SAT-based Model Checking: Interpolation, IC3, and Beyond

Orna GRUMBERG [a], Sharon SHOHAM [b] and Yakir VIZEL [a]

[a] *Computer Science Department, Technion, Haifa, Israel*

[b] *School of Computer Science, Academic College of Tel Aviv-Yaffo*

Abstract.

SAT-based model checking is currently one of the most successful approaches to checking very large systems. In its early days, SAT-based (bounded) model checking was mainly used for bug hunting. The introduction of *interpolation* and *IC3\PDR* enable efficient complete algorithms that can provide full verification as well.

In this paper, we survey several approaches to enhancing SAT-based model checking. They are all based on iteratively computing an *over-approximation* of the set of reachable system states. They use different mechanisms to achieve scalability and faster convergence.

The first one uses *interpolation sequence* rather than interpolation in order to obtain a more precise over-approximation of the set of reachable states. The other approach integrates lazy abstraction with IC3 in order to achieve scalability. *Lazy abstraction*, originally developed for software model checking, is a specific type of abstraction that allows hiding different model details at different steps of the verification. We find the IC3 algorithm most suitable for lazy abstraction since its state traversal is performed by means of *local* reachability checks, each involving only two consecutive sets. A different abstraction can therefore be applied in each of the local checks.

The survey focuses on hardware model checking, but the presented ideas can be extended to other systems as well.

Keywords. Model Checking, SAT-based Model Checking, Interpolation, Interpolation Sequence, Bounded Model Checking (BMC), IC3, Unbounded Model Checking, Abstraction, Lazy abstraction, Hardware model checking

1. Introduction

Computerized systems dominate almost every aspect of our lives and their correct behavior is essential. *Model checking* [8, 10, 28] is an automated verification technique for checking whether a given system satisfies a desired property. The system is usually described as a finite-state model in a form of a state transition graph. The specification is given as a temporal logic formula. Unlike testing or simulation based verification, model checking tools are exhaustive in the sense that they traverse *all* behaviors of the system, and either confirm that the system behaves correctly or present a *counterexample*.

Model checking has been successfully applied to verifying hardware and software systems. Its main limitation, however, is the *state explosion problem* which arises due to the huge state space of real-life systems. The size of the model induces high memory and time requirements that may make model checking not applicable to large systems. Much of the research in this area is dedicated to increase model checking applicability and scalability.

The first significant step in this direction was the introduction of BDDs [4] into model checking. BDD-based *Symbolic Model Checking* (SMC) [5] enabled model checking of real-life hardware designs with a few hundreds of state elements. However, current design blocks with well-defined functionality typically have thousands of state elements and more. To handle designs of that scale, SAT-based *Bounded Model Checking* (BMC) [2] has been developed. Its main drawback, however, is its orientation towards "bug-hunting" rather than full verification.

Several approaches have been suggested to remedy the problem. *Induction* [29], *interpolation* [22], *interpolation sequence* [6,32], *IC3/PDR* [3,12], and *L-IC3* [31] developed different techniques for SAT-based *Unbounded Model Checking* (UMC), which provide full verification.

Of these SAT-based unbounded model checking techniques, L-IC3 and [6] also use *Abstraction-refinement* [9], which is another well known methodology for tackling the state-explosion problem. Abstraction hides model details that are not relevant for the checked property. The resulting abstract model is then smaller, and therefore easier to handle by model checking algorithms. *Lazy abstraction* [15, 23], developed for software model checking, is a specific type of abstraction that allows hiding different model details at different steps of the verification.

In this paper we discuss four of the above mentioned SAT-based approaches for full verification. These methods all compute an over-approximated set of the system's reachable states while checking that the specification is not violated. The first two approaches we describe are taken from [22] and [32]. They combine BMC with *interpolation* [11] or *interpolation-sequence* [16,23] respectively. These methods use an unrolling of the model's transition relation in order to traverse the system's state space. The third algorithm we discuss is IC3/PDR [3, 12]. In contrast to the first two methods, IC3 avoids unrolling of the transition relation. Instead, the computation of reachable states uses *local reachability checks* between consecutive time frames. Last, we present L-IC3 [31] which provides a SAT-based *lazy* abstraction-refinement algorithm based on IC3/PDR. L-IC3 uses the *visible variables abstraction* [18], which is particularly suitable for hardware systems. However, the abstraction is used in a lazy manner in the sense that different sets of visible variables are used in different iterations of the state-space traversal.

The rest of the paper is organized as follows. Section 2 defines some basic notions. Section 3 presents the Bounded Model Checking (BMC) algorithm. Section 4 defines interpolation and interpolation sequence and discusses their computation. Sections 5 and 6 describe how interpolation sequence and interpolation can be used in model checking, and Section 7 compares between the two methods. Section 8 gives an overview of the IC3 algorithm. Section 9 presents the visible (lazy) abstraction. L-IC3 is described in Section 10. We conclude in Section 11.

2. Preliminaries

Temporal logic model checking [10] is an automatic approach to formally verifying that a given system satisfies a given specification. The system is often modelled by a finite state transition system and the specification is written in a *temporal logic*. Determining whether a model satisfies a given specification is often based on an exploration of the model's state space in a search for violations of the specification.

In this survey we focus on hardware. As such we consider finite state transition systems defined over Boolean variables, as follows.

Definition 2.1. A *finite state transition system* (a model) is a tuple $M = (V, U, \mathit{INIT}, \mathit{TR})$ where V is a set of Boolean variables, $U \subseteq V$ is a set of state variables, $V \setminus U$ is a set of input variables, $\mathit{INIT}(V)$ is a propositional formula over V describing the initial states, and $\mathit{TR}(V, V')$ describes a total transition relation which is defined as a propositional formula over V and the next-state variables $V' = \{v' \mid v \in V\}$.

The transition relation is described using next-state functions for each state variable. Namely, $\mathit{TR}(V, V') = \bigwedge\limits_{v \in U} (v' = f_v(V, V'))$ where $f_v(V, V')$ is a propositional formula that assigns the next value to $v \in U$ based on current and next-state variables. Note that for an input variable $v \in V \setminus U$, f_v is not defined.

The set of Boolean variables of M induces a set of states $S = \{0, 1\}^{|V|}$, where each state $s \in S$ is given by a valuation of the variables in V. A formula over V (resp. V, V') represents the set of states (resp. pairs of states) obtained by its satisfying assignments. With abuse of notation we will refer to a formula η over V as a set of states and therefore use the notion $s \in \eta$ for states represented by η. Similarly for a formula η over V, V', we will sometimes write $(s, s') \in \eta$.

The formula $\eta[V \leftarrow V']$, or η' in short, is identical to η except that each variable $v \in V$ is replaced with v'. In the general case V^i is used to denote the variables in V after i time units (thus, $V^0 \equiv V$). Let η be a formula over V^i, the formula $\eta[V^i \leftarrow V^j]$ is identical to η except that for each variable $v \in V$, v^i is replaced with v^j.

A *path* in M is a sequence of states $\pi = s_0, s_1, \ldots$ such that for all $i \geq 0$, $s_i \in S$ and $(s_i, s_{i+1}) \in \mathit{TR}$. The length of a path is denoted by $|\pi|$. If π is infinite then $|\pi| = \infty$. If $\pi = s_0, s_1, \ldots, s_n$ then $|\pi| = n$. A path is an *initial path* when $s_0 \in \mathit{INIT}$. We sometimes refer to a prefix of a path as a path as well.

A formula in *Linear Temporal Logic* (LTL) [10,27] is of the form $\mathbf{A}f$ where f is a path formula. A model M satisfies an LTL property $\mathbf{A}f$ if all infinite initial paths in M satisfy f. If there exists an infinite initial path not satisfying f, this path is defined to be a *counterexample*.

In this paper we consider a subset of LTL formulas of the form $\mathbf{AG}\,p$, where p is a propositional formula. $\mathbf{AG}\,p$ is true in a model M if along every initial infinite path all states satisfy the proposition p. In other words, all states in M that are reachable from an initial state satisfy p. This does not restrict the generality of the suggested methods since model checking of liveness properties can be reduced to handling safety properties [1]. Further, model checking of safety properties can be reduced to handling properties of the form $\mathbf{AG}\,p$ [17].

The *model checking problem* is the problem of determining whether a given model satisfies a given property. For properties of the form $\mathbf{AG}\, p$ this can be done based on the set of states reachable from the initial states, called *reachable states* in short. Let M be a model, *Reach* be the set of reachable states in M, and $f = \mathbf{AG}\, p$ be a property. If for every $s \in \mathit{Reach}$, $s \models p$ then the property holds in M. On the other hand, if there exists a state $s \in \mathit{Reach}$ such that $s \models \neg p$ then there exists an initial path $\pi = s_0, s_1, \ldots, s_n$ such that $s_n = s$. The path π is a *counterexample* for the property f.

Model checking has been successfully applied in hardware verification, and is emerging as an industrial standard tool for hardware design . The main technical challenge in model checking, however, is the *state explosion* problem which occurs if the system is a composition of several components or if the system variables range over large domains.

Notation Throughout the paper we denote the value *false* as $\perp$ and the value *true* as $\top$. For a propositional formula η we use $\mathrm{Vars}(\eta)$ to denote the set of all variables appearing in η. For a set of formulas $\{\eta_1, \ldots, \eta_n\}$ we will use $\mathrm{Vars}(\eta_1, \ldots, \eta_n)$ to denote the variables appearing in $\eta_1, \ldots, \eta_n$. That is, $\mathrm{Vars}(\eta_1, \ldots, \eta_n) = \mathrm{Vars}(\eta_1) \cup \ldots \cup \mathrm{Vars}(\eta_n)$.

3. Bounded Model Checking

Many problems, including some versions of model checking, can naturally be translated into the *satisfiability* problem of the propositional calculus. The satisfiability problem is known to be NP-complete. Nevertheless, modern SAT-solvers, developed in recent years, can check satisfiability of formulas with several thousands of variables within a few seconds. SAT-solvers such as Grasp [19], Prover [30], Chaff [26], MiniSAT [13], and many others, are based on sophisticated learning techniques and data structures that accelerate the search for a satisfying assignment, if exists.

A SAT-solver is a complete decision procedure that given a propositional formula, determines whether the formula is *satisfiable* or *unsatisfiable*. Most SAT-solvers assume a formula in *Conjunctive Normal Form* (CNF), consisting of a conjunction of a set of clauses, each of which is a disjunction of literals, where a literal is a propositional variable or its negation. A CNF formula is satisfiable if there exists a *satisfying assignment* for which every clause in the set is evaluated to $\top$. If the clause set is satisfiable then the SAT solver returns a satisfying assignment for it. If it is not satisfiable (unsatisfiable), meaning, it has no satisfying assignment, then modern SAT solvers produce a *proof of unsatisfiability* [24, 33]. The proof of unsatisfiability has many useful applications. We will introduce one of them in the next section.

Below we describe a simple way to exploit satisfiability for bounded model checking of properties of the form $\mathbf{AG}\, p$, where p is a propositional formula.

Bounded model checking (BMC) [2] is an iterative process for checking properties of a given structure up to a given bound. Let M be a model and $f = \mathbf{AG}\, p$ be the property to be verified. Given a bound k, BMC either finds a counterexample of length k or less for f in M, or concludes that there is no such counterexample.

```
function BMC(M,f,k)
    i := 0
    while i ≤ k do
        build φ^i_M(f)
        result = SAT(φ^i_M(f))
        if result = true then
            return cex      // returning the counterexample
        else
            i = i + 1
        end if
    end while
    return No cex for bound k
end function
```

Figure 1. Bounded model checking

In order to search for a counterexample of length k the following propositional formula is built:

Formula 1. $\varphi_M^k(f) = INIT(V^0) \wedge TR(V^0, V^1) \wedge TR(V^1, V^2) \wedge \ldots \wedge TR(V^{k-1}, V^k) \wedge (\neg p(V^k))$

$\varphi_M^k(f)$ is then passed to a SAT solver which searches for a satisfying assignment. If there exists a satisfying assignment for $\varphi_M^k(f)$ then the property $\mathbf{AG}\,p$ is violated, since there exists a path of M of length k violating the property. In order to conclude that there is no counterexample of length k or less, BMC iterates all lengths from 0 up to the given bound k. At each iteration a SAT procedure is invoked.

When M and f are obvious from the context we omit them from the formula $\varphi_M^k(f)$ denoting it as φ^k. The BMC algorithm is described in Figure 1.

The main drawback of this approach is its incompleteness. It can only guarantee that there is no counterexample of size smaller or equal to k. It cannot guarantee that there is no counterexample of size greater than k.

Thus, this method is mainly suitable for refutation. Verification is obtained only if the bound k exceeds the length of the longest path among all shortest paths from an initial state to some state in M. In practice, it is hard to compute this bound and even when known, it is often too large to handle. Several methods for full verification with SAT have been suggested, such as induction [29], ALL-SAT [7, 14, 21], interpolation [20, 22, 32], and Property Directed Reachability (PDR/IC3) [3, 12, 31]. In the rest of the paper we will focus on SAT-based verification with interpolation and PDR.

4. Interpolation

In this section we introduce two notions, *interpolation* [11] and *interpolation-sequence* [16] that, when combined with BMC, can provide full program verification.

Definition 4.1. Let (A, B) be a pair of formulas such that $A \wedge B \equiv \bot$. The *interpolant* for (A, B) is a formula I such that:

- $A \Rightarrow I$.
- $I \wedge B \equiv \bot$.
- $\text{Vars}(I) \subseteq \text{Vars}(A) \cap \text{Vars}(B)$.

The interpolant can be viewed as the part of A that is sufficient to contradict B. As mentioned above, modern SAT solvers produce a *proof of unsatisfiability* if the checked formula is unsatisfiable. An interpolant can be extracted from a proof of unsatisfiability [22], where different proofs yield different interpolants.

A similar notion can be defined when we have a sequence of formulas whose conjunction is unsatisfiable.

Definition 4.2. Let $\Gamma = \langle A_1, A_2, \ldots, A_n \rangle$ be a sequence of formulas such that $\bigwedge \Gamma \equiv \bot$. That is $\bigwedge \Gamma = A_1 \wedge \ldots \wedge A_n$ is unsatisfiable. An *interpolation-sequence* for Γ is a sequence $\langle \mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_n \rangle$ such that:

1. $\mathcal{I}_0 \equiv \top$ and $\mathcal{I}_n \equiv \bot$
2. For every $0 \leq j < n$ it holds that $\mathcal{I}_j \wedge A_{j+1} \Rightarrow \mathcal{I}_{j+1}$
3. For every $0 < j < n$ it holds that $\text{Vars}(\mathcal{I}_j) \subseteq \text{Vars}(A_1, \ldots, A_j) \cap \text{Vars}(A_{j+1}, \ldots, A_n)$

Computing an interpolation-sequence for a sequence of formulas is done in the following way: given a proof of unsatisfiability Π, for each $\mathcal{I}_i$, $0 < i < n$, the sequence of formulas is partitioned in a different way such that $\mathcal{I}_i$ is the interpolant for the formulas $A(i) = \bigwedge_{j=1}^{i} A_j$ and $B(i) = \bigwedge_{j=i+1}^{n} A_j$, obtained based on Π. In fact, all interpolants $\mathcal{I}_i$ in the sequence can be computed efficiently at once, by a single traversal of a given proof of unsatisfiability [32].

Theorem 4.3. *Let $\Gamma = \langle A_1, A_2, \ldots, A_n \rangle$ be a sequence of formulas such that $\bigwedge \Gamma \equiv \bot$ and let Π be a proof of unsatisfiability for $\bigwedge \Gamma$. For every $1 \leq i < n$ let us define $A(i) = A_1 \wedge \ldots \wedge A_i$ and $B(i) = A_{i+1} \wedge \ldots \wedge A_n$. Let $\mathcal{I}_i$ be the interpolant for the pair $(A(i), B(i))$ extracted using Π then the sequence $\langle \top, \mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_{n-1}, \bot \rangle$ is an interpolation sequence for Γ.*

5. Exploiting Interpolation-Sequence in Model Checking

In this section we present a SAT-based algorithm for full verification (sometimes also called *unbounded model checking* (UMC)), which combines BMC and interpolation-sequence [32]. BMC is used to search for counterexamples while the interpolation-sequence is used to produce over-approximated sets of reachable states and to check for termination.

Interpolation-sequence has been introduced and used in [16] and [23]. In [16] it is used for computing an abstract model based on predicate abstraction for software model checking. In [23] interpolation-sequence is used for software model checking and lazy abstraction and is applied to individual execution paths in the control flow graph. The method presented in this section exploits interpolation-sequence in a different manner. In particular, it is applied to the whole model for imitating *symbolic model checking* (SMC).

From this point and on, we will use M to denote the finite state transition system and $f = \mathbf{AG}\, p$ for a propositional formula p, as the property to be verified.

In order to better understand the algorithm and the motivation behind it, we first review some basic concepts of SMC.

5.1. Symbolic Model Checking

SMC performs *forward reachability analysis* by computing sets of reachable states S_j where j is the number of transitions needed to reach a state in S_j when starting from the initial states. More precisely, $S_0 = \mathit{INIT}$ and for every $j \geq 1$, $S_{j+1}(V') = \exists V(S_j(V) \wedge \mathit{TR}(V, V'))$. The computation of S_{j+1} is referred to as an *image* operation on the set S_j. Once S_j is computed, if it contains states violating p, a counterexample of length j is found and returned. Otherwise, if for $j \geq 1$ $S_j \subseteq \bigcup\limits_{i=0}^{j-1} S_i$ then a *fixpoint* has been reached, meaning that all reachable states have been found already. If no reachable state violates the property then the algorithm concludes that $M \models f$.

5.2. Interpolation-Sequence Based Model Checking (ISB)

The method presented in this section demonstrates how over-approximated sets, similar to S_i in their characteristics, can be extracted from BMC, based on interpolation-sequences.

As we have seen, BMC alone is only sound and not complete. In order to be able to determine if $M \models f$, current SAT-based model checking algorithms are based on a computation that over-approximates the reachable states of M. We use the notion of *Reachability Sequence*:

Definition 5.1. A *reachability sequence* (RS) of length $k + 1$ with respect to a model M and a property $\mathbf{AG}\, p$, denoted $\Omega(M, p, k)$, is a sequence $\langle F_0, \ldots, F_k \rangle$ of propositional formulas over V such that the following holds:

- $F_0 = \mathit{INIT}$
- $F_i \wedge \mathit{TR} \Rightarrow F'_{i+1}$ for $0 \leq i < k$
- $F_i \Rightarrow p$ for $0 \leq i \leq k$

A reachability sequence Ω is said to be *monotonic* (MRS) when $F_i \Rightarrow F_{i+1}$ for $0 \leq i < k$.

Recall that the formula F'_{i+1} is equivalent to $F_{i+1}[V \leftarrow V']$, and that implication between formulas corresponds to inclusion between the set of states represented by the formulas. Thus, for non-monotonic reachability sequence, the set of states represented by F_i over-approximates the states reachable from INIT in exactly i steps. When Ω is monotonic F_i represents all the states that are reachable from INIT in *at most* i steps. We refer to i as *time frame (or frame) i*. When M, p and k are clear from the context we omit them and write Ω.

Informally, we will use the notion of *fixpoint* when we can conclude that all *reachable* states in the model have been visited[1]. Using a RS enables us to determine wether a fixpoint has been reached or not.

We now show how we use BMC and interpolation-sequence to compute a RS. Note that, an interpolation-sequence exists for a bound N only when the BMC formula φ^N is unsatisfiable, i.e. when there is no counterexample of length N. In case a counterexample exists, BMC returns a counterexample and the interpolation-sequence is not needed.

Definition 5.2. A *BMC-partitioning* for φ^N is the sequence $\Gamma = \langle A_1, A_2, \dots, A_{N+1}\rangle$ of formulas such that $A_1 = \mathit{INIT}(V^0) \wedge \mathit{TR}(V^0, V^1)$, for every $2 \le i \le N$ $A_i = \mathit{TR}(V^{i-1}, V^i)$ and $A_{N+1} = \neg p(V^N)$. Note that $\varphi^N = \bigwedge_{i=1}^{N+1} A_i$ $(= \bigwedge \Gamma)$.

For a bound N, consider a BMC formula φ^N and its BMC-partitioning Γ. In case φ^N is unsatisfiable, the interpolation-sequence of Γ is denoted by $\bar{I}^N = \langle I_0^N, I_1^N, \dots, I_{N+1}^N\rangle$. Note that Γ contains $N+1$ elements and therefore the interpolation-sequence contains $N+2$ elements where the first element and the last one are always $\top$ and $\bot$, respectively.

Next, we intuitively explain our method. We start with $N = 1$. Consider the formula φ^1 and its BMC-partitioning: $\langle A_1, A_2\rangle$. In case φ^1 is unsatisfiable, there exists an interpolation-sequence of the form $\bar{I}^1 = \langle I_0^1 = \top, I_1^1, I_2^1 = \bot\rangle$. By Def. 4.2, $\top \wedge A_1 \Rightarrow I_1^1$ where $A_1 = \mathit{INIT}(V^0) \wedge \mathit{TR}(V^0, V^1)$. Therefore $S_1 \subseteq I_1^1$, where S_1 is the set of states reachable from the initial states in one transition. Also, $I_1^1 \wedge \neg p(V^1)$ is unsatisfiable, since $I_1^1 \wedge A_2 \Rightarrow \bot$, where $A_2 = \neg p(V^1)$. Therefore, $I_1^1 \models p$.

In the next BMC iteration, for $N = 2$, consider φ^2 and its BMC-partitioning $\langle A_1, A_2, A_3\rangle$. In case φ^2 is unsatisfiable, we get $\bar{I}^2 = \langle \top, I_1^2, I_2^2, \bot\rangle$. Here too, $S_1 \subseteq I_1^2$ and the states reachable from it in one transition are a subset of I_2^2 since $I_1^2 \wedge A_2 \Rightarrow I_2^2$. Also, $S_2 \subseteq I_2^2$ and $I_2^2 \models p$. Let us define the sets $F_1 = I_1^1 \wedge I_1^2$ and $F_2 = I_2^2$. These sets have the following properties, $S_1 \subseteq F_1$, $S_2 \subseteq F_2$, $F_1 \models p$ and $F_2 \models p$. Moreover, $F_1[V^1 \leftarrow V] \wedge TR(V, V') \Rightarrow F_2[V^2 \leftarrow V']$.

In the general case if φ^N is unsatisfiable then for every $1 \le j \le N$, $S_j \subseteq I_j^N$. If we now define $F_j = \bigwedge_{k=j}^{N} I_j^k$ then for every $1 \le j \le N$ we get:

- $F_j \models p$ since $I_j^j \models p$.
- $F_j \wedge \mathit{TR}(V, V') \Rightarrow F'_{j+1}$ since $I_j^k(V^j) \wedge \mathit{TR}(V^j, V^{j+1}) \Rightarrow I_{j+1}^k(V^{j+1})$ for every $1 \le k \le N$
- $S_j \subseteq F_j$ since $S_j \subseteq I_j^k$ for every $1 \le k \le N$.

As a result, the sequence $\langle F_0 = \mathit{INIT}, F_1, F_2, \dots, F_N\rangle$ is a RS and can be used to determine if $M \models f$. Intuitively, the sets I_j are similar to the sets S_j computed by SMC except that they are over-approximations of S_j. Therefore, these sets can

[1] Since we compute over-approximated sets of reachable states, the computed sets are not monotonic. Therefore, we cannot define a monotonic function g for which the existence of a fixpoint is guaranteed.

function $\textsc{UpdateReachable}(\Omega, \bar{I}^k)$
    $j = 1$
    while $(j < k)$ **do**
        $F_j = F_j \wedge I_j^k$
        $\Omega[j] = F_j$
        $j = j + 1$
    end while
    $\Omega[k] = I_k^k$
end function

Figure 2. Updating the reachability sequence Ω

be used to imitate the forward reachability analysis of the model's state-space by means of an over-approximation. This is done in the following manner. BMC runs as usual with one extension. After checking bound N, if a counterexample is found, the algorithm terminates. Otherwise, the interpolation-sequence $\bar{I}^N$ is extracted and the sets F_j for $1 \leq j \leq N$ are updated. If $F_j \Rightarrow \bigvee_{i=1}^{j-1} F_i$ for some $1 \leq j \leq N$, then we conclude that a fixpoint has been reached and all reachable states have been visited. Thus, $M \models f$. If no fixpoint is found, the bound N is increased and the computation is repeated for $N + 1$.

Next, we explain why the algorithm uses $F_j = \bigwedge_{k=j}^{N} I_j^k$ rather than I_j^N in its Nth iteration. Informally, the following facts are needed in order to guarantee the correctness of the algorithm. For every $1 \leq j \leq N$ we need the following:

1. F_j should satisfy p.
2. $F_j(V) \wedge TR(V, V') \Rightarrow F_{j+1}(V')$ for $j \neq N$.
3. $S_j \subseteq F_j$.

This means that the algorithm cannot be implemented using the extracted interpolation sequence $\bar{I}^N$ alone. This is because $\bar{I}^N$ does not satisfy condition (1): while $I_N^N \models p$, I_j^N for $j \neq N$, does not necessarily satisfy p. This can be remedied by conjoining each I_j^N with I_j^j. However, now condition (2) no longer holds. Taking $F_j = \bigwedge_{k=j}^{N} I_j^k$ results in a sequence with all three properties. By that, the sequence follows the properties of Def. 5.1.

The algorithms for updating the RS and checking for a fixpoint are described in Figure 2 and Figure 3, respectively. The complete model checking algorithm using the method described above is given in Figure 4. We refer to it as *Interpolation-Sequence Based Model Checking* (ISB).

It is important to note that a call to $\textsc{UpdateReachability}$ changes all elements of the RS Ω. Therefore, the function $\textsc{FixpointReached}$ cannot count on inclusion checks done in previous iterations and needs to search for a fixpoint at every point in Ω. Moreover, it is not sufficient to check for inclusion of only the last element I_N of Ω. Indeed, if there exists $j \leq N$ such that $F_j \Rightarrow \bigvee_{i=1}^{j-1} F_i$ then all

function FixpointReached(Ω)
  $j = 1$
  while ($j \leq \Omega.length$) **do**
    $R = \bigvee_{i=0}^{j-1} F_i$
    $\varphi = F_j \wedge \neg R$ // Negation of $F_j \Rightarrow R$
    if (SAT(φ) == false) **then return** true
    end if
    $j = j + 1$
  end while
  return false
end function

Figure 3. Checking if a fixpoint has been reached

function ISB(M,f)
  $k := 0$
  $result$ = BMC($M, f, 0$)
  if ($result == cex$) **then**
    return cex
  end if
  $\Omega = \langle INIT \rangle$ // Reachability sequence
  while (true) **do**
    $k = k + 1$
    $result$ = BMC(M, f, k)
    if ($result == cex$) **then**
      return cex
    end if
    $\bar{I}^k = \langle \top, I_1^k, \ldots, I_k^k, \bot \rangle$
    UpdateReachable(Ω,$\bar{I}^k$)
    if (FixpointReached(Ω) == $true$) **then**
      return $true$
    end if
  end while
end function

Figure 4. The ISB Algorithm

reachable states have been found already. However, the implication $F_N \Rightarrow \bigvee_{i=1}^{N-1} F_i$ might not hold due to additional *unreachable* states in I_N. This is because for all $1 \leq j < N$, F_{j+1} is an over-approximation of the states reachable from F_j and not the exact image (that is, $F_j(V) \wedge TR(V, V') \Rightarrow F_{j+1}[V \leftarrow V']$ rather than $F_j(V) \wedge TR(V, V') \equiv F_{j+1}[V \leftarrow V']$).

Theorem 5.3. *For every model M and property $f = \mathbf{AG}\, p$ there exists a bound N such that ISB terminates. Moreover,*

```
function CheckReachable(M,f,k)
    R = M.INIT        // Initialize R - initial states of M
    if (BMC(M, f, 1, k) == cex) then
        return cex
    end if
    M' = M
    repeat
        A = J(V^0) ∧ TR(V^0, V^1)
        B = TR(V^1, V^2) ∧ ... ∧ TR(V^{k-1}, V^k) ∧ ( ⋁_{j=1}^{k} ¬p(V^j))
        J = SAT.getInterpolant(A, B)
        if J ⊆ R then
            return fixpoint
        end if
        R = R ∪ J
        M'.INIT = J
    until (BMC(M', f, 1, k) == cex)
    return abort
end function
```

Figure 5. Computing reachable states using interpolation and BMC with a specific bound k

- *$M \models f$ if and only if there exists an index $0 < j \leq N$ such that $F_j \Rightarrow \bigvee_{i=0}^{j-1} F_i$.*
- *There exists a path π of length N such that π violates f if and only if ISB returns cex.*

6. Interpolation Based Model Checking (IB)

In [22], *interpolation* has been suggested for the first time in order to obtain a SAT-based model checking algorithm for full verification.

The algorithm, referred to as *Interpolation Based Model Checking* (IB), combines BMC and Interpolation [11]. Similarly to the ISB algorithm presented in the previous section, the interpolant is used to compute a reachability sequence (Def. 5.1). However, the computation is done differently. As before, the algorithm concludes that the property holds when a fixpoint is reached during the computation of the reachable states and none of the computed states violates the property.

The following definition is useful in explaining the interpolation based algorithm. Recall that the verified property is of the form $f = \mathbf{AG}\, p$.

Definition 6.1. For a set of states X, X is a S_j*-approximation* w.r.t N, where $1 \leq j \leq N$, if the following two conditions hold: $S_j \subseteq X$ and there is no path of length $(N - j)$ or less violating p, starting from a state $s \in X$. We write $S_j \preceq_N X$ to denote that X is a S_j-approximation w.r.t N.

Note that the formula φ^k is used in BMC to represent a counterexample of length exactly k. This formula can be modified to represent a counterexample of length l for $1 \leq l \leq k$. We denote this formula by $\varphi^{1,k}$ and write $BMC(M, f, 1, k)$ when BMC runs on $\varphi^{1,k}$.

Formula 2. $\varphi^{1,k} = \mathit{INIT}(V^0) \wedge \mathit{TR}(V^0, V^1) \wedge \mathit{TR}(V^1, V^2) \wedge \ldots \wedge \mathit{TR}(V^{k-1}, V^k) \wedge (\bigvee_{j=1}^{k} \neg p(V^j))$

Consider the following partitioning for $\varphi^{1,k}$:

- $A = \mathit{INIT}(V^0) \wedge \mathit{TR}(V^0, V^1)$
- $B = \bigwedge_{i=1}^{k-1} \mathit{TR}(V^i, V^{i+1}) \wedge (\bigvee_{j=1}^{k} \neg p(V^j))$.

Clearly $\varphi^{1,k} \equiv A \wedge B$. Assume that $\varphi^{1,k}$ is unsatisfiable. By the interpolation theorem [11], there exists an interpolant J_1^k which, by Def. 4.1, has the following properties:

- J_1^k is defined over the variables of $\mathrm{Vars}(A) \cap \mathrm{Vars}(B)$, namely, V^1.
- $A \Rightarrow J_1^k$. Hence, $S_1 \subseteq J_1^k$.
- $J_1^k(V^1) \wedge B$ is unsatisfiable. This means that there is no path of length $k-1$ or less, starting from J_1^k, which violates p.

By the above we get that $S_1 \preceq_k J_1^k$. At this point, we get the reachability sequence $\langle \mathit{INIT}, J_1^k \rangle$. We can now proceed by replacing the initial states of M with the computed interpolant J_1^k. BMC is reinvoked with the same bound k and with the modified model $M' = (V, U, J_1^k[V^1 \leftarrow V], \mathit{TR})$ in which the initial states are J_1^k. A new interpolant J_2^k is then extracted. J_2^k satisfies $S_2 \preceq_{k+1} J_2^k$. The reachability sequence is then updated and contains a new element $\langle \mathit{INIT}, J_1^k, J_2^k \rangle$.

It is important to notice that J_1^k now satisfies $S_1 \preceq_{k+1} J_1^k$ since the BMC run on M' did not find a counterexample of length k starting from a state in J_1^k. In the general case we replace INIT with J_i^k and get J_{i+1}^k. By that, at the end of the i-th iteration, for a given bound k, the reachability sequence is $\langle \mathit{INIT}, J_1^k, J_2^k, \ldots, J_i^k \rangle$.

Figure 5 presents, for a given bound k, the computation of an over-approximated set of reachable states. Note that after L iterations of the main loop in CheckReachable we get L interpolants and for every $1 \leq i \leq L$, $S_i \preceq_{k+L} J_i^k$. All computed states are collected in R. If at any iteration, the interpolant J is contained in R, then all reachable states have been found with no violation of f. CheckReachable then returns "*fixpoint*".

On the other hand, if a counterexample is found on a modified model, then CheckReachable(M,f,k) is aborted, the reachability sequence is discarded, and CheckReachable(M,f,$k+1$) is initiated. CheckReachable now tries to construct a new reachability sequence. Recall that the counterexample has been obtained on an over-approximated set of states and therefore might not represent a real counterexample in the original model. In case a real counterexample exists, it will be found during a BMC run on the original model M for a larger bound.

SMC	ISB	IB
$\langle S_1, \dots, S_N \rangle$	$\langle F_1, F_2, \dots, F_N \rangle$ $S_i \preceq_N F_i$ After checking bounds 1 to N	$\langle J_1^1, J_2^1, \dots, J_N^1 \rangle$ $S_i \preceq_N J_i^1$ N iterations at bound 1, if possible
$\langle S_1, \dots, S_{N+L} \rangle$	$\langle F_1, \dots, F_L, \dots, F_{N+L} \rangle$ $S_i \preceq_{N+L} F_i$ After checking bounds 1 to $N+L$	$\langle J_1^N, J_2^N, \dots, J_L^N \rangle$ $S_i \preceq_{N+L} J_i^N$, $(1 \leq i \leq L)$ L iterations at bound N, if possible

Table 1. The correlation between the interpolants computed by ISB and IB to the sets computed by SMC

7. Comparing Interpolation-Sequence Based MC to Interpolation Based MC

In the previous sections we presented two model checking algorithms which combine BMC and interpolation: the Interpolation-Sequence Based (*ISB*) [32] and the Interpolation Based (IB) [22]. Both algorithms are based on the use of interpolation for computing a reachability sequence. In this section we analyze the differences between the algorithms.

Both methods compute an over-approximation of the set of reachable states. However, their state traversals are different. As a result, none is better than the other in all cases. In specific cases, though, one may converge faster.

Several technical details distinguish ISB from IB. First, the formulas from which the interpolants are extracted are different. For a given bound N, ISB uses the formula φ^N while IB uses $\varphi^{1,N}$.

Second, the approximated sets are computed in different manners. ISB computes the sets F_j incrementally and refines them after each iteration of BMC, as part of the BMC loop. IB, on the other hand, recomputes the interpolants whenever the bound is incremented (that is, whenever CheckReachable is called with a larger bound).

Third, ISB can be viewed as an addition to the BMC loop. At each application of BMC (with a different bound), the addition includes the extraction of an interpolation-sequence and the check if a fixpoint has been reached. Indeed, after N iterations of the BMC loop in ISB, there are N over-approximated sets of states, $F_1, \dots, F_N$ satisfying, for each $1 \leq j \leq N$, $S_j \preceq_N F_j$.

On the other hand, IB consists of two nested loops. The outer loop increments the bounds while the inner loop computes over-approximated sets of reachable states. If the outer loop is at some bound $N > 1$ and the inner loop performs L iterations then there are L sets of states $J_1^N, \dots, J_L^N$, each satisfying $S_i \preceq_{N+L} J_i^N$ $(1 \leq i \leq L)$. Table 1 summarizes the above differences.

In summary, IB can compute, at a given bound N, as many sets as needed as long as no counterexample is found (not necessarily a real counterexample). On the other hand, for bound N, ISB can only compute N sets. However, it does not need recurrent BMC calls for each bound (only one is needed). Thus, we can conclude that in cases IB can compute all the needed sets at a low bound it performs better than ISB. However, for examples where the needed sets can only be computed using higher bounds, ISB has an advantage. This fact is reflected in the experimental results reported in [32].

As mentioned before, when a counterexample exists the over-approximated sets of reachable states are not needed. If a property is violated then there exists a minimal bound N for which a violating path of length N exists. Both algorithms have to reach this bound in order to find the counterexample. Here, ISB has a clear advantage over IB. This is because after each BMC run on the original model, IB executes at least one additional BMC run on a modified model. Thus, IB invokes at least two BMC runs for each bound from 1 to $N-1$. Clearly, the second BMC run is more demanding than the inclusion check performed by ISB. In all experiments of [32], falsified properties always favored ISB.

8. SAT-based Reachability via IC3

In this section we describe Property Directed Reachability (PDR), also known as IC3 [3,12]. The interpolation based algorithms, presented in previous sections, are based on an unrolling of the model's transition relation in order to traverse its state space. IC3, on the other hand, avoids such unrolling.

IC3 is a SAT-based model checking algorithm that, given a model M and a property $\mathbf{AG}\,p$, computes a *monotonic* reachability sequence (MRS) $\Omega(M,p,k) = \langle F_0,\ldots,F_k\rangle$ (Def. 5.1) with an increasing k. The algorithm works iteratively, where at iteration k, the MRS of length $k+1$ is extended to an MRS of length $k+2$ by initializing the set F_{k+1} and possibly updating previous sets (with index $i \le k+1$) using learned invariants.

Definition 8.1. Let Ω be an MRS. A formula η is *inductive* up to j, if $F_j \wedge \eta \wedge TR \Rightarrow \eta'$. η is an *invariant* up to level j if $F_i \Rightarrow \eta$ holds for each $i \le j$.

Note that if η is inductive up to j then $F_i \wedge \eta \wedge TR \Rightarrow \eta'$ holds for each $i \le j$. Due to the properties of an MRS, η is an invariant up to j iff it is inductive up to level $j-1$, and in addition $F_0 \Rightarrow \eta$ (initialization).

Instead of unrolling the transition relation, IC3 uses *local reachability checks* between consecutive sets F_i and F_{i+1} to eliminate unreachable states. The computation continues until either a counterexample is found or a fixpoint is reached (i.e. $F_{i+1} \Rightarrow F_i$ for some i), in which case all reachable states satisfy the desired property.

We give a brief overview of how IC3 operates. For a complete description we refer the reader to [3].

IC3 starts by checking if either $INIT \wedge \neg p$ or $INIT \wedge TR \wedge \neg p'$ is satisfiable, in which case a counterexample of length zero or one is found and the algorithm terminates. If both are unsatisfiable, F_0 is initialized to $INIT$ and F_1 is initialized to p. $\langle F_0, F_1\rangle$ is an MRS (it satisfies the conditions in Def. 5.1).

IC3 extends and updates Ω, while strengthening the F_i's. The kth iteration starts from an MRS $\langle F_0,\ldots,F_k\rangle$. Then F_{k+1} is initialized to p. Clearly, $F_k \Rightarrow F_{k+1}$ and $F_{k+1} \Rightarrow p$ hold. Therefore, the purpose of strengthening is to ensure that $F_k \wedge TR \Rightarrow F'_{k+1}$. This is done by checking that $F_k \wedge TR \wedge \neg p'$ is unsatisfiable. If this formula is satisfiable then a state $s \in F_k$ is retrieved from the satisfying assignment. s is a bad state since it reaches $\neg p$ (and by that violates $F_k \wedge TR \Rightarrow F'_{k+1}$). At this point, either s is reachable from $INIT$, in which case a counterexample exists, or s is unreachable and needs to be removed from F_k. In order to determine if s is reachable, IC3 checks the formula: $F_{k-1} \wedge TR \wedge s'$. If this formula is unsatisfiable, then s can be removed from F_k (since the property $F_{k-1} \wedge TR \Rightarrow F'_k$ of an MRS holds without it as well), and the same process is repeated for other states in F_k that can reach $\neg p$ (if any). However, if $F_{k-1} \wedge TR \wedge s'$ is satisfiable, a predecessor $t \in F_{k-1}$ of s is extracted and handled similarly to s in order to determine if t (which is also a bad state) is reachable from $INIT$ or not. IC3 therefore moves back and forth along the F_i's, while retrieving bad states b and checking their reachability from $INIT$ via local reachability checks of the form $F_i \wedge TR \wedge b'$. During this process, the F_i's are strengthened by removing bad states that are not reachable. In fact, in order to remove a bad state b from F_i, IC3 finds a clause c that is an *invariant* up to i and implies $\neg b$, and adds c to F_i as a conjunct. If a state in $F_0 = INIT$ is reached during the backwards traversal, then a counterexample is obtained.

Definition 8.2. Satisfiability checks of the form $F_i \wedge TR \wedge \eta$ (where $\mathrm{Vars}(\eta) \subseteq V \cup V'$) are called *i-reachability checks*.

9. Abstraction

We consider the "visible variables" abstraction [18]. We start by describing it in our context. Let $M_c = (V, U, INIT, TR)$ be a model and let $U_i \subseteq U$ be a set of state-variables. We refer to U_i as the set of "visible variables".

Given U_i, we define an abstract model $M_i = (V_i, U_i, TR_i)$ of M_c where $TR_i = \bigwedge_{v \in U_i} (v' = f_v(V, V'))$ is an abstract transition relation, and $V_i \subseteq V$ is defined by $\{v \in V \mid v \in \mathrm{Vars}(TR_i) \vee v' \in \mathrm{Vars}(TR_i)\}$. Note that the behavior of invisible state variables (in $U \setminus U_i$) is nondeterministic.

We do not introduce an abstraction of $INIT$ as part of M_i since we always consider the concrete set of initial states. M_i is an *abstraction* of M_c, denoted $M_c \preceq M_i$, in the sense that both its set of states and its transition relation are abstractions of the concrete ones, as explained below. M_i induces a set of abstract states S_i which includes all valuations to V_i. Specifically, each concrete state $s \in S$ is abstracted by the abstract state $s_i \in S_i$ that agrees with s on the assignment to the joint variables in V_i. In this case we write $s \preceq s_i$. We sometimes refer to s_i as the set of concrete states it abstracts: $\{s \in S | s \preceq s_i\}$.

In addition, TR is abstracted by TR_i in the sense that $TR \Rightarrow TR_i$. Formally, the relation $\{(s, s_i) \mid s \preceq s_i\}$ is a simulation relation [25] from M_c to M_i.

Given an MRS $\Omega(M_c, p, k) = \langle F_0, \ldots, F_k \rangle$ and an abstract model M_i, we say that a formula η is *inductive* up to level j w.r.t. M_i, if $F_j \wedge \eta \wedge TR_i \Rightarrow \eta'$.

Lemma 9.1. *Any formula inductive up to j w.r.t. M_i is also inductive up to j w.r.t. M_c.*

The lemma holds since $TR \Rightarrow TR_i$. When we do not explicitly mention a model, we refer to inductiveness w.r.t. M_c. The notion of an invariant always refers to M_c.

9.1. Lazy Abstraction

lazy abstraction [15] allows to use different details of the model at different iterations of the state-space traversal. We adapt the notion of lazy abstraction to abstraction based on *visible variables* [18], and allow different variables to be visible at different time frames.

Definition 9.2. An *abstraction sequence* w.r.t. a model M_c is a sequence $\bar{U} = \langle U_0, \ldots, U_k \rangle$ where $U_i \subseteq U$ for $0 \leq i \leq k$, is a set of visible state-variables. $\bar{U}$ is *monotonic* if $U_i \subseteq U_{i+1}$ for each $0 \leq i < k$.

An abstraction sequence $\bar{U}$ represents different levels of abstraction of M_c. It induces a sequence of abstract models $\langle M_0, \ldots, M_k \rangle$ where M_i is defined as above. If $\bar{U}$ is monotonic, the induced sequence of abstract models is also monotonic in the sense that $M_0 \succeq \ldots \succeq M_k \succeq M_c$.

Definition 9.3. Let $\bar{U} = \langle U_0, \ldots, U_k \rangle$ be a monotonic abstraction sequence and $\Omega(M_c, p, k) = \langle F_0, \ldots, F_k \rangle$ an MRS. A sequence $s_i, \ldots, s_j$ of abstract states where $0 \leq i < j \leq k+1$ is an *abstract path from i to j* if (i) for each $i \leq l < j$, $(s_l, s_{l+1}) \models TR_l$, and[2] (ii) for each $i \leq l \leq \min\{j, k\}$, $s_l \cap F_l \neq \emptyset$.

An abstract path $s_0, \ldots, s_j$ from 0 to j is an *abstract counterexample of length j* if $s_j \cap \neg p \neq \emptyset$.

Note that the definition above is not standard. It refers to different transition relations at different steps. Also, it requires the abstract states to be part of the corresponding F_i.

Definition 9.4. An abstraction sequence $\langle {U_0}^r, \ldots, {U_k}^r \rangle$ is a *refinement* of an abstraction sequence $\langle U_0, \ldots, U_k \rangle$ if $U_i \subseteq {U_i}^r$ for each i.

10. Lazy Abstraction and IC3

In this section we describe the algorithm L-IC3, which adds lazy abstraction to IC3. The key ingredients of L-IC3 are an *abstraction sequence* $\bar{U}$ that induces different abstractions at different time frames as well as an *MRS* Ω.

L-IC3 starts with an initialization step and then works in *stages* (Fig. 6). Its initialization (lines 2-5) is similar to the initialization of IC3 with one exception. If no counterexample of length 0 or 1 exists, then in addition to initializing Ω to $\langle F_0 = \mathit{INIT}, F_1 = p \rangle$, it initializes $\bar{U}$ to $\langle U_0 = \mathrm{Vars}(p) \rangle$. Clearly, after initialization, Ω is an MRS.

[2] Requirement (ii) dismisses paths that are known to be spurious based on Ω. $\min\{j, k\}$ is used for the case where $j = k + 1$, in which nonempty intersection is required only up to k.

1: **function** L-IC3(p)
2: $\Omega = \langle INIT, p\rangle; \bar{U} = \langle \mathrm{Vars}(p)\rangle$
3: **if** Init-IC3($\Omega, \bar{U}, p$) == cex **then**
4: **return** cex
5: **end if**
6: **while** A-IC3($\Omega, \bar{U}$) == abs-cex **do**
7: **if** Refine($\Omega, \bar{U}$) == cex **then**
8: **return** cex
9: **end if**
10: **end while**
11: **return** fixpoint
12: **end function**

Figure 6. L-IC3

Each L-IC3 stage (lines 6-10) consists of an abstract model checking step and a refinement step, both performed by variations of IC3. $\bar{U}$ and Ω are updated in both steps.

The abstract model checking extends and updates the MRS Ω until either a fixpoint is reached, or an abstract counterexample is found (line 6). In the latter case, the counterexample is *abstract* since it is computed w.r.t. the abstract transitions. However, it is also restricted by Ω (see Def. 9.3). A refinement is then performed (line 7). If the refinement finds a concrete counterexample then it terminates. Otherwise it refines $\bar{U}$ and updates Ω into an MRS (of the same length).

A new L-IC3 stage (line 6) of abstraction-refinement then begins, invoking A-IC3 with the updated Ω and the refined $\bar{U}$.

An invocation of L-IC3 results in either a fixpoint (in which case the property is proved), or a concrete counterexample.

10.1. Abstract Model Checking via A-IC3

The abstract model checking algorithm, A-IC3 (Fig. 7), either finds an abstract counterexample (line 22), or reaches a fixpoint (line 26) by computing an MRS Ω.

Using different abstractions The computation of Ω is done using a variation of IC3 which considers a *sequence of abstract* models, induced by a monotonic abstraction sequence $\bar{U} = \langle U_0 \ldots, U_k\rangle$. Both abstract transition relations and abstract states are used. Even though abstract models are used, the obtained MRS satisfies the requirements of Def. 5.1, which refer to the concrete transition relation *TR*. To emphasize this, we sometimes refer to the sequence as a *concrete* MRS.

Recall that IC3 performs i-reachability checks of the form $F_i \wedge TR \wedge \eta$. A-IC3 also performs these checks (within Strengthen, line 20), but instead of using the concrete *TR* it uses the *abstract* TR_i. This means that when traversing the model's state space, A-IC3 uses different abstract transition relations at different time frames. Further, when $F_i \wedge TR_i \wedge \eta$ is satisfiable, A-IC3 retrieves an *abstract* state $s_a \in M_i$ from the satisfying assignment. This abstract state is either used to strengthen Ω, or it is part of an abstract counterexample.

13: **function** A-IC3($\Omega, \bar{U}$)
14: $k = |\Omega| - 1$
15: **while** $\Omega.fixpoint()$ == false **do**
16: $U_k = U_{k-1}$
17: $\bar{U}.add(U_k)$
18: $F_{k+1} = p$
19: $\Omega.add(F_{k+1})$
20: result = STRENGTHEN($\Omega, \bar{U}, k$)
21: **if** result == abs-cex **then**
22: **return** abs-cex
23: **end if**
24: $k = k + 1$
25: **end while**
26: **return** fixpoint
27: **end function**

Figure 7. A-IC3

Incrementality If A-IC3 finds a counterexample at iteration k, it returns it. After refinement (line 7) A-IC3 is re-invoked with an updated Ω that is an MRS of the same length. The computation of Ω resumes from iteration $k+1$ (line 14)[3].

Iterations In iteration $k \geq 1$, the MRS $\langle F_0, \ldots, F_k \rangle$ and the abstraction sequence $\langle U_0, \ldots, U_{k-1} \rangle$ are extended by 1 and updated as follows (see Fig. 7).

1. Check if a fixpoint is reached. If not:
2. U_k is initialized to U_{k-1} and added to $\bar{U}$.
3. F_{k+1} is initialized to p and added to Ω.
4. The sets $F_0, \ldots, F_{k+1}$ are strengthened iteratively until $\langle F_0, \ldots, F_{k+1} \rangle$ becomes an MRS, or an abstract counterexample is found.

Below we describe items 2 and 4 in more detail.

(2) Extending $\bar{\mathbf{U}}$: U_k is initialized to U_{k-1} (line 16). This is aimed at immediately eliminating from TR_k spurious transitions that lead from states in $F_{k-1} \subseteq F_k$ to $\neg p$ and were already removed from TR_{k-1}. Note that this initialization does not imply that the U_i sets will always be equal, since refinement might change them in different ways.

(4) Iterative Strengthening of $\mathbf{\Omega}$: A-IC3 obtains an MRS of length $k+1$ by strengthening the F_i's s.t. no abstract counterexample of length $k+1$ exists w.r.t. the MRS $\langle F_0, \ldots, F_k \rangle$. This is a sufficient condition to ensure that Ω is an MRS. For this purpose, A-IC3 finds abstract states that might be a part of an abstract counterexample at a certain time frame, and attempts to block them by learning corresponding invariants. Recall that the abstract counterexamples we consider are restricted not only by the abstract transition relations, but also by the F_i sets (Def. 9.2). Technically, such states are described by abstract proof obligations (similarly to the notion of proof obligations used in IC3).

Definition 10.1. An *abstract proof obligation*, or an *obligation* in short, is a pair (s_a, n) consisting of a level $n \leq k$ and an abstract state s_a s.t. (1) s_a is a "bad

[3] An abstract counterexample is found w.r.t. $\Omega = \langle F_0, \ldots, F_{k+1} \rangle$ produced in iteration k, where $|\Omega| = k+2$. When A-IC3 is re-invoked, k is set to $|\Omega| - 1 = k+1$.

state" that reaches $\neg p$ along some abstract path, (2) $\neg s_a$ is an invariant up until n, (3) $s_a \cap F_{n+1} \neq \emptyset$, and (4) F_n reaches s_a in one step of TR_n.

Thus $n+1$ is the minimal level intersecting s_a, and n is the minimal level reaching s_a in one *abstract* step. Note that it is possible that F_n cannot reach s_a along the concrete transitions. A-IC3 maintains two sets of obligations - *may* and *must*.

Definition 10.2. An obligation (s_a, n) is a *must obligation* w.r.t. iteration k if s_a must be shown unreachable from F_n in one step w.r.t. TR_n, in order to ensure that no abstract counterexample of length $k+1$ exists. All other obligations are *may obligations* w.r.t. k.

If s_a can reach $\neg p$ via an abstract path from level $n+1$ to level $k+1$, then (s_a, n) is a must obligation: unless s_a is blocked from F_{n+1} (by removing from F_n all states that reach s_a in one step), an abstract counterexample of length $k+1$ would exist. The same violation may also be reached from s_a in later levels F_j, $n+1 < j \leq k+1$, in which case it will be a suffix of a longer abstract counterexample with a longer prefix up to s_a. Therefore, we may also want to block s_a in F_j, $n+1 < j \leq k+1$. However, since different abstract transition relations are considered at each level, it is also possible that the same path leading from s_a to $\neg p$ is not valid from level $j > n+1$ since, for example, $U_j \supset U_{n+1}$ and hence the first transition along the path does not satisfy TR_j. In this case, a longer counterexample is not a valid abstract path since its suffix is not valid.The attempt to block a state s_a that is known to reach a violation from level $n+1$ in levels greater than $n+1$ creates *may* obligations[4].

The may obligations are not *required* to be blocked, but blocking them can prevent A-IC3 from encountering the same obligations/states in future iterations. On the other hand, if we report an abstract counterexample based on a may obligation, it is possible that no real abstract counterexample exists, resulting in an unnecessary refinement step which can damage the efficiency of the algorithm. A-IC3 therefore greedily tries to handle may obligations and strengthen Ω accordingly, but refrains from reporting abstract counterexamples based on them. Note that if a may obligation is in fact a must w.r.t. some greater k, then it will reappear as a must obligation in the following iterations.

Key procedures used by A-IC3 for strengthening the F_i's by means of proof obligations appear in Fig. 8 and Fig. 9. A detailed description of these procedures can be found in [31]. Below we provide a brief explanation.

At iteration k, initial obligations are derived from satisfying assignments to the formula $F_k \wedge TR_k \wedge \neg p'$, using the procedure BLOCKSTATE, if the formula is satisfiable (if it is not, then an MRS is obtained and no strengthening is required). Obligations are then handled iteratively until no obligation remains. In order to handle an obligation (s_a, n) and show s_a to be unreachable from F_n in one step, A-IC3 attempts to strengthen F_n by extracting predecessors t_a of s_a that satisfy $F_n \wedge TR_n \wedge s'_a$, defining new proof obligations based on them, and handling

[4] IC3 does not make a distinction between may and must obligations and handles them all the same since in the concrete case, a longer counterexample is always a *valid* path (its suffix reaching a violation is always valid).

```
function Strengthen(Ω,Ū,k)
    while F_k ∧ TR_k ∧ ¬p′ == SAT do
        obligations = ∅
        retrieve abstract predecessor s_k
        if BlockState(Ω,s_k,k,k,must) == abs-cex then
            return abs-cex
        end if
        while obligations ≠ ∅ do
            ((s_a, n), handleMay) = ChooseNext(obligations)
            if F_n ∧ TR_n ∧ s′_a == SAT then
                retrieve abstract predecessor t_n
                if BlockState(Ω,t_n,n,k,must) == abs-cex then
                    if handleMay then
                        obligations.clearAllMust()
                    else
                        return abs-cex
                    end if
                end if
            else
                obligations.removeMust(s_a,n)
                BlockState(Ω,s_a,n + 2,k,may)
            end if
        end while
    end while
    PropagateClauses(Ω)
    return done
end function
```

Figure 8. Iterative strengthening of A-IC3

these obligations (by the same procedure). If F_n is successfully strengthened s.t. $F_n \wedge TR_n \wedge s'_a$ becomes unsatisfiable, then $\neg s_a$ becomes an invariant up to $n+1$.

Adding Invariants If $\neg s_a$ is an invariant up to $n+1$, then a stronger invariant that blocks s_a up to F_{n+1} is learned based on the *abstract model* M_n. Namely, $\neg s_a$ is strengthened to some sub-clause[5] c s.t. $F_0 \Rightarrow c$ and $F_n \wedge c \wedge TR_n \Rightarrow c'$, i.e. c is inductive up to n w.r.t. M_n and hence, by Lemma 9.1, also w.r.t. M_c. Consequently, c is also an invariant up to $n+1$, but it is a stronger invariant than $\neg s_a$ (since $c \Rightarrow \neg s_a$). The clause c is added as a conjunct to $F_0, \ldots, F_{n+1}$ while maintaining the properties of a (concrete) MRS[6].

10.2. Refinement

If A-IC3 finds an abstract counterexample of length $k+1$, refinement is invoked by L-IC3 (line 7). Refinement either finds a concrete counterexample or eliminates *all* concrete spurious counterexamples of length $k+1$. In the latter case, refinement also refines $\bar{U}$ to ensure that no *abstract* counterexample of length $k+1$ exists. Both an updated MRS $\Omega^r = \langle F_0^r, \ldots, F_{k+1}^r \rangle$ and a refined monotonic abstraction sequence $\bar{U}^r = \langle U_0^r, \ldots, U_k^r \rangle$ are returned.

The Refine procedure is described in Fig. 10. Refine first invokes C-Strengthen, the strengthening procedure of the concrete IC3, on the sequence

[5] A state s_a is represented by a conjunction of literals, which makes its negation $\neg s_a$ a clause (i.e., a disjunction of literals). A sub-clause of $\neg s_a$ consists of a subset of its literals.

[6] c is not necessarily inductive w.r.t. M_i where $i < n$ (in case $U_i \subset U_n$).

```
function BlockState(Ω,t_a,l,k,type)
    if l > k + 1 then
        min = k + 1
    else
        min = FindNonInductive(Ω,¬t_a,l − 1,k)
        if min == 0 then
            return abs-cex
        end if
        if min ≤ k then
            if type == must && min == l-1 then
                obligations.addMust(t_a, min)
            else
                obligations.addMay(t_a, min)
            end if
        end if
    end if
    AddInvariant(Ω,¬t_a,min)
    return done
end function
```

Figure 9. BlockState procedure of A-IC3

```
74: function Refine(Ω,Ū)
75:     result = C-Strengthen(Ω)
76:     if result == cex then
77:         return cex
78:     end if
79:     RefineAbstraction(Ω,Ū)
80:     return done
81: end function
```

Figure 10. Refine procedure of A-IC3

$\langle F_0, \ldots, F_{k+1}\rangle$ (whose prefix up to F_k is an MRS) obtained from the abstract model checking. If a concrete counterexample is found the algorithm terminates (lines 75-78). Otherwise, no concrete counterexample of length $k+1$ exists. Moreover, the updated (strengthened) sets $F_0^r, \ldots, F_{k+1}^r$ comprise an MRS. It remains to refine the abstraction sequence $\bar{U}$ in order to eliminate all *abstract* counterexamples of length $k+1$ as well. Thus, RefineAbstraction is invoked (line 79).

RefineAbstraction

A-IC3 found an abstract counterexample since it failed to strengthen the F_i's. Meaning, the relevant i-reachability checks $F_i \wedge TR_i \wedge t_a'$ could not be made unsatisfiable when using TR_i. C-Strengthen, on the other hand, succeeds to do so. Namely, for each i-satisfiability check $F_i \wedge TR_i \wedge t_a'$ of A-IC3 that was satisfiable, C-Strengthen manages to make the corresponding check $F_i^r \wedge TR \wedge t'$ for each $t \preceq t_a$ unsatisfiable, either by strengthening F_i^r or simply since it considers TR. Moreover, once $F_i^r \wedge TR \wedge t'$ becomes unsatisfiable, C-Strengthen derives from it a clause $c \Rightarrow \neg t$ s.t. $F_i^r \wedge c \wedge TR \Rightarrow c'$ holds. C-Strengthen strengthens Ω^r by adding c (invariant) as a new clause in all sets up to F_{i+1}^r. We consider it a *learned clause* at level $i+1$. The purpose of RefineAbstraction is to ensure that for a learned clause c at level $i+1$, $F_i^r \wedge c \wedge TR_i^r \Rightarrow c'$ (with TR_i^r instead of TR) also holds. Meaning, c is inductive up to i w.r.t. M_i^r.

Lemma 10.3. *Let c be a clause learned by* C-Strengthen *at level $i+1$. If $F_i^r \wedge \mathrm{TR}_i^r \Rightarrow {F_{i+1}^r}'$ then $F_i^r \wedge c \wedge \mathrm{TR}_i^r \Rightarrow c'$.*

Based on the previous lemma, in order to ensure $F_i^r \wedge c \wedge TR_i^r \Rightarrow c'$, it suffices to ensure unsatisfiability of $F_i^r \wedge TR_i^r \wedge \neg {F_{i+1}^r}'$ for every level $i+1$ in which learned clauses exist.

To ensure unsatisfiability of a formula $F_i^r \wedge TR_i^r \wedge \neg {F_{i+1}^r}'$, we consider the same formula over TR, which is clearly unsatisfiable. We derive from it an unSAT-core. The next-state variables that appear in the unSAT-core, denoted $NS(\text{unSatCore}) = \{v \in V \mid v' \in \text{Vars}(\text{UnSatCore})\}$, are added to U_i.

Lemma 10.4. *Let $F_i^r \wedge \mathrm{TR} \wedge \eta'$ be an unsatisfiable formula and let UnSatCore be its unsat core. Let $U_i^r \supseteq NS(\mathit{UnSatCore})$. Then $F_i^r \wedge \mathrm{TR}_i^r \wedge \eta'$ is unsatisfiable.*

Finally, we propagate variables that were added to U_i^r forward in order to obtain a monotonic abstraction sequence. Since we only add variables to U_i^r, i.e. make the transition relation TR_i^r more precise, then the corresponding formulas remain unsatisfiable.

10.3. Correctness Arguments

The MRS obtained by L-IC3 is concrete. Specifically, it does not necessarily satisfy $F_i \wedge TR_i \Rightarrow F_{i+1}$. This results both from refinement that adds invariants learned based on the concrete TR, and from A-IC3 that learns an invariant based on some TR_i, but also adds it to F_{j+1} for $j < i$ even if it is not inductive w.r.t. TR_j. This complicates the correctness proof.

In particular, in IC3, when a proof obligation (s, n) is handled, then for any predecessor t of s, $\neg t$ is an invariant up to $n-1$, otherwise s would belong to a lower frame (since $F_i \wedge TR \Rightarrow F_{i+1}$). Now consider an abstract proof obligation (s_a, n). If we assume to the contrary that the predecessor t_a intersect some F_i (for $i < n$) then we can still deduce that the transition $(t_a, s_a) \models TR_n$ also exists at a lower frame, i.e. $(t_a, s_a) \models TR_i$ for $i < n$. This is since $TR_n \Rightarrow TR_i$ (recall that the same does not necessarily hold for $i > n$). However, if $t_a \cap F_i \neq \emptyset$, we cannot immediately deduce that $s_a \cap F_{i+1} \neq \emptyset$ since $F_i \wedge TR_i \Rightarrow F_{i+1}$ might *not* hold. It turns out that this property does hold (see [31]), but more complicated arguments are needed, based on the following:

Lemma 10.5. *Let $\Omega = \langle F_0, \ldots, F_{k+1} \rangle$ and $\bar{U} = \langle U_0, \ldots, U_k \rangle$ be the sequences obtained at the end of a refinement step or at the end of an iteration of A-IC3 in the case that no counterexample was found. Then*

1. *Ω is an MRS.*
2. *For every clause c that was added to some F_i in Ω there exists some $j \geq i-1$ s.t. c is inductive up to j w.r.t. M_j.*
3. *No abstract counterexample of length $k+1$ exists w.r.t. the prefix $\langle F_0, \ldots, F_k \rangle$ of Ω.*

Theorem 10.6. *L-IC3 either terminates with a fixpoint, in which case the property holds, or with a concrete counterexample.*

N	♯Vars	Laziness - Time Frames and Number of Vars													
		♯TF	♯AV	♯TF	♯AV	♯TF	♯AV	♯TF	♯AV	♯TF	♯AV	♯TF	♯AV	♯TF	♯AV
f_1	11866	[0-0]	323	[1-1]	647	[2-2]	686	[3-3]	699	[4-4]	705	[5-5]	713	[6-6]	714
		[7-7]	728	[8-8]	743	[9-9]	752	[10-10]	755	[11-11]	761	[12-12]	767	[13-13]	777
		[14-14]	783	[15-15]	789	[16-18]	811								
f_2	5693	[0-7]	12												
f_3	5693	[0-0]	8	[1-1]	56	[2-2]	64	[3-3]	74	[4-4]	82	[5-7]	91		
f_4	5693	[0-6]	31	[7-7]	42	[8-8]	51	[9-13]	54						
f_5	5773	[0-0]	260	[1-1]	381	[2-2]	401	[3-3]	419	[4-34]	430				
f_6	1183	[0-0]	185	[1-1]	248	[2-2]	255	[3-3]	259	[4-4]	262	[5-5]	268	[6-8]	270
		[9-9]	273	[10-30]	274										
f_7	1247	[0-0]	57	[1-1]	62	[2-2]	73	[3-7]	76						
f_8	1247	[0-0]	63	[1-1]	64	[2-2]	72	[3-6]	83						
f_9	1277	[0-0]	263	[1-1]	303	[2-2]	318	[3-3]	321	[4-4]	322	[5-5]	323	[6-26]	347
f_{10}	1389	[0-0]	253	[1-1]	304	[2-2]	324	[3-3]	341	[4-4]	351	[5-5]	355	[6-7]	363
		[8-9]	399	[10-10]	409	[11-12]	415	[13-13]	419	[14-16]	429	[17-18]	431		
f_{11}	1183	[0-0]	79	[1-1]	113	[2-9]	114								
f_{12}	1204	[0-0]	58	[1-1]	67	[2-2]	75	[3-7]	76						
f_{13}	3844	[0-0]	470	[1-1]	504	[2-2]	528	[3-3]	533	[4-4]	534	[5-11]	650		
f_{14}	3832	[0-0]	333	[1-1]	365	[2-2]	386	[3-5]	391	[6-6]	442	[7-10]	446		
f_{15}	3854	[0-0]	428	[1-1]	453	[2-2]	495	[3-3]	499	[4-4]	503	[5-5]	560	[6-6]	574
		[7-7]	576	[8-10]	577										
f_{16}	3848	[0-0]	432	[1-1]	462	[2-2]	487	[3-3]	498	[4-4]	501	[5-5]	634	[6-6]	650
		[7-13]	658												
f_{17}	3854	[0-0]	426	[1-1]	480	[2-2]	525	[3-3]	539	[4-4]	540	[5-5]	559	[6-11]	570
f_{18}	3848	[0-0]	469	[1-1]	547	[2-2]	551	[3-3]	553	[4-4]	635	[5-5]	672	[6-10]	674

Table 2. Lazy abstraction. N stands for the name of the verified property. ♯*Vars* stands for the number of state variables in the concrete model M_c. ♯TF stands for the time frames and ♯AV represents the number of variables (defining the abstract TR_i) in the abstract model M_i at the given time frame i (appearing in the column ♯TF).

10.4. Lazy IC3 In Practice

The laziness of the abstraction-refinement algorithm used by L-IC3 is demonstrated in Table 2. The table shows how the abstraction is refined along increasing time frames. Different frames contain different variables that are needed in order to prove or disprove the given property. This demonstrates the fact that L-IC3 indeed takes advantage of the lazy abstraction framework.

11. Conclusion

We presented four methods for SAT-based unbounded model checking. The first two, ISB and IB, are based on interpolation-sequence and interpolation, respectively. The other two, IC3 and L-IC3, are based on local reachability checks. All approaches are based on an overapproximate computation of the set of reachable states of a given system. The computation continues until either a fixpoint is reached or a counterexample is generated. They differ in the way this computation is conducted. In particular, ISB and IB require unrolling of the transition relation and use interpolants to overapproximate sets of reachable states, whereas IC3 and L-IC3 conduct only local reachability checks, which involve consecutive time frames and require no unrolling. As is often the case with model checking techniques and tools, none of the approaches is overall superior to the other.

References

[1] A. Biere and C. Artho. Liveness checking as safety checking. In *FMICS02*.

[2] A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu. Symbolic Model Checking Without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, LNCS. Springer.

[3] A. R. Bradley. SAT-based model checking without unrolling. In *VMCAI*, 2011.

[4] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE transactions on Computers*, C-35(8):677–691, 1986.

[5] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[6] Gianpiero Cabodi, Sergio Nocco, and Stefano Quer. Interpolation sequences revisited. In *DATE*, pages 316–322, 2011.

[7] P. Chauhan, E. M. Clarke, and D. Kroening. Using SAT based image computation for reachability analysis. Technical Report CMU-CS-03-151, Carnegie Mellon University, School of Computer Science, 2003.

[8] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[9] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. *Journal of the ACM*, 50(5):752–794, 2003.

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

[11] W. Craig. Linear reasoning. a new form of the herbrand-gentzen theorem. *Journal of Symbolic Logic*, 22(3):250–268, 1957.

[12] N. Een, A. Mishchenko, and R. Brayton. Efficient implementation of property directed reachability. In *FMCAD*, 2011.

[13] N. Eén and N. Sörensson. An extensible SAT-solver. In *SAT*, pages 502–518, 2003.

[14] O. Grumberg, A. Schuster, and A. Yadgar. Reachability Using a Memory-Efficient All-Solutions SAT Solver. In *Fifth Inernation Conference on Formal Methods in Computer-Aided Design (FMCAD'04)*, November 2004.

[15] T.A. Henzinger, R. Jhala, and R. Majumdar. Lazy abstraction. In *POPL'02*.

[16] R. Jhala and K.L. McMillan. Interpolant-Based Transition Relation Approximation. In *17th International Conference on Computer Aided Verification (CAV'05)*, LNCS 3576, Edinburgh, July 2005.

[17] O. Kupferman and M.Y. Vardi. Model checking of safety properties. In *Computer-Aided Verification (CAV'99)*, Lecture Notes in Computer Science. Springer-Verlag.

[18] R. P. Kurshan. *Computer-aided verification of coordinating processes: the automata-theoretic approach*. Princeton University Press, 1994.

[19] J.P. Marques-Silva and K.A. Sakallah. Conflict analysis in search algorithms for propositional satisfiability. In *IEEE International Conference on Tools with Artificial Intelligence*, 1996.

[20] K. McMillan. Applications of craig interpolation to model checking. In *11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Lecture Notes in Computer Science, pages 1–12, Edinburgh, Scotland, April 2005. Springer.

[21] Ken L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Computer Aided Verification*, 2002.

[22] K.L. McMillan. Interpolation and SAT-based Model Checking. In *Proceedings of the 15th International Conference on Computer Aided Verification (CAV'03)*, volume 2725 of *LNCS*, Bolder, Colorado, 2003.

[23] K.L. McMillan. Lazy Abstraction with Interpolants. In *18th International Conference on Computer Aided Verification (CAV'06)*, LNCS 4144, Seattle, August 2006.

[24] K.L. McMillan and N. Amla. Automatic abstraction without counterexamples. In *Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, pages 331–346, Warsaw, Poland, April 2003.

[25] R. Milner. An algebraic definition of simulation between programs. In *In proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pages 481–489, September 1971.

[26] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient SAT solver. In *39th Design Aotomation Conference (DAC'01)*, 2001.

[27] A. Pnueli. The temporal logic of programs. In *Proceedings of the Eighteenth Annual Symposium on Foundations of Computer Science (FOCS'77)*, 1977.

[28] J.P. Quielle and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the 5th International Symposium on programming*, 1982.

[29] M. Sheeran, S. Singh, and G. Staalmarck. Checking safety properties using induction and a SAT-solver. In *Third International Conference on Formal methods in Computer-Aided Design (FMCAD'00)*, Austin, Texas, November 2000.

[30] M. Sheeran and G. Staalmarck. A tutorial on stalmarck's proof procedure for propositional logic. *Formal Methods in System Design*, 16(1), January 2000.

[31] Y. Vizel, O. Grumberg, and S. Shoham. Lazy abstraction and SAT-based reachability in hardware model checking. In *FMCAD*, 2012.

[32] Yakir Vizel and Orna Grumberg. Interpolation-sequence based model checking. In *FMCAD*, pages 1–8, 2009.

[33] Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany*, pages 10880–10885. IEEE Computer Society, 2003.