A new and flexible way to handle an incremental representation of (over-approximated) forward reachable states. The new model checking algorithm IGR, Interpolation with Guided Refinement, partially takes inspiration from IC3 and interpolation sequences. It bases its robustness and scalability on incremental refinement of state sets, and guided unwinding/simplification of transition relation unrollings. State sets, the central data structure of our algorithm, are incrementally refined, and they represent a valuable information to be shared among related problems, either in concurrent or sequential (multiple-engine or multiple-property) execution schemes. We provide experimental data, showing that IGR extends the capability of a state-of-the-art model checker, with a specific focus on hard-to-prove properties.

Keywords Formal verification · Hardware model checking · Interpolation · SAT solving

A preliminary version [1] of this paper was presented at FMCAD2014 http://www.cs.utexas.edu/users/hunt/FMCAD/FMCAD14/index.shtml/.

This work was supported in part by SRC contract 2012-TJ-2328.

✉ M. Palena
marco.palena@polito.it

G. Cabodi
gianpiero.cabodi@polito.it

P. E. Camurati
paolo.camurati@polito.it

P. Pasini
paolo.pasini@polito.it

[1] Dipartimento di Automatica ed Informatica, Politecnico di Torino, Turin, Italy

1 Introduction

Craig interpolants (ITPs for short) [2, 3], introduced by McMillan [4] in the Unbounded Model Checking (UMC) field, have shown to be effective on difficult verification instances. Though recently challenged by new techniques (IC3, Incremental Construction of Inductive Clauses for Indubitable Correctness [5]), our experience within the field of HWMCC competitions [6] and industrial co-operations shows that interpolation-based approaches still play an important role within a portfolio-based tool.

From a high-level Model-Checking perspective, Craig interpolation is an operator able to compute over-approximated images. The approach can be viewed as an iterative refinement of proof-based abstractions, to narrow down a proof to relevant facts. Over-approximations of the reachable states are computed from refutation proofs of unsatisfied BMC-like runs, in terms of *AND*/*OR* circuits, generated in linear time and space, w.r.t. the proof.

Their most interesting features are completeness and the automated abstraction mechanism. Whereas one of their major challenges is the inherent redundancy of interpolant circuits, as well as the need for fast and scalable techniques to compact them. Improvements over the base method [4] were proposed in [7–12] and [13], in order to push forward applicability and scalability of the technique.

Interpolant compaction is a potential approach that we have specifically addressed in [14–16]. We follow here a second track of research: alternative ITP-based traversal schemes for model checking algorithms, under the underlying purpose of incrementally computing state sets and reducing the complexity of their computation. We also follow the idea of incrementality in order to support optimal data structures for the verification of multiple properties [17], and for a tighter integration with counterexample- and/or proof-based abstraction/refinement approaches [18, 19].

Our purpose is to improve the standard interpolation algorithm in order to support incremental computation of reachable states sets and dynamic tuning of the backward unrolling from the target. We target incremental data structures in order to enable the reuse of previously computed overapproximations and make interpolant-based algorithms better suited for the verification of multiple properties or for integration with abstraction/refinement approaches. Furthermore, maintaining overapproximated reachable states information in an incremental data structure allows us to dynamically adjust the bound of the BMC formulas checked during each traversal step, in order to better control the precision of the computed image overapproximations.

1.1 Contributions

The main contributions of this work are:

- The integration of a trace data structure in the standard interpolation algorithm, along with a flexible way to compute and refine state set representations;
- An optimization that aims at simplifying the representation of image approximations and/or bad cones (i.e., the set of bad states that are backward reachable in at most k steps from the target.) with ad-hoc redundancy removal;
- Techniques to refine the precision of the computed overapproximations by guiding the bound of the set of backward reachable states from the target used in BMC checks;

– A novel interpolation-based model checking algorithm that makes use of all the above techniques.

1.1.1 Additional remarks

A preliminary version [1] of this paper was originally presented at FMCAD2014. The key differences w.r.t. the original paper are:

– The introduction (Section I) has been revisited in order to better characterize the context;
– Background and notations (Section II) have been extended and improved in order to make the paper as self-contained as possible;
– The description of the proposed algorithm (Sections III through V) have been completely revisited, rewritten and expanded to better illustrate the underlying details of the procedure. Theorems have been introduced to support the theoretical foundation on which the proposed approach is built upon;
– An additional section (Section VI), describing the applicability of IGR alongside lazy abstraction and in a multiple-properties context, has been introduced;
– Experimental results (Section VII) have been revamped to better characterize the proposed algorithm. More in details, we provide a better characterization of the proposed techniques/schemes composing the IGR algorithm itself. We also introduced new benchmarks derived from the latest Hardware Model Checking competitions.

1.2 Outline

Section 2 introduces background notions and notation about BMC and UMC, SAT-based Craig interpolant Model Checking, and IC3. The next three Sections introduce our contributions: Sect. 3 discusses the use of incremental state sets in interpolation, Sect. 4 introduces base concepts on guiding cones through state sets and Sect. 5 presents the overall IGR algorithm. Furthermore, in Sect. 6, we discuss the integration of IGR within lazy abstraction and multiple properties verification loops. The proposed algorithm is experimentally evaluated in Sect. 7. Finally, Sects. 8 and 9 conclude the paper with some summarizing remarks.

2 Background

2.1 Model and notation

We address systems modelled by labelled state transition structures and represented implicitly by 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 representing the states of the system, $\mathcal{I}$ is a Boolean formula over X representing the set of *initial states* of the system and T is a Boolean formula over $X \times X'$ that represents the *transition relation* of the system.

Variables of X are called *state variables* of $\mathcal{S}$. A state of $\mathcal{S}$ is thus represented by a complete truth assignment s to its state variables. Boolean formulas over X represent sets of system states. We denote as Space($\mathcal{S}$) the state space of $\mathcal{S}$. Given a Boolean formula F over X and a complete truth assignment s such that $s \vDash F$, then s is a state contained in the set represented by F and is thus called an *F-state*. Primed state variables X' are used to represent future states of $\mathcal{S}$, i.e., states reached after a transition. Accordingly, Boolean formulas over X' represent sets of future states. We denote as s, s' a complete truth assignment to $X \times X'$ obtained by combining a complete truth assignment s to X and a complete truth assignment s' to X'.

We use transition systems to model the behaviour of hardware sequential circuits, where each state variable $x_i \in X$ corresponds to a latch, the set of initial states I is defined by reset values of latches and the transition relation T is the conjunction $\bigwedge_i (x_i' \leftrightarrow \exists PI.\delta_i(X, PI) = \top)$ being formulas representing the next-state function of each latch. Note that primary inputs PI of the circuit are abstracted away in the resulting transition system.

Definition 2 A *literal* is a Boolean variable or the negation of a Boolean variable. A *clause* is a disjunction of literals whereas a *cube* is a conjunction of literals. A Boolean formula is said to be in *Conjunctive Normal Form* (CNF) iff it is a conjunction of clauses.

Definition 3 A truth assignment for a Boolean formula F over X is a function $\tau : Y \subseteq X \rightarrow \{\top, \bot\}$ that maps variables in Y to truth values. A truth assignment τ for F is *complete* iff $Y \equiv X$, otherwise τ is *partial*.

Definition 4 A truth assignment τ satisfies a literal x, written $\tau \vDash x$, iff $\tau(x) = \top$. Conversely, a truth assignment τ satisfies a literal $\neg x$ iff $\tau(x) = \bot$. A truth assignment τ satisfies a clause C, written $\tau \vDash C$, iff at least a literal in C is satisfied by τ. A truth assignment τ satisfies a CNF formula F, written $\tau \vDash F$, iff each clause in F is satisfied by τ.

Definition 5 A Boolean formula F is *satisfiable* iff there exists a truth assignment τ for F so that $\tau \vDash F$. Otherwise F is *unsatisfiable*. Two Boolean formulas F and G are *equi-satisfiable* iff either both F and G are satisfiable or both are unsatisfiable.

With abuse of notation we sometimes represent a truth assignment as a set of literals of different variables. A truth assignment represented this way assigns each variable to the truth value satisfying the corresponding literal in the set. We also represent a clause (cube) as a set of literals, leaving the disjunction (conjunction) implicit when clear from the context.

Most modern SAT solvers [20, 21] adopt clauses as their main representation and manipulation formalism for Boolean functions. Given a Boolean formula F, whenever we need to explicitly indicate its *before/after* version, w.r.t. an evaluation (e.g., a refinement step), we use a -1 superscript for the *before* version: F^{-1}. We will use letters in boldface for arrays of functions: e.g., $\boldsymbol{F} = (F_0, F_1, ...)$.

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 $s, s' \vDash T$ then s is said to be a *predecessor* of s' and s' is said to be a *successor* of s. A sequence of states $\boldsymbol{\pi}^{0,n} = (s_0, \ldots, s_n)$ is said to be a *path* in $\mathcal{S}$ iff $s_i, s'_{i+1} \vDash T$ for every couple of adjacent states in the sequence (s_i, s_{i+1}), $0 \leq i < n$.

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

Definition 8 A state $s \in \text{Space}(\mathcal{S})$ is said to be *reachable within k steps* (or *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 $s \in \text{Space}(\mathcal{S})$ is said to be *reachable* in $\mathcal{S}$ if it is reachable within an arbitrary (finite) number of steps in $\mathcal{S}$.

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

Definition 11 We denote with $\mathcal{R}_i(\mathcal{S})$, 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 We define the *reachability diameter* of $\mathcal{S}$ to be the minimal number $d \in \mathbb{N}$ of steps required for reaching 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 We denote with $\mathcal{R}(\mathcal{S})$, the *set of states reachable* in $\mathcal{S}$, i.e.,

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

Whenever more time frames are involved in a formula, we use a superscript notation: e.g., in circuit unrollings, we use X^i for the X variables instantiated at the i-th time frame. Support variables will be omitted for simplicity when they can be easily guessed from the context.

Definition 14 A *path formula* of length $k = j - i$ from timeframe i to timeframe 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:

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

A path formula $\Pi(i, j)$ represents all paths of length $k = j - i$ starting at timeframe i in $\mathcal{S}$, whereas an initial path formula $\Pi_0(k)$ describes all paths of length k starting from the initial states in $\mathcal{S}$. An initial path formula $\Pi_0(k)$, therefore, can be used to represent the set of states reachable in exactly k steps from the initial states in $\mathcal{S}$.

Definition 16 Given a transition system $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ we define an *invariant property P* as a Boolean formula that must hold true in every reachable state s of $\mathcal{S}$, i.e.,

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

We call *target* the set of states represented by $\neg P$. States or sets of states are called *bad* if they are part of the target or can reach the target.

Definition 17 Given a transition system $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ and an invariant property P over X a propositional formula F over X is said to be *safe* w.r.t. P iff F is stronger than P, i.e., $F \rightarrow P$.

Definition 18 A *bad cone* of length $k = j - i$ from timeframe i to timeframe 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 $Cone(i, j)$ represents all paths starting at timeframe i that reach the target in at most $k = j - i$ steps, i.e., represents the set of bad states that are backward reachable in at most k steps from the target.

Definition 19 Given a transition system $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$, a *trace* of length k with respect to $\mathcal{S}$ is a sequence $\mathbf{F_k} = (F_0, \ldots, F_k)$ where each F_i is a propositional formula over X, called *frame*, such that the following conditions hold:

$$F_0 = \mathcal{I} \qquad (Base)$$
$$F_i \wedge T \rightarrow F'_{i+1} \text{ for } 0 \leq i < k \qquad (Image\ Approximation)$$

A trace $\mathbf{F_k}$ may also satisfy one or both of the following additional conditions, being P an invariant property over X:

$$F_i \rightarrow F_{i+1} \text{ for } 0 \leq i < k \qquad (Monotonicity)$$
$$F_i \rightarrow P \text{ for } 0 \leq i < k \qquad (Safety)$$

A trace $\mathbf{F_k}$ is said to be *monotonic* if it satisfies the monotonicity condition. A trace $\mathbf{F_k}$ is said to be *safe* (with respect to P) if it satisfies the safety condition. Note that for a trace $\mathbf{F_k}$ to be safe with respect to P according to the previous definition it is not required for frame F_k to be safe. Figure 1 provides a graphical representation of different kinds of traces.

Definition 20 Given a transition system $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$, let F be a propositional formula over X, F is said to be an *inductive invariant* of $\mathcal{S}$ if it satisfies the following conditions:

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

Note that an inductive invariant F of $\mathcal{S}$ is an over-approximation of the set of reachable states $\mathcal{R}(\mathcal{S})$.

Definition 21 Given a transition system $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ and an invariant property P over X, an inductive invariant F of $\mathcal{S}$ is said to be an *inductive strengthening* of P iff it is safe w.r.t. P.

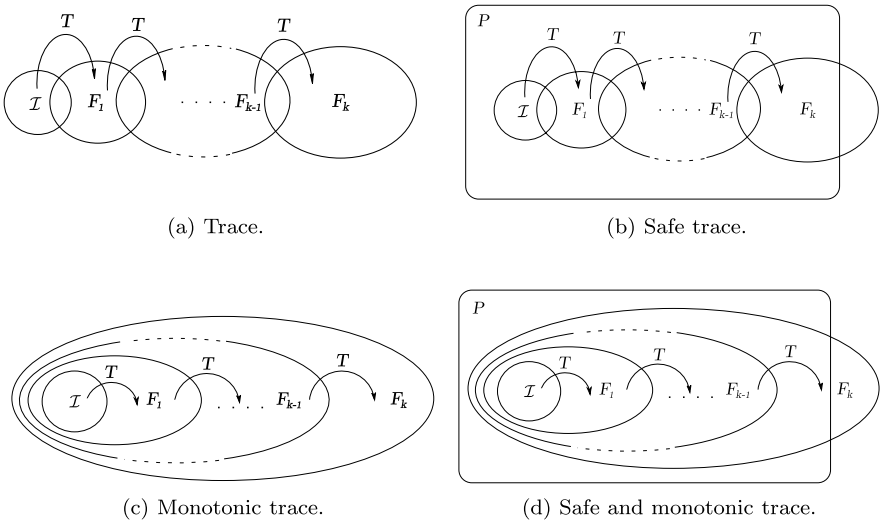


Fig. 1 Different types of traces with respect to a given $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$

2.2 Bounded and unbounded 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 (BMC) [22] is an iterative process to check whether there exists a counterexample to P of length at most k in $\mathcal{S}$ or to prove its absence. In order to do this, BMC simply performs a SAT check on a formula defined as follows.

Definition 22 A *BMC formula* of length k for P in $\mathcal{S}$ is the propositional formula $bmc(k)$ over $X^0 \cup \cdots \cup X^k$:

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

Intuitively, a BMC formula of length k represents all initial paths in $\mathcal{S}$ of length at most k that reach a bad state in $\neg P$. If the formula is SAT, there exists a counterexample to P of length at most k in $\mathcal{S}$. Otherwise, no such a counterexample exists.

BMC tools iteratively solve BMC formulas of increasing bound, until either a counterexample is found or some maximum bound is reached. Though BMC is effective at finding counterexamples, it is not able to detect whether P holds in $\mathcal{S}$. Therefore, specific techniques are required in order to support Unbounded Model Checking. The ability to check reachability fix-points and/or to find inductive invariants is thus the main difference, and additional complication, between BMC and UMC.

2.3 Interpolation-based model checking

Craig's interpolation theorem is a seminal result in mathematical logic about the relationship between model theory and proof theory. The original formulation of the

theorem, due to Craig [2], was given in the context of first-order logic. Variants of the theorem hold for other logical systems as well, including propositional logic. We provide here the formulation of the theorem in propositional logic, which is the one typically encountered in the context of model checking.

Theorem 1 *Given two propositional formulas A and B, if $A \wedge B$ is unsatisfiable then there is a propositional formula I, called interpolant between A and B, such that* (1) $A \rightarrow I$ *is valid*, (2) $I \wedge B$ *is unsatisfiable and* $\mathrm{Vars}(I) \subseteq \mathrm{Vars}(A) \cap \mathrm{Vars}(B)$.

Intuitively, I is an abstraction of A from the viewpoint of B that summarizes and translates in the shared language between A and B, the reasons why A is inconsistent with B. We denote with $I = \mathrm{ITP}(A, B)$ the procedure that derives a Craig's interpolant from a pair of inconsistent formulas A and B.

Interpolants can be derived from refutation proofs of unsatisfiable SAT solving runs. Given an unsatisfiable formula $A \wedge B$, most SAT solvers are capable to generate a proof of refutation either in resolution-based or clausal form. In the case of resolution proofs, an interpolant $I = \mathrm{ITP}(A, B)$ can be derived as an AND/OR combinational circuit in polynomial time and space with respect to the size of the proof. In the context of model checking, if A represents a set of reachable states and B represents a set of bad states, then the interpolant $I = \mathrm{ITP}(A, B)$ is a safe overapproximation of A with respect to B. As a result, such overapproximations can be used to detect a reachability fix-point. The first complete algorithm for symbolic model checking based on Craig's interpolation is due to McMillan [4]. Such an algorithm, called ITP or *standard interpolation*, computes Craig's interpolants to overapproximate reachable states of the system. Such interpolants are computed from refutation proofs of unsatisfiable BMC runs.

The algorithm is composed of two nested loops. The outer loop is implemented in procedure ItpModelChecking (Algorithm 1) whereas the inner loop is implemented in procedure ApproxFwdTrav (Algorithm 2). At each iteration of the outer loop, the procedure ApproxFwdTrav is invoked to perform an overapproximated forward traversal of the reachable states while keeping safety with respect to a backward unrolling from the target (*bad cone*). ApproxFwdTrav can be thought of as computing a safe monotonic trace. The trace is not explicitly maintained, instead only its final frame is kept at each iteration and used as a base for computing the next one.

The procedure ApproxFwdTrav operates a forward traversal in which interpolation is used as an overapproximated image operator. At each iteration the procedure checks a BMC formula of fixed length k, composed of two parts:

$$A \stackrel{\text{def}}{=} R(X^0) \wedge T(X^0, X^1)$$
$$B \stackrel{\text{def}}{=} \mathit{Cone}(1, k) = \mathit{\Pi}(1, k) \wedge \bigvee_{i=1}^{k} \neg P(X^i)$$

where R is a set of overapproximated forward reachable states. It is easy to see that A represents the image of the set of states at the current traversal step, whereas B represents the set of bad states that are backward reachable in at most $k-1$ transitions from the target. If at a given iteration the results of the BMC check is Sat then a possibly spurious counterexample has been found (the forward traversal has hit the cone).

Being an overapproximation of the image of R, the interpolant is treated as a *candidate inductive invariant*. The algorithm checks whether consecution $I \rightarrow R$ is valid (i.e., $\neg R \wedge I$ is unsatisfiable). If that is the case, R is an inductive invariant for $\mathcal{S}$ and, since R is safe w.r.t. P, it is also an inductive strengthening for P. Otherwise, a new set of overapproximated forward reachable states is computed as $R \vee I$ and the algorithm iterates. The

sequence of R composed at each iteration of the nested procedure can be thought of as a safe monotonic trace. The monotonicity is due to the fact that the first R is initialized with $\mathcal{I}$ (line 2) and each consecutive R is a disjunction of the previous one and of an overapproximation of the states reachable from the previous (line 14). Safety (with respect to P) follows from the fact that $\mathcal{I}$ was proved to be safe (Algorithm 1, lines 2–4) and each interpolant I used to construct R does not intersect $Cone(1, k)$. The sequence of interpolants, instead, can be seen as a non-monotonic safe trace.

Considering Algorithm 1, first the initial states $\mathcal{I}$ are checked to be safe (lines 2–3). If that is not the case, there is a trivial counterexample consisting of a single initial state only. The procedure then terminates returning the counterexample found. Otherwise, the bound k for the bad cone is initialized to 1 (line 4) and the procedure starts iterating forward overapproximated traversals of reachable states while keeping safety with respect to a bad cone of increasing depth k from the target (lines 5–11). Increasing the bound k helps finding real counterexamples and generating more precise overapproximations on the next iteration. This is because as the bad cone from the target unwinds, some of the states in the overapproximated images computed at previous iterations would be reached by the unrolling and therefore excluded from the new images. Note that at each iteration of the outer loop, the nested procedure starts a forward traversal from scratch from the initial states. As k increases, the algorithm is guaranteed to find a bound k in which the computed interpolants are precise enough to find an inductive strengthening if P holds for the system, or to find a real counterexample otherwise.

The overall algorithm may end up with three possible results:

- *reachable*, if it proves $\neg p$ reachable in k steps, hence the property has been disproved;
- *unreachable*, if the approximate traversal reaches a fix-point. In this case the property is proved;
- *undefined*, if the target is reachable from the over-approximate state set in un to k steps. Then, k in increased for a new iteration.

The algorithm is sound and complete [4].

Algorithm 1. *Top-level procedure of McMillan's interpolation algorithm. It iterates forward overapproximated traversals of reachable states while keeping safety with respect to a bad cone of increasing depth from the target.*

Input: $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ a transition system; P a property over X.
Output: $\langle res, cex \rangle$ with $res \in \{\text{SUCCESS}, \text{FAIL}\}$; cex a (possibly empty) initial path representing a counterexample.

```
1:  procedure ItpModelChecking(S, P)
2:      if ∃s0 ⊨ I(X) ∧ ¬P(X) then
3:          return ⟨Fail, (s0)⟩
4:      k ← 1
5:      while true do
6:          ⟨res, cex⟩ ← ApproxFwdTrav(S, P, k)
7:          if res is Unreach then
8:              return ⟨Success, −⟩
9:          else if res is Reach then
10:             return ⟨Fail, cex⟩
11:         k ← k + 1
```

Algorithm 2. *Inner procedure of McMillan's interpolation algorithm. It operates a forward overapproximated traversal of the reachable state space while keeping safety with respect to a bad cone of fixed depth from the target.*

Input: $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ a transition system; P a property over X; k bound of a backward unrolling from the target.
Output: $\langle res, cex \rangle$ with $res \in \{\text{REACH}, \text{UNREACH}, \text{UNDEF}\}$; cex a (possibly empty) initial path representing a counterexample.

1: **procedure** APPROXFWDTRAV($\mathcal{S}$, P, k)
2: $R \leftarrow \mathcal{I}$
3: **if** $\exists \boldsymbol{\pi}^{0,k} \models R(X^0) \wedge T(X^0, X^1) \wedge Cone(1, k)$ **then**
4: **return** $\langle \text{REACH}, \boldsymbol{\pi}^{0,k} \rangle$
5: **while** $\top$ **do**
6: $A \leftarrow R(X^0) \wedge T(X^0, X^1)$
7: $B \leftarrow Cone(1, k)$
8: **if** $\exists \boldsymbol{\pi}^{0,k} \models A \wedge B$ **then**
9: **return** $\langle \text{UNDEF}, - \rangle$
10: **else**
11: $I \leftarrow \text{ITP}(A, B)$
12: **if** $\nexists s \models I \wedge \neg R$ **then**
13: **return** $\langle \text{UNREACH}, - \rangle$
14: $R \leftarrow R \vee I$

2.4 IC3

IC3 [5] is a SAT-based algorithm for symbolic invariant verification. Given a transition system $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ and an invariant property P over X to be checked, IC3 aims at finding an inductive strengthening of P for $\mathcal{S}$.

To this end, IC3 maintains two main data structures. The first is a *trace* $\mathbf{F_k} = (F_0, \ldots, F_k)$ that is both monotonic and safe w.r.t. the property P. At a given iteration of the algorithm, being $\mathbf{F_k}$ such a trace, each frame F_i, with $0 \leq i < k$, is a safe overapproximation of the set of states reachable in at most i steps in $\mathcal{S}$. The purpose of the algorithm is to iteratively refine such $\mathbf{F_k}$ in order to satisfy the condition $F_{i+1} \rightarrow F_i$ for some $0 \leq i < k$, thus finding an inductive strengthening of ψ. In order to do this, IC3 maintains a second data structure called *proof-obligation queue* that is used to collect sets of states in $\mathbf{F_k}$ that can reach a violation of the property in some number of steps. IC3 processes those sets of states according to a given priority and for each of them it either finds a backward path to the initial states or learns a new inductive lemma that can be used to refine $\mathbf{F_k}$ to exclude such states from the overapproximation. In the first case the algorithm has found a counterexample to P. In the second case, the algorithm continues its search of an inductive strengthening of P over a tighter approximation of the reachable states sets. At various points during its operation, IC3 requires to solve SAT calls. A peculiarity of the algorithm is the fact that its SAT calls are very frequent but involve only a single instance of the transition relation T. Performing many *local reachability checks*, IC3 achieves a better control on the precision of the computed overapproximations.

Considering the trace $\mathbf{F_k}$, each frame F_i is represented by a set of clauses, denoted by clauses(F_i), in order to enable efficient syntactic checks for the equality of frames. The base condition of the trace is fulfilled by initializing the first frame with $\mathcal{I}$ at the start of the algorithm. Monotonicity is maintained syntactically, by enforcing the condition clauses(F_{i+1}) $\subseteq$ clauses(F_i). Image approximation and safety, instead, are guaranteed explicitly by the algorithm's operations.

The introduction of IC3 suggested a different way to compute information about reachable states, as (unlike ITP-based approaches) IC3 requires no unrolling of the transition relation. One of the major contributions of IC3 is an inductive reasoning, where induction is exploited under stepwise assumptions-assertions. IC3 is incremental in that it finds inductive subclauses of the negations of states. The main limitation of IC3 is the potential clause-based state set enumeration. Some interesting ideas of IC3, that partially influenced our work, are:

- The incremental representation of state sets;
- The *push* operation, that possibly re-uses clauses from inner state sets to outer ones;
- Redundancy removal by subsumption.

3 Incremental state sets in ITP

In this section we describe our model of incremental state sets. Instead of directly introducing the overall IGR algorithm (see Sect. 5), we first propose here some modifications to the standard interpolation algorithm [4], that would allow reusing and refining previously computed interpolants. In the proposed variant a trace of overapproximations to reachable states is maintained and incrementally refined, in order to enable the reuse of previously computed interpolants. Since interpolants are safe image overapproximations with respect to the property under verification, the trace maintained by our variant of ITP is safe as well.

As already pointed out, incremental state sets are present in ITPSEQ[23, 24] and DAR[25]. Compared to those works, our approach, as described in the sequel, is much closer to standard interpolation. More in detail:

- We just focus on approximations of forward reachable states, with no attempt to mix forward and backward state sets (as in DAR);
- We keep the standard interpolation scheme, extended by saving and reusing previously computed state sets;
- We always refine (i.e., strengthen) state sets, which does not prevent us from possibly simplifying their representation by using ad–hoc redundancy removal.

At each i-th iteration of the inner loop an ITP overapproximation of the states reachable in i steps in the system is computed by extracting an interpolant from the refutation proof of a BMC formula. The computed interpolant is discarded at the end of the iteration. Furthermore, when a spurious counterexample is found, the current forward traversal is interrupted, the backward cone from the target is unwound by one step and the forward traversal of reachable states restarts from the initial states. The key idea of the proposed method is to keep track of the overapproximations computed during each run of ApproxFwdTrav, in order to enable their reuse in further iterations of the outer loop. In order to do this, we extend the standard interpolation algorithm to maintain a *trace* of reachable states. As the bound k of the cone increases, stronger overapproximations are computed at each traversed time frame and used to refine the trace.

Algorithm 3. *Top-level procedure of the proposed ITP variant that keeps track of the computed interpolants using a trace.*

Input: $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ a transition system; P a property over X.
Output: $\langle res, cex \rangle$ with $res \in \{\text{SUCCESS}, \text{FAIL}\}$; cex a (possibly empty) initial path representing a counterexample.

```
1: procedure IncrItpModelChecking(S, P)
2:     if ∃s_0 : s_0 ⊨ I(X) ∧ ¬P(X) then
3:         return ⟨FAIL, (s_0)⟩
4:     F_k[0] ← I
5:     k ← 1
6:     while true do
7:         ⟨res, cex⟩ ← IncrApproxFwdTrav(S, P, F_k, k)
8:         if res is UNREACH then
9:             return ⟨SUCCESS, −⟩
10:        else if res is REACH then
11:            return ⟨FAIL, cex⟩
12:        k ← k + 1
```

We use a trace $\mathbf{F_k} = (F_0, \ldots, F_k)$ in order to keep track of previously computed overapproximations. Each timeframe of the trace is represented as an AIG circuit. Each timeframe F_i of $\mathbf{F_k}$, with $0 \leq i < k$, is an over-approximation of the set of states that are reachable in exactly i steps. The trace is constructed so that it is safe with respect to P.

Algorithm 4. *Inner procedure of the proposed ITP variant that keeps track of the computed interpolants using a trace.*

Input: $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ a transition system; P a property over X; $\mathbf{F_k}$ a trace; k bound of a backward unrolling from the target.
Output: $\langle res, cex \rangle$ with $res \in \{\text{REACH}, \text{UNREACH}, \text{UNDEF}\}$; cex a (possibly empty) initial path representing a counterexample.

```
1: procedure IncrApproxFwdTrav(S, P, F_k, k)
2:     R ← F_0
3:     if ∃π^{0,k} ⊨ F_0(X^0) ∧ T(X^0, X^1) ∧ Cone(1, k) then
4:         return ⟨REACH, π^{0,k}⟩
5:     i ← 0
6:     while ⊤ do
7:         if i = |F_k| then
8:             F_k[i + 1] ← ⊤
9:         A ← F_i(X^0) ∧ T(X^0, X^1)
10:        B ← Cone(1, k)
11:        if ∃π^{0,k} ⊨ A ∧ B then
12:            return ⟨UNDEF, −⟩
13:        else
14:            I ← ITP(A, B)
15:            F_{i+1} ← F_{i+1} ∧ I
16:            if ∄s ⊨ F_{i+1} ∧ ¬R then
17:                return ⟨UNREACH, −⟩
18:            R ← R ∨ F_{i+1}
19:            i ← i + 1
```

The proposed ITP variant, called IncrItpModelChecking, is sketched in Algorithms 3 and 4. The differences between the proposed variant and standard interpolation are the following:

- The trace is initialized with $F_0 = \mathcal{I}$ prior to starting the first forward traversal (Algorithm 3, line 4).[1]
- Each time the forward traversal reaches the end of the current trace, a new frame is F_{i+1} instantiated equal to $\top$ and added to the trace (Algorithm 4, lines 7–8).
- Every time a new interpolant, overapproximating states reachable in $i + 1$ steps, is computed, the corresponding frame F_{i+1} in the trace is refined (Algorithm 4, lines 15).

Note that refinement is a strengthening step, performed by conjoining the previous set with a new term. In the following we prove that the sequence of formulas $(F_0, \ldots, F_k)$ computed by the algorithm constitutes a safe trace for the system with respect to P.

Theorem 2 *At any moment during the execution of Algorithm* 4, *the sequence* $\mathbf{F_k} = (F_0, \ldots, F_k)$ *maintained by the algorithm is a trace.*

Proof Given a sequence $\mathbf{F_k} = (F_0, \ldots, F_k)$ computed by Algorithm 4, we need to prove that the base and image approximation conditions of Definition 19 hold for $\mathbf{F_k}$. Since F_0 is initialized with $\mathcal{I}$ (Algorithm 3, line 4) and never refined, the base condition holds. We prove that the image approximation condition holds by induction on the refinement step (Algorithm 4, line 16). Assume $\mathbf{F_k}$ to be a trace prior to a refinement step. We denote by F_i^* the i-th element of the sequence after the refinement step. Since the sequence prior to the refinement is assumed to be a trace, $F_i \wedge T \rightarrow F_{i+1}$ is valid for each $0 \leq i < k$. From the definition of $I =$ ITP $(F_i \wedge T, B)$ it follows that $F_i \wedge T \rightarrow I$ is valid. By conjoining the two together, the following is valid:

$$F_i \wedge T \rightarrow F_{i+1} \wedge I \tag{1}$$

Therefore, considering the image approximation condition after a refinement step, we have:

$$\begin{aligned} F_i^* \wedge T &\rightarrow F_{i+1}^* \\ = \; F_i \; \wedge T &\rightarrow F_{i+1} \wedge I \end{aligned}$$

which is valid according to Formula 1. Refinement preserves the image approximation condition of Definition 19, therefore $\mathbf{F_k} = (F_0, \ldots, F_k)$ is a trace. □

Theorem 3 *At each moment during the execution of Algorithm* 4, *the trace* $\mathbf{F_k} = (F_0, \ldots, F_k)$ *maintained by the algorithm is safe.*

Proof The safety of the trace follows trivially by induction. Timeframe F_0 is initialized with $\mathcal{I}$, which the algorithm checks to be safe (Algorithm 3, lines 2–4). Each subsequent timeframe is initialized with $\top$ (which is not safe) and refined by conjunction of interpolants (which are safe according to the definitions of interpolants and bad cone). Therefore, for each timeframe F_i of $\mathbf{F_k}$, with $0 \leq i < k$, the safety condition $F_i \rightarrow P$ holds. Note that, the outermost timeframe F_k is the only one to be potentially unsafe (equal to $\top$), but its safety is not required according to Definition 19.

[1] We use the notation $\mathbf{F_i}$ instead of F_i to refer to a frame of $\mathbf{F_k}$ that does not exists yet and that is being initialized for the first time.

3.1 Frames and cone simplification

In this subsection we describe an optimization that aims at simplifying the representation of frames and/or bad cones with ad-hoc redundancy removal. The purpose of the proposed optimization is to keep overapproximations of reachable states and cones small. Such a simplification step is based on the general notion of redundancy removal under *observability don't cares*. We denote *simplification under a care set* as the function $\textsc{Simplify}(F, C)$, where F is a formula over X to be simplified and C is another formula over X to be used as a care set for the simplification of F. The care set is defined with respect to a reference formula G over $X \cup W$ in which F appears as a subformula, as follows.

Definition 23 Given a propositional formula F over X and another formula G over $X \cup W$ such that F is a subformula of G, we define the *care set* C_F^G of F with respect to G as the set of assignments over X under which the value of F affects the value of G. A care set for F with respect to G can be represented by the formula:

$$C_F^G = G \oplus G[F \leftarrow \neg F]$$

The complement of a care set C_F^G is called *don't care set* of F with respect to G and it represents the set of assignments over X under which the value of F does not affect the value of G.

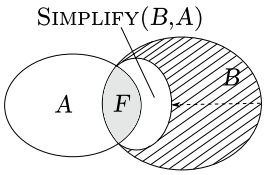
The knowledge of the care set of a formula F with respect to a reference formula G can be used to simplify F. Simplification of a formula F under a care set C_F^G with respect to a reference formula G can involve the application of any number of equivalence-preserving or strengthening transformations over F, as long as the following constraint is preserved:

$$G \equiv G[F \leftarrow \textsc{Simplify}(F, C_F^G)]$$

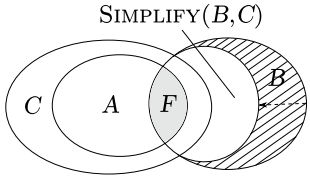
Computation of care sets and don't care sets can be costly [26]. Considering a conjunction $F = A \wedge B$ as a reference formula, the following Lemma describes two straightforward ways to obtain care sets for either of the conjoined formulas. We focus on B being the case for A dual.

Lemma 1 *Given $F = A \wedge B$, with both A and B propositional formulas over X, then A is a care set for B with respect to F. Given C a propositional formula over X such that $A \rightarrow C$, then C is a care set for B with respect to F.*

Figures 2a and b illustrate, in terms of sets of assignments, the simplification of a formula B with respect to a reference formula $F = A \wedge B$ under care sets as defined by Lemma 1. For instance, given two propositional formulas $A = a$ and $B = (a \vee b)$, the knowledge that any assignment satisfying their conjunction $F = A \wedge B$ must be a satisfying assignment of either formula, can be used to simplify the other. Using A as a care set for B, since $F = a \wedge (a \vee b)$ is satisfied only by assignments satisfying A (i.e., assignments μ such that $\mu(a) = \top$) we can simplify B through the injection of a constant $\top$ prior to conjoining it to A, obtaining $\top \vee b \equiv b$. The resulting formula after simplification $A \wedge \textsc{Simplify}(B, A) = a \wedge b$ is syntactically more compact than the equivalent $A \wedge B = a \wedge (a \vee b)$.



(a) Simplification of B using A as care set. The SIMPLIFY procedure simplifies B without affecting its conjunction F with A.



(b) Simplification of B using C as care set, with $A \rightarrow C$. The SIMPLIFY procedure simplifies B without affecting its conjunction F with A.

Fig. 2 Examples of simplification under a care set

Though many redundancy removal techniques can be used to perform simplification under a care set, such as latch correspondence, signal correspondence and equivalence to constants, our experience shows that most of them are too expensive to perform at each forward traversal iteration of an ITP scheme. As we need a fast operator, we limit SIMPLIFY to the removal of equivalences between state variables, also known as *latch correspondences*. Given a formula B to simplify and a care set A, the SIMPLIFY operator identifies pairs of state variables (x_1, x_2) such that $A \rightarrow (x_1 \leftrightarrow x_2)$. Then, B is simplified as $B[x_1 \leftarrow x_2]$.

Simplification under a care set can be used to simplify the representation of frames during the refinement step (Algorithm 4, line 17), or to simplify the bad cone $Cone(1,k)$ prior to checking its intersection with the image of F_i (Algorithm 4, line 13). During forward overapproximated traversal, when the image of the current set of reachable states $F_i \wedge T$ is checked for intersection with the bad cone $Cone(1,k)$, we can simplify $Cone(1,k)$ using any overapproximation of the states reachable in the next k transitions that is already in the trace. In particular, each frame F_j in $\mathbf{F_k}$, with $i < j < i + k$, can be used as a care set to simplify $Cone(1,k)$. We denote with TRACESIMPLIFY($Cone(1,k)$, $\mathbf{F_k}$, i, k) the function applying SIMPLIFY($Cone(1,k), F_j$) for each $i < j < i + k$, i.e., the function applying latch correspondences substitution at each intermediate transition relation boundaries in $Cone(1,k)$. This way, we exploit reachability information computed during previous iterations of the algorithm to simplify the formula to be fed to the SAT solver.

4 Guided cone

In standard interpolation, when a spurious counterexample is found the current forward traversal is restarted from the initial states after the bad cone has been expanded by one step. We explore the idea to dynamically unwind or rewind the cone during forward traversal in order to guarantee the refinement of some previously computed overapproximation of reachable states. The depth of the cone is therefore guided by the frames in the trace so that it can lead to a strengthening refinement for some of them.

Compared to the proposed approach, ITPSEQ and DAR refine frames based on BMC-like runs of growing depth. IC3, instead, drives the refinements based on a prioritized selection of backward reachable cubes.

Supposing that at a given point during the execution of INCRITPMODELCHECKING the algorithm has computed a trace $\mathbf{F_k}$, we follow two directions sharing the goal of potentially expanding and refining $\mathbf{F_k}$:

Cone Unwinding When the forward traversal hits the cone, we start a new traversal at an intermediate step in order to guarantee the refinement of the trace.

Cone Rewinding When the forward traversal hits the cone, we continue the traversal with iteratively smaller cones in order to refine and expand the trace.

Overall, guided cone unwinding/rewinding allows us to dynamically tune the unrolling from the target and therefore to have a better control over the precision of the computed overapproximations (interpolants). In this respect, standard interpolation is too rigid, as overapproximations are always strengthened expanding the cone by one and restarting the traversal from scratch. If the diameter of the system is very large, ITP takes a large number of iterations to converge. ITPSEQ, computes overapproximations to the reachable states incrementally, but with a fixed and rigid scheme. Much more flexibility is present in DAR, where local and global strengthening techniques are used to refine just when and where needed. Although backward refinement in DAR has similarities to our approach, it is based on the idea of using overapproximated backward reachable states when refining forward reachable ones. Our approach, instead, is based on backward cones in order to represent the exact backward behaviour.

4.1 Cone unwinding

At a given iteration i of the forward traversal, given the bound k of the cone, if the following formula is SAT, then a possibly spurious counterexample is found.

$$F_i(X^0) \wedge T(X^0, X^1) \wedge Cone(1, k) \tag{2}$$

Standard interpolation, in that case, unwinds the bad cone by one step and starts a new traversal from the initial states. By doing so, when/if step i is reached again in the traversal, the overapproximation F_i might have been strengthened enough to exclude the spurious counterexample previously found. If that is not the case, standard interpolation restarts the traversal again, incrementing k until either all spurious counterexamples are excluded from the overapproximation or (at least) one counterexample is confirmed to be real. We propose an alternative approach to handle spurious counterexamples, the purpose being to reuse the computed overapproximations as much as possible. In our approach, we unwind the cone of the minimum depth necessary to strengthen a frame F_j, with $0 < j \leq i$, in order to directly refute the spurious counterexample.

Whenever Formula 2 is SAT, there is a path from a state $\sigma \in F_i$ to a target state and so there could exist a counterexample of length (at most) $i + k$. The counterexample is feasible if σ is reachable from the initial states, otherwise we say that the counterexample is spurious.

Theorem 4 *A counterexample is spurious if the following formula is* UNSAT *for a given* j, *with* $0 < j \leq i$:

$$F_{i-j}(X^0) \wedge T(X^0, X^1) \wedge Cone(1, k + j) \tag{3}$$

Proof We can demonstrate such a claim by reduction to absurdity. Suppose that Formula 3 is UNSAT for a given j, with $0 < j \leq i$. Suppose also that $bmc(i + k)$ is SAT, therefore a counterexample of length (at most) $i + k$ exists. Then, being F_l an overapproximation of the

states reachable in l steps, for all l such that $0 \leq l < i + k$ the following formula must be SAT:

$$F_l(X^0) \wedge T(X^0, X^1) \wedge Cone(1, k + i - l) \tag{4}$$

By substituting $j = i - l$ in Formula 3 we obtain Formula 4. The contradiction follows from the fact that such a formula should be UNSAT by assumption. □

Theorem 4 can be used in multiple ways in order to set up a refinement process of F_i. We call "cone unwinding" the process of iteratively finding a minimal value of j, such that Formula 3 is UNSAT. Once j is found, we can restart the interpolation process at F_{i-j} with the cone "unwound" by j time frames, $Cone(1, k + j)$.

With respect to standard interpolation we skip intermediate cones, whenever $j > 1$, and we skip the initial traversal iterations, from F_1 to F_{i-j}, which are reused instead of being recomputed.

In order to check if a counterexample is feasible, we start an iterative process checking BMC-like problems in which every time we unwind the cone by one step and we consider the previous frame. Starting from frame F_i and cone $Cone(1, k)$, at the j-th iteration we consider the BMC-like problem described in Formula 3. At each iteration we trade an over-approximated set of forward reachable states F_{i-j} for an exact image backward reachable from the target, $Cone(1, k + j)$. If the BMC-like problem is SAT at a given j, then F_{i-j} is not strong enough to refute the counterexample and the process iterates. Eventually, either we find UNSAT, refuting the spurious counterexample, or we reach the initial states, confirming the counterexample as a concrete one. In fact, for $j = i$ we have exactly $bmc(i + k)$.

Supposing that we have found a j, with $0 < j < i$, such that Formula 3 is UNSAT, then we can restart the forward traversal from F_{i-j} with $Cone(1, k + j)$, which is guaranteed to generate an interpolant and refine the current trace, as proved in the following theorem.

Theorem 5 *Code unwinding is guaranteed to refine the current trace.*

Proof Given an iteration i of the forward traversal procedure and a bound k of the cone for which Formula 2 is SAT, cone unwinding can be applied. Suppose that, for a given iteration j, Formula 3 is UNSAT for j but SAT for $j - 1$, then the interpolant I computed for Formula 3 at iteration j has the property that all paths of length $k + j - 1$ starting from I are safe. Since Formula 3 is SAT for $j - 1$ there exists a state in F_{i-j+1} that reach a bad state in at most $k + j - 1$ steps. Both I and F_{i-j+1} are overapproximations of the states reachable in $i + j - 1$ steps from $\mathcal{I}$. By conjoining I and F_{i-j+1} all the paths that reach a bad state in at most $k + j - 1$ steps are removed, thus refining the current trace. □

From a practical point of view, we are unwinding $Cone(1, k)$ in a guided way through the frames F_{i-j}, with $0 < j \leq i$, in order to find the minimum value of j able to refute a spurious counterexample.

An additional consideration is that, upon detecting j such that Formula 3 is UNSAT, we can continue the iterative process for a given number of bounds checking whether or not Formula 3 still holds for higher values of j. Note that, this is not guaranteed since the trace is non monotonic. Note also that, even if we find Formula 3 to be UNSAT for higher values of j, the corresponding interpolant is not guaranteed to refine the trace. This nevertheless can be empirically useful in order to restart traversal at a lower bound, i.e., closer to $\mathcal{I}$, in

order to avoid to restart too close to frames that may lead to counterexamples in one or just a few steps.

The unwinding strategy is heuristically more suitable to handle problems characterized by deeper bounds, which typically need multiple iterations of standard interpolation to converge.

4.2 Cone rewinding

We call "cone rewinding" a strategy that, given a frame F_i safe with respect to $Cone(1,k)$, seeks to refine the next frames F_{i+j} (with $0 < j < k$) with cones of decreasing bound, that guarantee interpolant computation (UNSAT problems). This strategy is inspired by the variant of interpolation sequences proposed in [24]. We start at F_i, with $Cone(1,k)$, under the condition that Formula 2 is UNSAT. This guarantees that a refinement of F_{i+1} is possible, using the interpolant derived from the corresponding SAT check. As a consequence, it follows from the nature of interpolants that $F_{i+1}(X^1) \wedge Cone(1,k)$ is UNSAT. By performing variable relabelling and making the first instance of transition relation in the cone explicit, we obtain the following formula

$$F_{i+1}(X^0) \wedge T(X^0, X^1) \wedge Cone(1, k-1) \tag{5}$$

which is UNSAT. Formula 5 is the base of the rewinding process, as we have moved one step forward in image computation, from F_i to F_{i+1}, and the cone has been reduced by one time-frame, from $Cone(1,k)$ to $Cone(1,k-1)$.

Rewinding iterations can stop at any intermediate step, or even go ahead until we have a cone of bound 1. This is a purely heuristic choice. We activate the rewinding process in two cases (controlled by engine setup options):

- After cone unwinding, from k to $k+j$, which means that, instead of starting an interpolant iteration with a cone of bound $k+j$, we decrease the cone at each iteration. We call this strategy IGR (A).
- We keep a given cone $Cone(1,k)$ until it is hit (so Formula 2 is SAT), then we start the rewinding process from the previous step, as $F_i(X^0) \wedge T(X^0, X^1) \wedge Cone(1, k-1)$ is guaranteed to be UNSAT. We call this strategy IGR (B).

4.3 Overall strategy

To sum up, IGR (A) and IGR (B) are two ways of exploiting the same ability to refine frames (overapproximations of reachable states). In IGR (A) we know that a given frame F_i is safe respect to $Cone(1,k)$, and we refine, or generate, the frames $(F_{i+1}, \ldots, F_{i+k})$ of the trace. In IGR (B), we initially performs as many image steps as possible with $Cone(1,k)$. Once the cone is hit at step i of the forward traversal, we move back at step $i-1$, and we compute a sequence of k interpolants, that can be used to refine, or generate, the frames $(F_i, \ldots, F_{i+k-1})$ of the trace. The purpose of such refinement sequences is to let future traversals operate over more precise overapproximations of the reachable states.

5 IGR: interpolation with guided refinement

In this section we describe a novel complete invariant verification procedure based on the ideas presented before. The proposed algorithm, called *Interpolation with Guided Refinement* (IGR), can be seen as a variant of ITP that incorporates explicit trace computation and refinement, images and cones simplification under observability don't care and guided cone unwinding/rewinding.

Algorithm 5. *Top-level procedure of IGR.*

Input: $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ a transition system; P a property over X.
Output: $\langle res, cex \rangle$ with $res \in \{\text{Success}, \text{Fail}\}$; cex a (possibly empty) initial path representing a counterexample.

```
1:  procedure IgrModelChecking(S, P)
2:      if ∃s_0 : s_0 ⊨ I(X) ∧ ¬P(X) then
3:          return ⟨Fail, (s_0)⟩
4:      F_k[0] ← I
5:      i_hit ← 0
6:      k_hit ← 1
7:      while true do
8:          ⟨res, cex, i, k⟩ ← Unwind(S, P, F_k, i_hit, k_hit)
9:          if res is Reach then
10:             return ⟨Fail, cex⟩
11:         ⟨res, i_hit, k_hit⟩ ← IgrApproxFwdTrav(S, P, F_k, i, k)
12:         if res is Unreach then
13:             return ⟨Success, −⟩
```

The top-level procedure of IGR is reported in Algorithm 5. The procedure starts by checking safety of the initial states (lines 2–3) and initializing the trace $\mathbf{F_k}$ (line 4). The indexes i_{hit} and k_{hit} are also initialized (lines 5–6). Such indexes are used to keep track of the traversal step and cone depth at which a cone was hit during the previous traversal. Then, the outer loop starts iterating overapproximated forward traversals (lines 7–13). First the procedure Unwind is invoked to seek the best frame at which to start the next traversal (line 8). Such a procedure performs cone unwinding (as described in Sect. 4.1) starting from the step i_{hit} and cone depth k_{hit}, until either a concrete counterexample or a frame that could be refined by computing a new interpolant is found. In the first case, the procedure Unwind returns a Reach result and a counterexample. The algorithm, thus, terminates with Fail producing the counterexample as an output (lines 9–10). In the second case, the procedure Unwind returns an Undef result, a step i and a cone depth k to be used for the next traversal. Note that, at the first iteration, i_{hit} is 0 and k_{hit} is 1, therefore Unwind simply checks whether the initial states can reach the target in one transition. If no real counterexample was found, the algorithm starts a new forward overapproximated traversal invoking the procedure IgrApproxFwdTrav (line 11). Upon termination of such a procedure, if the result is Unreach, then an inductive invariant has been found during traversal. In that case, the algorithm terminates with Success (lines 12–13). Otherwise, the cone was hit and the traversal procedure returns an Undef result together with the step i_{hit} and cone depth k_{hit} at which that occurred. In that case, a possibly spurious counterexample was found during traversal and the algorithm iterates to perform a new traversal.

The overall task of IgrModelChecking can thus be summarized as:

– Iteratively choose a starting frame F_i and a cone $Cone(1, k)$, unwound in a guided manner throughout the (overapproximated) trace $\mathbf{F_k}$.
– Start a new forward traversal from F_i with $Cone(1, k)$ that is expected to refine $\mathbf{F_k}$ and filter out the last spurious counterexample found within F_i.

The first sub-task is handled by UNWIND, whereas the second is performed by IGRAPPROXFWDTRAV.

Algorithm 6. *Inner procedure of the proposed ITP variant that keeps track of the computed interpolants using a trace.*

Input: $\mathcal{S} = \langle X, \mathcal{I}, T\rangle$ a transition system; P a property over X; $\mathbf{F_k}$ a trace; i start step of the traversal; k bound of a backward unrolling from the target.
Output: $\langle res, i_{hit}, k_{hit}\rangle$ with $res \in \{\text{UNREACH}, \text{UNDEF}\}$; i_{hit} the step (if any) at which the cone is hit during traversal; k_{hit} the depth of the cone hit.

1: **procedure** IGRAPPROXFWDTRAV($\mathcal{S}$, P, $\mathbf{F_k}$, k)
2: $R \leftarrow \bigvee_{j=0}^{i} F_j$
3: $rewind \leftarrow \bot$
4: $i_{hit} \leftarrow i$
5: $k_{hit} \leftarrow k$
6: **while** $\top$ **do**
7: **if** $i = |\mathbf{F_k}|$ **then**
8: $\mathbf{F_k}[i+1] \leftarrow \top$
9: **if** $rewind \wedge k > 0$ **then**
10: $k \leftarrow k - 1$
11: $A \leftarrow F_i(X^0) \wedge T(X^0, X^1)$
12: $B \leftarrow \text{TRACESIMPLIFY}(Cone(1, k), \mathbf{F_k}, i+1, k)$
13: **if** $\exists \boldsymbol{\pi}^{0,k} \models A \wedge B$ **then**
14: $rewind \leftarrow \top$
15: $i_{hit} \leftarrow i$
16: $k_{hit} \leftarrow k$
17: **else**
18: $I \leftarrow \text{ITP}(A, B)$
19: $F_{i+1} \leftarrow \text{SIMPLIFY}(F_{i+1}, I) \wedge I$
20: **if** $\nexists s \models F_{i+1} \wedge \neg R$ **then**
21: **return** $\langle \text{UNREACH}, -, -\rangle$
22: $R \leftarrow R \vee F_{i+1}$
23: $i \leftarrow i + 1$
24: **if** $\neg rewind \wedge i > D$ **then**
25: $rewind \leftarrow \top$
26: $i_{hit} \leftarrow i$
27: $k_{hit} \leftarrow k$
28: **else if** $rewind \wedge k = 0$ **then**
29: **return** $\langle \text{UNDEF}, i_{hit}, k_{hit}\rangle$

Procedure IGRAPPROXFWDTRAV, described in Algorithm 6, performs a forward overapproximated traversal of reachable states starting from a given frame in the trace, while keeping safety with respect to a cone of given depth. The procedure first computes the current overapproximated set of states reachable at step i by disjoining the first i frames (line 2). Then, at each iteration of the traversal loop (lines 6–29), the algorithm proceeds in two different ways based on whether or not a *cone rewinding* (see Sect. 4.2) has been triggered, as controlled by the variable *rewind*. If cone rewinding has not been triggered, the algorithm performs a traversal step using a cone of bound k. Otherwise the algorithm decreases the bound of the cone at each iteration to perform rewinding (lines 9–10). During each traversal step, the procedure first performs cone simplification (line 12) as described

in Sect. 3.1 and then checks whether the current set of overapproximated reachable states R hits the cone (line 13). If that is the case, the algorithm saves the current step and cone bound in i_{hit} and k_{hit}, respectively, and triggers a refinement sequence (lines 14–16). Otherwise, a new overapproximated image is computed through interpolation (lines 18) and the current frame F_i is refined and simplified (line 19) as described in Sect. 3.1. Then, the algorithm checks if the computed overapproximation is an inductive invariant, returning UNREACH if that is the case (lines 20–21). If no inductive invariant has been found, the new set of overapproximated forward reachable states is computed as $R \vee F_{i+1}$, to be used for the next iteration (line 22). Each time the forward traversal reaches the end of the current trace, a new frame is F_{i+1} instantiated equal to $\top$ and added to the trace (lines 7–8). At the end of each iteration, if a given (user-controllable) traversal depth threshold D has been reached, the algorithm forces rewinding (lines 24–27). When rewinding has been triggered, either because a spurious counterexample was found or because it was forced, the algorithm continues the traversal decreasing the cone bound at each iteration. When the cone has been completely rewound, the algorithm terminates returning UNDEF together with the step and cone bound at which either a spurious counterexample was found (lines 15–16) or rewinding has been forced (lines 26–27).

The threshold D heuristically controls activation of cone rewinding. Whenever $D = 0$, rewinding is always active, so the approach obtains a minimal refinement, and mimics the effect of interpolation sequences. High values of D keep the k value constant until a hit, a scheme much closer to standard interpolation. We empirically observed that small values are better at small sequential depths, as they can produce more refinement steps of limited cost.

Algorithm 7. *Unwinding procedure of IGR.*

Input: $\mathcal{S} = \langle X, \mathcal{I}, T \rangle$ a transition system; P a property over X; i_{hit} the step at which the cone was hit during traversal; k_{hit} the depth of the cone hit.

Output: $\langle res, cex, i, k \rangle$ with $res \in \{\text{UNDEF}, \text{REACH}\}$; cex a (possibly empty) initial path representing a counterexample; i the step at which resume forward traversal; k the depth of the cone to use in the resumed traversal.

1: **procedure** UNWIND($\mathcal{S}$, P, i_{hit}, k_{hit})
2: $\quad i \leftarrow i_{hit}$
3: $\quad k \leftarrow k_{hit}$
4: $\quad$ **while** $i \geq 0 \wedge \exists \boldsymbol{\pi}^{0,k} \models F_i(X^0) \wedge T(X^0, X^1) \wedge Cone(1, k)$ **do**
5: $\quad\quad i \leftarrow i - 1$
6: $\quad\quad k \leftarrow k + 1$
7: $\quad$ **if** $i < 0$ **then**
8: $\quad\quad$ **return** $\langle \text{REACH}, \boldsymbol{\pi}^{0,k}, -, - \rangle$
9: $\quad$ **return** $\langle \text{UNDEF}, -, i, k \rangle$

At each iteration, the UNWIND procedure, described in Algorithm 7, properly computes i and k, starting from i_{hit} and k_{hit} (related to the previous spurious counterexample). Following the strategy described in Sect. 4.1, the cone bound k is extended, and i is decremented (lines 5–6), until an UNSAT BMC check is obtained or the initial states are reached (line 4). In the first case, UNWIND has found a frame at which starting traversal is expected to refine $\mathbf{F_k}$ and filters out the last spurious counterexample found. The procedure then returns an UNDEF result together with the values i and k computed (line 9). Otherwise, the procedure has detected a real counterexample as a side effect. In that case, the procedure returns REACH along the counterexample found (line 8).

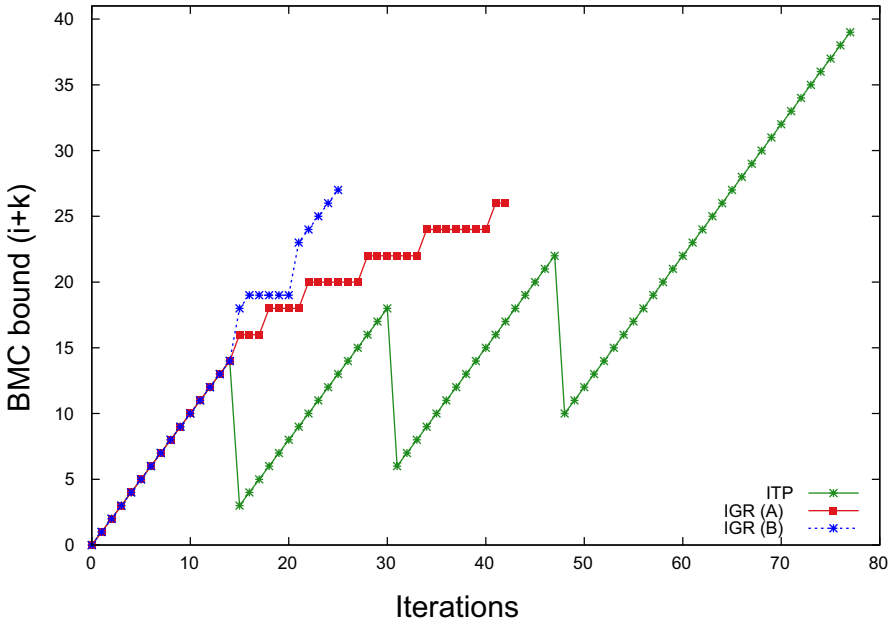


Fig. 3 BMC bound comparison for `intel015`, between standard interpolation (ITP) and IGR in two versions: **A** (rewind always enabled) and **B** (rewind disabled until hit)

5.1 IGR setup comparison: a case study on `intel015`

Figures 3 and 4 report experimental data on a case study, namely circuit `intel015` from [6], that we selected among the ones where standard interpolation could be compared with IGR. Figure 3 plots $i + k$, the sum of state set indexes (i) and cone bounds (k). This is usually logged as an equivalent BMC bound. Iterations (on the x axis) indicate algorithm iterations (with image computation). The standard interpolation line clearly shows that BMC bounds grow linearly within ApproxFwdTrav, and they restart from the newly adjusted cone bound[2] at new ApproxFwdTrav calls. The IGR (A) line plots a run of IGR, with cone rewinding always enabled: this means that the iterative decrease of k compensates the increase of i, keeping the BMC bound constant within IgrApproxFwdTrav (except when we reach $k = 0$). The IGR (B) line plots a run of IGR, with cone rewinding disabled until a BMC hit. In this case we observe an initial increase of BMC bounds, followed by a phase with constant BMC bound. Overall, IGR exploits its ability to avoid restarting from low bounds and seeking for optimal restarts, which can provide convergence at lower iteration indexes.

A comparison between IGR (A) and (B) shows that the latter can converge in fewer iterations, due to its ability to increase BMC bounds. However Fig. 4, that plots cumulative CPU times, shows that IGR (A) can be faster.

[2] Following [8], we heuristically increment cone bounds by more than 1, based on the depth of the previous ApproxFwdTrav run.

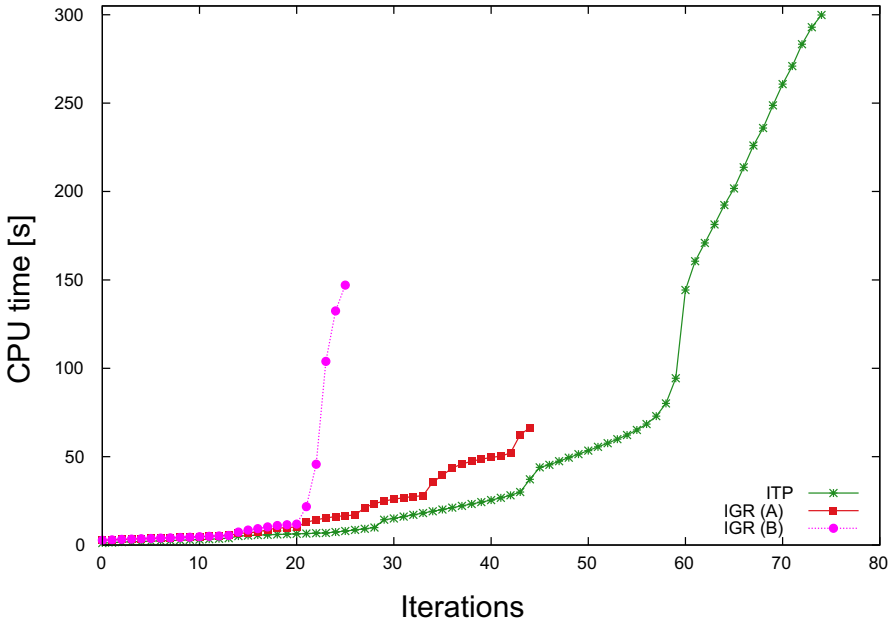


Fig. 4 CPU time comparison for `intel015`, between standard interpolation (ITP) and IGR in two versions: **A** (rewind always enabled) and **B** (rewind disabled until hit)

Intuitively, guided and simplified cones in IGR can generate cheaper BMC problems, as compared to standard interpolation. IGR (A) benefits from triggering more, but possibly simpler, refinement steps and hence, SAT calls. Although this is a good way to avoid highly expensive BMC problems, IGR (B) often performs better in case of models with higher diameters (e.g., in the range of hundredths).

6 Lazy abstraction and multiple properties

Due to the fully incremental representation chosen for reachable states, IGR can be tightly embedded in lazy abstraction, as well as multiple property verification loops.

Typically, abstraction-refinement approaches, such as [27] and [28], iterate incremental model refinements, solving a model checking problem after each refinement step. More recently [29] explores a tighter integration between a model checking algorithm (IC3) and a lazy abstraction framework. As IGR is based on a similar trace data structure, it can be easily integrated within a lazy abstraction framework. Frames can be inherited by all refined models. Refinements on frames can be considered as model strengthening steps. Let $\mathcal{S}^j$ and $\mathcal{S}^{j+1}$ be two abstract models (after refinement steps j and $j+1$). Let $\mathcal{R}_i^E(\mathcal{S}^j)$ and $\mathcal{R}_i^E(\mathcal{S}^{j+1})$ be the states reachable by them in i steps. As refinement of frames strengthens a model, $\mathcal{R}_i^E(\mathcal{S}^{j+1}) \subseteq \mathcal{R}_i^E(\mathcal{S}^j)$, so state set overapproximations for $\mathcal{S}^j$ also overapproximate states in $\mathcal{S}^{j+1}$.

A similar framework can be adopted in multiple property verification, where frames can be inherited and reused by all properties under check on the same model. Reusability of state sets is guaranteed here by sharing the same model over different property checks.

7 Experimental results

We implemented a prototype version of our methodology on top of the PdTRAV tool [30], a state-of-the-art model checking academic tool.

The experimental data in this section provide an evaluation of the techniques proposed, as well as a comparison with standard interpolation.

We aim at showing that IGR can improve over standard interpolation, and be an effective part of a model checking portfolio, by covering problems not completed by other engines or by simply improving performance over them. We thus do not provide an extensive set of results, with all available options, over a complete set of benchmarks. Instead, we focused most of our efforts on a set of experiments that we deemed "challenging", so adequate to our purpose.

We performed an extensive experimentation on a selected sub-set of publicly available benchmarks from past Hardware Model Checking Competitions (HWMCCs) [6] suites. Most of the benchmarks are from the last bit-level competition (HWMCC2017) with some addition of challenging benchmarks from previous editions, that were excluded from HWMCC2017, due to a benchmark selection strategy that obviously avoided full inclusion of past edition benchmarks.

It is worth noticing that most of the selected benchmarks are from industrial origin (IBM, Intel).

Benchmarks were selected by focusing on two groups:

- Benchmarks completed by at least one tool at HWMCC competitions that we were able to solve with IGR (in more than 60 seconds). This set of benchmarks was used to provide a comparison between IGR and other tools. As we didn't replicate the experiments with other tools, such a comparison is based on available data from HWMCC competitions.
- Benchmarks left unsolved at HWMCC competitions that we were able to solve with IGR. This set of benchmarks was used to compare IGR to standard interpolation over challenging problems. Though HWMCC experiments were run under a 1 hour time limit, we used a much higher time limit in our experiments. Note that we are not claiming that no other tool could cover these benchmarks, but simply that they can be deemed as challenging (some of them are indeed very challenging) and thus worth further investigation. Our results show that IGR improves over standard interpolation on such benchmarks.

Our experiments were run on a quad-core workstation, with 2.5 GHz CPU frequency and 32 GB of main memory. We ran the proposed set of experiments taking into account different setups, as detailed below.

Table 1 reports details about the selected benchmarks in terms of model name, number of primary inputs, latches and AIG nodes. Table 2 compares the best results we could obtain using IGR to the best ones obtained using standard interpolation and to the best ones achieved by any contestant of past HWMCCs.

Table 1 Circuit details of the selected HWMCC benchmarks

Model			
Name	#PI	#FF	#AIG
6s8	86	396	3016
6s38	343	1931	10847
6s102	72	1108	7700
6s144	480	3337	45470
6s189	479	2436	39830
6s194	532	2131	13617
6s428rb093	410	3790	29084
intel010	1111	280	10156
intel011	1024	273	9362
intel015	1024	273	9362
6s35	77	1571	11504
6s148	480	3337	45470
6s160	149	559	8716
6s195	87	1257	8046
6s171	94	1263	8073
6s366r	86	1998	20560
oc8051gm06iram	364	934	12067
oc8051gm3bacc	364	934	12055
oc8051gm49acc	364	934	11990
oc8051gm88iram	364	934	12761
intel028	7426	7436	99835

Tables are split into three parts. Benchmarks solved by other tools in HWMCC competitions are listed in the first two parts, differentiating between IBM and Intel benchmarks. The last section shows problems unsolved at HWMCC competitions, this is thus the most interesting contribution of IGR: the section shows 6 IBM benchmarks and 2 benchmarks derived from an OpenCores.org implementation of the 8051 Intel micro-controller [31]. Though we were able to verify other unsolved 8051 benchmarks, we limited them to two instances.

In Table 2, for IGR we report the maximum BMC bound depth reached (Column MaxB), the bound at which a fixed point has been found during forward traversal (Column Fwd), the last restart forward iteration, i.e., the bound at the which the forward traversal has been restarted last as a result of cone unwinding, (Column Last) and the verification time (Column Time). In addition we report whether or not lazy abstraction (Column LA) or phase abstraction [32] (Column PA) were used to verify the benchmark. For ITP we just report the maximum bound depth reached (Column MaxB), the fixed point forward iteration (Column Fwd) and the verification time (Column Time). Finally, we provide the best results obtained during past HWMCCs in terms of number of solvers that were able to verify the benchmark (Column Solvers) and verification time (Column Time). To this respect, it is worth noticing that time statistics from competitions were measured on a different machine, by portfolio based (concurrent) model checkers. The comparison with other engines is not as easy. To this respect it is worth noticing that the best model checkers at HWMCCs highly rely on aggressive transformational techniques, that seek to pre-simplify problems under various equivalence-preserving notions, before getting to Model Checking engines.

Table 2 Comparison of results of IGR vs. ITP on selected HWMCC benchmarks

	IGR						ITP			HWMCC	
Model	MaxB	Fwd	Last	Time	LA	PA	MaxB	Fwd	Time	Solvers	Time
6s8	48	21	2	835.4	No	No	45	21	835.4	4	147.82
6s38	27	18	1	264.23	Yes	No	25	17	310.71	2	606.89
6s102	23	7	1	488.47	Yes	No	31	20	726.62	8	10.58
6s144	36	22	1	291.48	No	Yes	40	22	160.62	6	155.98
6s189	30	16	1	96.46	No	Yes	37	26	282.66	3	110.48
6s194	30	16	1	154.45	No	No	40	29	852.17	7	54.38
6s428rb093	7	6	2	746.75	No	No	–	–	–	2	273.34
intel010	56	36	14	200.91	No	Yes	65	44	265.7	3	96.37
intel011	56	36	20	190.73	No	Yes	64	42	899.89	4	440.09
intel015	40	22	12	130.3	No	Yes	–	–	–	3	272.22
6s35	83	71	70	428.88	No	No	81	71	2704.61	0	–
6s148	29	16	1	254.13	No	Yes	39	27	288.94	0	–
6s160-f4	56	37	1	61573.25	No	No	–	–	MEMOUT	0	–
6s195-f8	139	70	1	11447.65	No	No	39706.02	140	71	0	–
6s171	570	261	279	302129.05	Yes	No	–	–	MEMOUT	0	–
6s366r	79	69	66	226.73	No	No 83	72	1811.43	0	–	
oc8051gm06iram	38	26	15	6125.45	No	No	49	36	17084.02	0	–
oc8051gm3bacc	33	22	14	816.28	No	No	33	22	1128.29	0	–
oc8051gm49acc	33	20	14	867.18	No	No	44	23	1318.41	0	–
oc8051gm88iram	44	33	16	14409.62	No	No	42	31	14796.59	0	–
intel028	166	74	16	9855.81	No	Yes	–	–	MEMOUT	0	–

Data clearly show that, in some cases, interpolation based approaches are more expensive than HWMCC best results: depending on individual instances, this could be due to other engines in portfolios (e.g. IC3) and/or netlist preprocessing/simplification steps. Overall, the performance of IGR and ITP are not far from the HWMCC best. A comparison between IGR and ITP generally confirms the higher ability of IGR to converge at lower BMC bounds, as seen in columns labelled MaxB and Fwd.

A different outcome can be derived by experiments in the last section, that shows the power of IGR in terms of scalability. The cases solved by both IGR and ITP confirm the previous comments; differences in execution times can generally be attributed to the ability of IGR to skip intermediate bounds and/or restarts from the initial state. The benchmarks only solved by IGR, or solved with relevant gains (6*s*160, 6*s*195, 6*s*171, *intel*028) are more related to the ability to reuse previous interpolants, which can be used for simplification purposes and or as additional constraints in solver runs. It is worth noticing that 6*s*160 and 6*s*195 were preprocessed by framing (which means composing a given number of transition relation frames into one).

It is also worth noticing that lazy abstraction helped improving performance in 2 of the listed cases. Lazy abstraction was not implemented with standard interpolation, where we just rely on CEGAR-based abstraction, as reuse of information from precious abstraction would be limited, whereas IGR fully reuses previous interpolants.

An updated version of the results presented in the last section of Table 2 (most challenging benchmarks), as well as a copy of the pdtrav binary used to perform the experiments is available at [33].

In order to gather more data, we did a second experimental evaluation of IGR, extended to the single property benchmarks set of the latest HWMCC, purged of the instances solved by any engine in less than a minute. We repeated a "competition run" with our multi-engine portfolio in three different setups:

- Enabling all the available engines within PdTRAV, thus including standard interpolation techniques, IC3, IGR, BMC and reachability-based techniques;
- Excluding IC3;
- Excluding both IC3 and IGR.

The results are plotted in Fig. 5, which clearly confirms IC3 as the most powerful engine. But it also shows a good impact of IGR, as a relevant contribution to the portfolio. The run with the full set of engines solved 116 problems, of which 47 were covered by IC3, and 10 by IGR. When disabling *IC*3, the overall result decreased to 81, with IGR solving 18 problems. Data also show that IGR is still not oriented to fast runs (within minutes). As seen in Table 2, a 2 hours timeout better shows the gain of IGR over ITP.

8 Related works

Our work is related to many recent papers on SAT-based Model Checking. Among the others, let us mention ITPSEQ [23], DAR [25] and AVY [34]. Our approach shares with them the purpose to push forward scalability and performance of interpolation-based model checking and the idea of incremental refinement of the computed overapproximations.

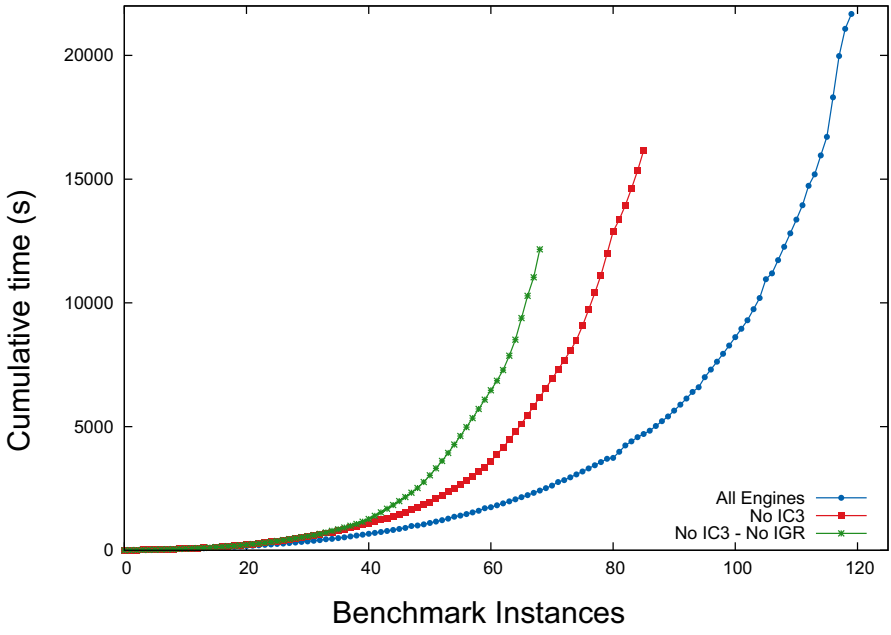


Fig. 5 Wall clock cumulative time comparison on HWMCC instances solved by PdTRAV (concurrent multi-engine version), with all engines active, without IC3, and without IC3 and IGR. Time limit 900 s per instance

Our approach takes ideas from all above works, it is based on interpolation, it computes just forward approximations of state sets, which allows us to potentially reuse them for multiple properties, or sub-properties, of the same model.

The use of a trace data structure is partially inspired by IC3 [5], DAR and ITPSEQ. Compared to IC3, the proposed approach relies on transition relation unrollings, instead of local reachability checks, in order to increase the precision of the computed over-approximations. In addition, contrary to IC3, IGR stores overapproximations as AIG circuits instead of sets of clauses and does not enforce monotonicity of the sequence of reachable states sets it derives. Compared to other interpolation-based approaches, the proposed method keeps the standard interpolation scheme of ITP, without considering backward reachable overapproximations like DAR. Contrary to standard interpolation, in which the trace data structure can be thought as implicitly computed but not explicitly maintained, in the proposed approach the trace is explicitly represented and iteratively refined. Compared to interpolation sequences we never compute an interpolation sequence from a single SAT run and proof, but we activate sequences of standard interpolation and/or approximate image calls.

Cone rewinding is similar to computing an interpolation sequence [24]. Some user-controllable parameters in the proposed algorithm can be configured to make the procedure mimic either the ITPSEQ or the standard ITP algorithm.

Simplification under observability don't cares is an extensively studied subject in the field of logic synthesis [35]. The proposed algorithm makes use of some very

light-weight simplification techniques in order to remove redundancies from the computed approximations and transition relation unrollings from the target.

Many other internal details, at the level of SAT and circuit-based reasoning, take inspiration from the above, as well as other existing works. Let us mention for instance clause propagation by *pushing*, redundancy removal by subsumption, that we brought from IC3 and re-implemented on circuit-based (AIG) representations.

9 Conclusions

We addressed the problem of optimizing interpolant-based verification techniques for SAT-based Unbounded Model Checking. Our main contribution is to provide a new approach, that improves over standard interpolation, by exploiting the ideas of incremental refinement and guidance through state sets. We experimentally observed that the proposed optimizations have improved both performance and scalability of our existing UMC approaches.

References

1. Cabodi G, Palena M, Pasini P (2014) Interpolation with guided refinement: Revisiting incrementality in sat-based unbounded model checking, In: Proceedings of the 14th conference on formal methods in computer-aided design, ser. FMCAD '14. Austin, TX: FMCAD Inc, pp. 12:43–12:50. [Online]. Available: http://dl.acm.org/citation.cfm?id=2682923.2682938
2. Craig W (1957) Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. J Symbol Logic 22(3):269–285
3. Lyndon RC (1959) An interpolation theorem in the predicate calculus. Pacific J Math 9(1):155–164
4. McMillan KL (2003) Interpolation and SAT-based model checking, In: Proceedings computer aided verification, ser. LNCS, vol. 2725. Boulder, CO, USA: Springer, pp. 1–13
5. Bradley AR (2011) Sat-based model checking without unrolling, In: VMCAI, Austin, Texas, Jan. 2011, pp. 70–87
6. Biere A, Jussila T The model checking competition web page, http://fmv.jku.at/hwmcc
7. McMillan KL, Jhala R (2005) Interpolation and SAT-based model checking, In: Proceedings computer aided verification, ser. LNCS, vol. 3725. Edinburgh, Scotland, UK: Springer, pp. 39–51
8. Marques-Silva J (2005) Improvements to the implementation of Interpolant–based model checking, In: Proceedings correct hardware design and verification methods, ser. LNCS, vol. 3725. Edinburgh, Scotland, UK: Springer, pp. 367–370
9. D'Silva V, Purandare M, Kroening D (2008) Approximation refinement for interpolation-based model checking, in verification, model checking and abstract interpretation, ser. Lecture Notes in Computer Science, vol. 4905. Springer, pp. 68–82
10. Cabodi G, Murciano M, Nocco S, Quer S (2008) Boosting interpolation with dynamic localized abstraction and redundancy removal. ACM Trans Design Autom Electr Syst 13(1):309–340
11. Cabodi G, Camurati P, Murciano M (2008) Automated abstraction by incremental refinement in interpolant-based model checking, In: Proceedings international conference on computer-aided design. San Jose, California: ACM Press, Nov. pp. 129–136
12. D'Silva V, Kroening D, Purandare M, Weissenbacher G (2010) Interpolant strength. In: Proceedings of the 11th international conference on verification, model checking, and abstract interpretation, ser. VMCAI'10. Berlin, Heidelberg: Springer-Verlag, p. 129–145. [Online]. Available: https://doi.org/10.1007/978-3-642-11319-2_12
13. Li B, Somenzi F (2006) Efficient abstraction refinement in interpolation-based unbounded model checking, In: Tools and algorithms for the construction and analysis of systems, vol. 3920, pp. 227–241
14. Cabodi G, Loiacono C, Vendraminetto D (2013) Optimization techniques for Craig interpolant compaction in unbounded model checking, In: Proceedings design automation & test in Europe conference Grenoble, France: IEEE Computer Society, Mar. pp. 1417–1422

15. Cabodi G, Loiacono C, Vendraminetto D (2015) Optimization techniques for Craig interpolant compaction in unbounded model checking. Form Methods Syst Des 46(2):135–162. https://doi.org/10.1007/s10703-015-0229-0
16. Cabodi G, Camurati PE, Palena M, Pasini P, Vendraminetto D (2016) Reducing interpolant circuit size by ad-hoc logic synthesis and sat-based weakening. In: Proceedings of the 16th conference on formal methods in computer-aided design, ser. FMCAD '16. Austin, TX: FMCAD Inc, pp. 25–32. [Online]. Available: http://dl.acm.org/citation.cfm?id=3077629.3077640
17. Goldberg E, Güdemann M, Kroening D, Mukherjee R (2018) Efficient verification of multi-property designs (the benefit of wrong assumptions), In: 2018 Design, automation test in Europe Conference Exhibition (DATE), pp. 43–48
18. Clarke EM, Grumberg O, Jha S, Lu Y, Veith H (2000) Counterexample-guided abstraction refinement, In: CAV, pp. 154–169
19. Gupta A, Ganai M, Yang Z, Ashar P (2003) Iterative abstraction using SAT-based BMC with proof analysis, In: Proceedings international conference on computer-aided design, San Jose, California, Nov. pp. 416–423
20. Moskewicz M, Madigan C, Zhao Y, Zhang L, Malik S (2001) Chaff: Engineering an efficient SAT solver, In: Proceedings 38th design automation Conference Las Vegas, Nevada: IEEE Computer Society, Jun
21. Eén N, Sörensson N (2009) The Minisat SAT solver, http://minisat.se, Apr
22. Biere A, Cimatti A, Clarke EM, Fujita M, Zhu Y (1999) Symbolic model checking using SAT procedures instead of BDDs, In: Proceedings 36th design automation conference. New Orleans, Louisiana: IEEE Computer Society, Jun. pp. 317–320
23. Vizel Y, Grumberg O (2009) Interpolation-sequence based model checking. In: Proceedings formal methods in computer-aided design, ser. LNCS, vol. 2517. Austin, Texas, USA: Springer, Nov. pp. 1–8
24. Cabodi G, Nocco S, Quer S (2011) Interpolation sequences revisited. In: Proceedings design automation & test in Europe conference Grenoble, France: IEEE Computer Society, Mar. pp. 316–322
25. Vizel Y, Grumberg O, Shoham S (2013) Intertwined forward-backward reachability analysis using interpolants, In: Tools and algorithms for the construction and analysis of systems, ser. LNCS, vol. 7795. Rome, Italy: Springer, Mar. pp. 308–323
26. Mishchenko A, Brayton RK (2005) SAT-Based complete Don't-Care computation for network optimization, In: Proceedings design automation & test in Europe conferenece, pp. 412–417
27. Clarke E, Grumberg O, Jha S, Lu Y, Veith H (2003) Counterexample-guided abstraction refinement for symbolic model checking. J ACM 50(5):752–794. https://doi.org/10.1145/876638.876643
28. Gupta A, Strichman O (2005) Abstraction refinement for bounded model checking. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 112–124. [Online]. Available: https://doi.org/10.1007/11513988_11
29. Vizel Y, Grumberg SSO (2012) , Lazy abstraction and SAT-Based reachability in hardware model checking, In: Proceedings formal methods in computer-aided design. Cambridge, UK: IEEE, Oct. pp. 173–181
30. Cabodi G, Nocco S, Quer S (2011) Benchmarking a model checker for algorithmic improvements and tuning for performance. Formal Methods Syst Design 39(2):205–227
31. Subramanyan P, Vizel Y, Ray S, Malik S (2015) Template-based synthesis of instruction-level abstractions for SOC verification, In: 2015 Formal methods in computer-aided design (FMCAD), pp. 160–167
32. Baumgartner J, Aziz A (2003) An abstraction algorithm for the verification of level-sensitive latch-based netlists, Formal Methods in System Design, vol. 23, pp. 39–65, 07
33. Cabodi G, Camurati P, Palena M, Pasini P" (2021) , Igr - experiments, https://github.com/P3900/igr-exp
34. Vizel Y, Gurfinkel A (2014) Interpolating property directed reachability, In: Proceedings of the 16th international conference on computer aided verification - Vol. 8559. New York, NY, USA: Springer-Verlag New York, Inc., pp. 260–276. [Online]. Available: https://doi.org/10.1007/978-3-319-08867-9_17
35. Mishchenko A, Brayton R, Jiang J-HR, Jang S (2011) Scalable don't-care-based logic optimization and resynthesis. ACM Trans Reconfigurable Technol Syst 4(4):1–23. https://doi.org/10.1145/2068716.2068720