Interpolation-Sequence Based Model Checking

Yakir Vizel[1,2] Orna Grumberg[1]

1. Computer Science Department, The Technion, Haifa, Israel
{yvizel,orna}@cs.technion.ac.il

2. Architecture, System Level and Validation Solutions, Intel Development Center, Haifa, Israel

***Abstract*—SAT-based model checking is the most widely used method for verifying industrial designs against their specification. This is due to its ability to handle designs with thousands of state elements and more. The main drawback of using SAT-based model checking is its orientation towards "bug-hunting" rather than full verification of a given specification. Previous works demonstrated how Unbounded Model Checking can be achieved using a SAT solver. In this work we present a novel SAT-based approach to full verification. The approach combines BMC with *interpolation-sequence* in order to imitate BDD-based Symbolic Model Checking. We demonstrate the usefulness of our method by applying it to industrial-size hardware designs from Intel. Our method compares favorably with McMillan's interpolation based model checking algorithm.**

I. Introduction

Model checking [6] is an automatic approach to formally verifying that a given system satisfies a given specification. The system to be verified is modelled as a finite state machine and the specification is described using temporal logic [13]. Model checking algorithms are based on exploration of the models' state space while searching for violations of the specification.

The introduction of BDD-based *Symbolic Model Checking* (SMC) [5] enabled model checking of real-life designs with a few hundreds of state elements. However, current design blocks with well-defined functionality typically have thousands of state elements and more. SAT-based *Bounded Model Checking* (BMC) [4] can handle designs of that scale. However, BMC is limited to finding a counterexample of a bounded length. Thus, BMC is usually used for bug hunting.

In this work we present a novel SAT-based approach to full verification. The approach combines BMC with *interpolation-sequence* [8], [11] in order to imitate BDD-based Symbolic Model Checking. Our method runs BMC iteratively as usual. However, at each iteration k, if the checked formula is unsatisfiable, then a sequence of k interpolants $\{I_1^k, \ldots, I_k^k\}$ is computed. I_j^k over-approximates the set S_j of states, reachable from the initial states in j steps. In the next BMC iteration, the newly obtained interpolant I_j^{k+1} is conjuncted with I_j^k. The result, denoted I_j, is itself an over-approximation of S_j, but a more precise one, since it contains less states which are not in S_j. Thus, I_j can be viewed as a *refinement* of the computed interpolants. Further, I_j is guaranteed to include no violation of the checked property.

The process terminates with either a counterexample produced by BMC, or by reaching a fixpoint, indicating that no more reachable states will be found. In the latter case, since no violation of the formula has been encountered so far, it is guaranteed that the property holds.

We emphasize that the setting of combining BMC with interpolation in order to compute an over-approximation of the set of reachable states seems similar to McMillan's interpolation based model checking algorithm [10]. However, exploiting interpolation-sequence the way we do results in a different traversal of the sets of reachable states, thus may converge faster. Furthermore, our algorithm often requires less calls for BMC. The paper includes a thorough comparison between the two methods, both on the algorithmic level and by running experiments. Our comparison identifies important cases in which our algorithm performs better than the one in [10].

We implemented our algorithm and the one in [10] within Intel's verification tool. All experiments were conducted on models from Intel's next generation Microprocessor designs. The checked properties are real specifications, used to verify those designs. The experiments compare various parameters of the two methods. In all our experiments, when a fixpoint could be reached only at a high bound, our method performed better than [10]. The algorithm in [10], on the other hand, performed better when a fixpoint could be reached at a low bound. In addition, falsified properties always favored our method.

When describing our method we assume a safety property of the form AGq, where q is a propositional formula. This, however, does not restrict its generality since model checking of liveness properties can be reduced to handling safety properties [2]. Further, model checking of safety properties can be reduced to handling properties of the form AGq [9].

A. Related Work

SAT-based *Bounded Model Checking* (BMC) [4] is widely used for the verification of large systems. BMC can usually handle much larger designs than other known methods such as BDD based SMC [5]. However, it is mostly limited to bug finding.

Several works extend BMC for full verification. [3] defined a *Reachability Diameter*, which sets a bound on the number of BMC iterations needed for full verification. This bound, however, is usually hard to compute. Moreover, the bound is very large and therefore the resulting formulas are too large for a SAT solver to handle.

[14] suggests to use *Induction* for full verification. This method uses the BMC check as the induction base. Then, the induction step is checked by checking a second formula. Note that induction works automatically only for simple local properties. For complex properties, the user has to come up with a good inductive invariant. *Proof-Based Abstraction* [12] exploits BMC to determine an abstract model on which BDD-based model checking can be applied. *Interpolation-Based Model Checking* [10] exploits interpolation to compute an over-approximation of the reachable states. The latter work is closest to ours. We compare the two works in a later section, once the details of the methods are presented.

In this work we use *interpolation-sequence* rather than the usual interpolation. Interpolation-sequence has been introduced and used in [8] and [11].

In [8] it is used for computing an abstract model based on predicate abstraction, for software model checking. In [11] interpolation-sequence is used for software model checking and lazy abstraction. While this work uses the interpolation-sequence to compute over-approximations of reachable states and predicates, the computation considers a specific possible execution of the verified software. Our work, on the other hand, uses the interpolation-sequence to gain information on the entire model. Clearly, the two works use different criteria for convergence.

B. Outline

The rest of the paper is organized as follows. In section II we present some background, including interpolation (II-A), model checking (II-B) and bounded model checking (II-C). Our algorithm is described in section III. In section IV we compare our method to the one of [10]. Section V presents our experimental results. Finally, we conclude in section VI.

II. Preliminaries

In this section we present a short description of Interpolation, Model Checking and Bounded Model Checking.

A. Interpolation

Throughout the paper we will denote the value *false* as $\bot$ and the value *true* as $\top$. For a formula X, $\mathcal{L}(X)$ is the set of variables appearing in X. For a set of formulas $\{X_1, \ldots, X_n\}$ we will use $\mathcal{L}(X_1, \ldots, X_n)$ to denote the variables appearing in $X_1, \ldots, X_n$.

Definition 2.1. 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:

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

A SAT solver is a complete decision procedure that given a set of clauses, determines whether the clause set is *satisfiable* or *unsatisfiable*. A clause set is said to be satisfiable if there exists a *satisfying assignment* such that 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* [12]. An interpolant can be produced out of a proof of unsatisfiability [10].

Definition 2.2. Let $\Gamma = \{A_1, A_2, \ldots, A_n\}$ be a set of formulas such that $\bigwedge \Gamma \equiv \bot$. That is $\bigwedge \Gamma = A_1 \wedge \ldots \wedge A_n$ is unsatisifiable. An **interpolation-sequence** for Γ is a set $\{\mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_n\}$ 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 $\mathcal{L}(\mathcal{I}_j) \subseteq \mathcal{L}(A_1, \ldots, A_j) \cap \mathcal{L}(A_{j+1}, \ldots, A_n)$

Computing an interpolation-sequence for a sequence of formulas is done in the following way: 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 $B(i) = \bigwedge_{j=i+1}^{n} A_j$.

Theorem 2.3. *Let $\Gamma = \{A_1, A_2, \ldots, A_n\}$ be a set 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 $B(i) = A_{i+1} \wedge \ldots \wedge A_n$. If $\mathcal{I}_i$ is the interpolant for the pair $(A(i), B(i))$ extracted using Π then the set $\{\top, \mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_{n-1}, \bot\}$ is an interpolant sequence for Γ.*

B. Model Checking

Model checking [6] is an automatic approach to formally verifying that a given system satisfies a given specification. The system is modelled by a Kripke structure and the specification is written in *temporal logic*. Determining whether a model satisfies a given specification is based on exploration of the model's state space in a search for violations of the specification.

Definition 2.4. Given a set of atomic propositions AP, a *Kripke structure* M is the quadruple $M = (S, \mathit{INIT}, \mathit{TR}, L)$ where S is a finite set of states, $\mathit{INIT} \subseteq S$ is a set of initial states and $\mathit{TR} \subseteq S \times S$ is a total transition relation. That is, for every $s \in S$ there exists $s' \in S$ such that $(s, s') \in \mathit{TR}$. Finally, $L : S \longrightarrow \mathcal{P}(AP)$ is the labeling function which associates with every state $s \in S$ the set $L(s)$ of atomic propositions that hold in s.

A *path* in a Kripke structure M is a sequence of states $\pi = (s_0, s_1, \ldots)$ such that for all $i > 0$, $s_i \in S$ 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 $s_0 \in$ *INIT*.

A formula in *Linear Temporal Logic* (LTL) [13] 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 paths in M satisfy f. If there exists a path not satisfying f, this path is defined to be a *counterexample*.

We consider a subset of LTL properties called *safety* properties since Liveness checking can be achieved by the method presented in [2]. In addition, only safety properties of the form AGq are considered where q is a propositional formula. This does not restrict the applicability of our results, since safety properties can be verified using *invariance checking* [9].

Given a property AGq, the model checking problem can then be described as exploring the state space of a model M while checking that q holds for all states.

Let M be a model, *Reach* be the set of reachable states and let $f = AGq$ be a property. If for every $s \in \mathit{Reach}$, $L(s) \models q$ then the property holds in M. On the other hand, if there exist a state $s \in \mathit{Reach}$ such that $L(s) \models \neg q$ 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.

We would sometimes like to represent a Kripke structure by means of propositional formulas. In order to do so, we define the set of state variables, denoted V. Given V where $|V| = n$, a state $s \in S$ is represented by a vector in the set $\{0,1\}^n$ and by that s is a valuation of the state variables in V. A set of states can be represented by a formula over V where the truth assignments represent the states. 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. For some variable v, v' is used to denote the value of v after one time unit. The set of these variables is denoted by 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.

C. Bounded Model Checking

Bounded model checking (BMC) [4] is an iterative process for checking properties up to a given bound. Let M be a Kripke structure and $f = AGq$ 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. In order to search for a counterexample of length k the following propositional formula is built:

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

$\varphi^k_M(f)$ is then passed to a SAT solver which searches for a satisfying assignment. If there exists a satisfying assignment for $\varphi^k_M(f)$ then the property is violated, since there exists a path 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 a given threshold bound k. At each iteration a SAT procedure is invoked.

```
1:  function BMC($M$,$f$,$k$)
2:      $i := 0$
3:      while $i \le k$ do
4:          build $\varphi^i_M(f)$
5:          $result = SAT(\varphi^i_M(f))$
6:          if $result = true$ then
7:              return $cex$        // returning the counterexample
8:          else
9:              $i = i + 1$
10:         end if
11:     end while
12:     return No $cex$ for bound $k$
13: end function
```

Fig. 1: Bounded model checking

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

The main drawback of this approach is the fact that it is not complete. 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.

III. A Novel SAT-based Model Checking Approach

In this section we present our novel SAT-based algorithm for unbounded model checking (UMC). The proposed algorithm explores the state space of the model by means of an over-approximation using BMC and interpolation-sequence. The innovation lies in the way BMC is combined with interpolation-sequence to extract the needed information.

From this point and on, we will use M to denote the Kripke structure representing the model and $f = AGq$ for a propositional formula q, as the property to be verified.

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

A. Revisiting Symbolic Reachability Analysis

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. Further, for every $j \ge 1$, $S_j \wedge \mathit{TR}(V, V') \equiv S_{j+1}$. Once S_j is computed, if it contains states violating q, a counterexample of length i is found and returned. Otherwise, if $S_j \subseteq \bigcup_{i=1}^{j-1} S_i$ then a *fixpoint* has been reached, meaning that all reachable states have been found already. If none violates the property then the algorithm concludes that $M \models f$.

The method presented in this section demonstrates how over-approximated sets, similar to S_i in their characteristics, can be extracted from BMC using an interpolation-sequence generated after each iteration of the BMC loop. These sets will be used to gain knowledge about the reachable states even though the sets are actually an over-approximation of the reachable states. Informally, we will use the notion of *fixpoint* when we can conclude that all *reachable* states in the model have been visited. Note that, the interpolation-sequence exists for a bound N only when there is no counterexample of length N. In case a counterexample exists, BMC returns a counterexample and the interpolation-sequence is not needed.

B. Interpolation-Sequence Based Model Checking

Definition 3.1. A *BMC-partitioning* for φ^N is the set $\Gamma = \{A_1, A_2, \ldots, A_{N+1}\}$ of formulas such that $A_1 = \mathit{INIT}(V^0) \wedge \mathit{TR}(V^0, V^1)$, for every $2 \leq i \leq N$ $A_i = \mathit{TR}(V^{i-1}, V^i)$ and $A_{N+1} = \neg q(V^N)$. Note that $\varphi^N = \bigwedge\limits_{i=1}^{k+1} A_i$ $(= \bigwedge \Gamma)$.

For a bound N, consider a BMC formula φ^N and its BMC-partitioning Γ. In case φ^N is unsatisfiable, its interpolation-sequence is denoted by $\bar{I}^N = (I_0^N, I_1^N, \ldots, I_{N+1}^N)$. Note that the BMC-partitioning for φ^N 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. Consider the formula φ^1 and its BMC-partitioning: A_1, A_2. In case that this formula is unsatisfiable there exists an interpolation-sequence of the form $\bar{I}^1 = (I_0^1 = \top, I_1^1, I_2^1 = \bot)$. By Definition 2.2, $S_1 \subseteq I_1^1$ since $\top \wedge A_1 \Rightarrow I_1^1$. Also, $I_1^1 \wedge \neg q(V^1)$ is unsatisfiable, since $I_1^1 \wedge A_2 \Rightarrow \bot$. Therefore, $I_1^1 \models q$. In the next BMC iteration, consider φ^2 and its BMC-partitioning A_1, A_2, A_3. In case that φ^2 is unsatisfiable, we get $\bar{I}^2 = (\top, I_1^2, I_2^2, \bot)$. 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 $I_2^2 \models q$. Let us define the sets $I_1 = I_1^1 \wedge I_1^2$ and $I_2 = I_2^2$. These sets have the following properties, $S_1 \subseteq I_1$, $S_2 \subseteq I_2$, $I_1 \models q$ and $I_2 \models q$. Moreover, $I_1[V^1 \leftarrow V] \wedge TR(V, V') \Rightarrow I_2[V^2 \leftarrow V']$.

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

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

As a result, the sets $I_1, I_2, \ldots, I_N$ 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 be used to imitate the forward reachability analysis of the model's state-space by means of an over-approximation. This is being 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 I_j for $1 \leq j \leq N$ are updated. If $I_j \Rightarrow \bigvee\limits_{i=1}^{j-1} I_i$ for some $1 \leq j \leq N$, then we conclude that a fixpoint has

```
1: function UpdateReachable($\bar{I}$,$\bar{I}^k$)
2:     $j = 1$
3:     while $(j < k)$ do
4:         $I_j = I_j \wedge I_j^k$
5:         $\bar{I}[j] = I_j$
6:         $j = j + 1$
7:     end while
8:     $\bar{I}[k] = I_k^k$
9: end function
```

Fig. 2: Updating the reachability vector

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 $N+1$.

Informally, the following facts are needed in order to guarantee the correctness of the algorithm described above for checking $M \models f$. For every $1 \leq j \leq N$ we need:

1) I_j should satisfy q.
2) $I_j \wedge \mathit{TR}(V, V') \Rightarrow I_{j+1}$ for $j \neq N$.
3) $S_j \subseteq I_j$.

This means that the algorithm cannot be implemented using $\bar{I}^N$ alone. This is because $\bar{I}^N$ does not satisfy condition (1): while $I_N^N \models q$, I_j^N for $j \neq N$, does not necessarily satisfy q. This can be remedied by conjuncting each I_j^N with I_j^j. However, now condition (2) no longer holds. Taking $I_j = \bigwedge\limits_{k=j}^{N} I_j^k$ results in set with all three properties.

Definition 3.2. If no counterexample of length N or less exists in M, then $I_j = \bigwedge\limits_{k=j}^{N} I_j^k[V^j \leftarrow V]$ where I_j^k is the j-th element in the interpolation-sequence extracted for the BMC-partitioning of φ^k. The *reachability vector* is defined to be $\bar{I} = (I_1, I_2, \ldots, I_N)$.

The algorithms for updating the reachablility vector and checking for a fixpoint are described in Fig 2 and Fig 3, respectively. The complete model checking algorithm using the method described above is given in Fig 4.

It is important to note that a call to UpdateReachability changes the reachability vector. Therefore, the function FixpointReached searches for a fixpoint at any point in $\bar{I}$. Moreover, it is not sufficient to check for inclusion of only the last element of $\bar{I}$. Indeed, if for any $j \leq N$, $I_j \Rightarrow \bigvee\limits_{i=1}^{j-1} I_i$ then all reachable states have been found already. However, the implication $I_N \Rightarrow \bigvee\limits_{i=1}^{N-1} I_i$ might not hold due to additional *unreachable* states in I_N. This is because for all $1 \leq j < N$, I_{j+1} is an approximation of the sets reachable from I_j and not the exact image ($I_j \wedge \mathit{TR}(V, V') \Rightarrow I_{j+1}[V \leftarrow V']$ rather than $I_j \wedge \mathit{TR}(V, V') \equiv I_{j+1}[V \leftarrow V']$).

The following lemmas and definition formalize the above and prove the correctness of the algorithm.

```
function FIXPOINTREACHED($\bar{I}$)
    $j = 2$
    while ($j \leq \bar{I}.length$) do
        $R = \bigvee_{k=1}^{j-1} I_k$
        $\phi = I_j \wedge \neg R$ // Negation of $I_j \Rightarrow R$
        if (SAT($\phi$) == false) then return true
        end if
        $j = j + 1$
    end while
    return false
end function
```

Fig. 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
    $\bar{I} = \varnothing$        // the reachability vector
    while (true) do
        $k = k + 1$
        $result$ = BMC($M, f, k$)
        if ($result == cex$) then
            return $cex$
        end if
        $\bar{I}^k = (\top, I_1^k, \ldots, I_i^k, \bot)$
        UPDATEREACHABLE($\bar{I}$,$\bar{I}^k$)
        if (FIXPOINTREACHED($\bar{I}$) == $true$) then
            return $true$
        end if
    end while
end function
```

Fig. 4: ISB Algorithm

Lemma 3.3. *If M does not have a counterexample of length N, then $S_j \subseteq I_j^N$ for every $1 \leq j \leq N$ and $I_N^N \models q$.*

Proof: M does not have a counterexample of length N. Therefore, the formula φ^N is unsatisfiable. Let $\bar{I}^N$ be the interpolation-sequence for the BMC-partitioning of φ^N. By Definition 2.2, for $j = 1$, $\top \wedge \mathit{INIT}(V^0) \wedge \mathit{TR}(V^0, V^1) \Rightarrow I_1^1$. For each $2 \leq j \leq N$, $I_i^N \wedge \mathit{TR}(V^j, V^{j+1}) \Rightarrow I_{j+1}^N$. Hence, $S_j \subseteq I_j^N$. Definition 2.2 also state that $I_N^N \wedge \neg q(V^N) \Rightarrow \bot$ and therefore $I_N^N \models q$. ■

Lemma 3.4. *If M does not have a counterexample of length N, then $S_j \subseteq I_j$ and $I_j \models q$ for every $1 \leq j \leq N$.*

Proof: For every $j \leq k \leq N$ by Lemma 3.3 $S_j \subseteq I_j^k$ and $I_j^i \models q$. Since I_j is the conjunction of I_j^k for every $j \leq k \leq N$, $S_j \subseteq I_j$ and $I_j \models q$. ■

Lemma 3.5. *Let $\bar{I} = (I_1, I_2, \ldots, I_N)$ be the reachability vector. For every $1 \leq j < N$, $I_j \wedge \mathrm{TR}(V, V') \Rightarrow I_{j+1}[V \leftarrow V']$.*

Proof: By Definition 3.2, $I_j = \bigwedge_{k=j}^{N} I_j^k[V^j \leftarrow V]$. Definition 2.2 implies that for every $j \leq k \leq N$, $I_{j-1}^k \wedge \mathit{TR}(V^{j-1}, V^j) \Rightarrow I_j^k$ we get $I_j \wedge \mathit{TR}(V, V') \Rightarrow I_{j+1}[V \leftarrow V']$. ■

Theorem 3.6. *Assume there is no path of length N or less violating f in M. If there exist $1 < j \leq N$ such that $I_j \Rightarrow \bigvee_{i=1}^{j-1} I_i$, then $M \models f$.*

Proof: By assumption, there is no path in M of length N or less that violates f. We now show that given $I_j \Rightarrow \bigvee_{i=1}^{j-1} I_i$ we can conclude that there is no path of any length violating f. Let $R = \bigvee_{i=1}^{j-1} I_i$. By assumption, $I_j \Rightarrow R$ and by Lemma 3.5, for every $1 \leq i < j$, $I_i \wedge \mathit{TR}(V^i, V^{i+1}) \Rightarrow I_{i+1}$. Thus, $R(V) \wedge \mathit{TR}(V, V') \Rightarrow R(V')$ (1). Moreover, for every $1 \leq i \leq j$ the formula $I_i \wedge \neg q$ is unsatisfiable (since $I_i \models q$ by Lemma 3.4). Hence, $R \wedge \neg q$ is unsatisfiable (2).

By induction we can show that all reachable states are in $R^* = R \vee \mathit{INIT}$. The base case handles an initial state. This holds trivially by the definition of R^*. Now let us assume it holds for all states reachable in k steps. It should be proved for states reachable in $k+1$ steps. Let s_{k+1} be a set reachable in $k+1$ steps from an initial state. Let $\pi = s_0, s_1, \ldots, s_k, s_{k+1}$ be an initial path to s_{k+1}. By the induction hypothesis $s_k \in R^*$. From (1) we know that $R[V \leftarrow V^k] \wedge \mathit{TR}(V^k, V^{k+1}) \Rightarrow R[V \leftarrow V^{k+1}]$. Therefore, $s_{k+1} \in R^*$.

By assumption, $\mathit{INIT} \models q$ since there is no path of length N or less violating f. By that and (2), $R^* \models q$. Thus, the set of reachable states satisfy q which implies that $M \models f$. ■

Lemma 3.7. *Suppose $M \models f$ then there exists a bound N such that $\bar{I} = \{I_1, I_2, \ldots, I_N\}$ and there exists an index $1 < j < N$ such that $I_j \Rightarrow \bigvee_{i=1}^{j-1} I_i$.*

Proof: The set of states S is finite. Let us define $N = j = |S| + 1$. $M \models f$ hence for every $0 \leq k \leq N$, φ^k is unsatisfiable. Thus, the interpolation-sequence $\bar{I}^k$ exists for every $0 \leq k \leq N$ and by that the reachability vector $\bar{I} = \{I_1, I_2, \ldots, I_N\}$ exists. Since $|S| < \infty$ we get $I_j \Rightarrow \bigvee_{i=1}^{j} I_i$. ■

Theorem 3.8. *There exists a path π of length N such that π violates f if and only if ISB terminates and returns cex.*

Proof: Assume that the minimal violating path is of length N. For $N - 1$ there is no path in M violating f. By Theorem 3.6 we get that for every j such that $1 \leq j < N$, $I_j \Rightarrow \bigvee_{i=1}^{j-1} I_i$ does not hold. Therefore, the algorithm cannot terminate by returning $true$ in the first $N-1$ iterations. When the algorithm reaches the N-th iteration, $BMC(M, f, N)$ will return a counterexample and the algorithm terminates. The other direction is immediate. ■

Theorem 3.9. *For every model M and a property $f = AGq$ there exists N such that ISB terminates.*

Proof: If $M \models f$ it follows by Theorem 3.6 and Lemma 3.7 that the algorithm terminates and returns $true$. If there is a path in M that violates f, it follows by Theorem 3.8 that the algorithm terminates and returns cex. ■

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

In the previous section we presented a new method for model checking, the Interpolation-Sequence Based MC (*ISB*) which combines BMC and interpolation-sequence. The closest work to this one is the Interpolation Based MC (IB) described in [10]. Thus, a comparison between the two works is imperative. Other SAT-based methods for full verification have been surveyed in the related work section. Moreover, the work presented in [1] shows a clear advantage to IB over other known methods for verification. We first describe IB, then, we compare the two methods.

The following definition will help us to better describe the differences between the two methods. Recall that the verified property is of the form $f = AGq$.

Definition 4.1. For every $1 \leq j \leq N$, let S_j be the set of states reachable in j steps from the initial states. For a set of states T, if $S_j \subseteq T$ and there is no path of length $(N-j)$ or less violating q, starting from a state $s \in T$, then T is said to be S_j*-approximation* w.r.t N. It is denoted by $S_j \preceq_N T$.

A. Interpolation Based Model Checking (IB)

In [10] McMillan presents a SAT-based model checking algorithm for full verification by combining BMC and Craig's Interpolation [7]. The interpolant is used to compute an over-approximation of the set of reachable states. The algorithm concludes that the property holds and no counterexample exists when a fixpoint is reached during the computation of reachable states and none of the computed states violate the property.

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

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

Clearly $\varphi^{1,k} \equiv A \wedge B$. Assume that $\varphi^{1,k}$ is unsatisfiable. By the interpolation theorem [7], there exists an interpolant J_1^k that follows Definition 2.1:

- J_1^k is over the variables of $\mathcal{L}(A) \cap \mathcal{L}(B)$, namely, V^1.
- $A \Longrightarrow J_1^k$. By that, $S_1 \subseteq J_1^k$.

```
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
        $A = J(V^0) \wedge TR(V^0, V^1)$
        $B = TR(V^1, V^2) \wedge \ldots \wedge TR(V^{k-1}, V^k) \wedge (\bigvee_{j=1}^{j=k} \neg q(V^j))$
        $J = SAT.getInterpolant(A, B)$
        if $J \subseteq R$ then
            return $fixpoint$
        end if
        $R = R \cup J$
        $M'.INIT = J$
    until ($BMC(M', f, 1, k) == cex$)
    return $abort$
end function
```

Fig. 5: Calculating the reachable states using a specific bound

- $J_1^k(V_1) \wedge B$ is unsatisfiable. This means that for every $0 \leq i \leq k-1$, there is no path of length $k-1$ or less starting from J_1^k and violating q.

By the above we get that $S_1 \preceq_k J_1^k$. This procedure is iterated by replacing the initial states in M with the computed interpolant J_1^k. BMC is reinvoked with the same parameters for the modified model $M' = (S, J_1^k[V^1 \leftarrow V], TR, L)$. A new interpolant J_2^k is then extracted. J_2^k satisfies $S_2 \preceq_{k+1} J_2^k$. 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' could 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. For a given bound k, the computation of over-approximated reachable states appears in Fig 5. Note that after L iterations of the main loop in CheckReachable we get L interpolants and for every $1 \leq i \leq L$, $S_i \preceq_{k+L} J_i^k$. If at any point, a counterexample is found on a modified model, CheckReachable is reinvoked with $k+1$. 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 that a real counterexample exists, it will be found during the BMC check on the original model M. A complete description of the algorithms for this method appears in [10].

B. Comparing ISB to IB

The sets of reachable states computed by each method are over-approximated and are different in their characteristics. Therefore, determining which one converges faster is not applicable. A few technical differences exist for ISB and IB. First, the formulas used for the interpolants extraction are different. For a given bound N, ISB uses the formula φ^N while

SMC	ISB	IB
$\{S_1, S_2, \ldots, S_N\}$	$\{I_1, I_2, \ldots, I_N\}$ $S_i \preceq_N I_i$ After checking bounds 1 to N	$\{J_1^1, J_2^1, \ldots, J_N^1\}$ $S_i \preceq_N J_i^1$ N iterations at bound 1 if possible
$\{S_1, \ldots, S_L, \ldots, S_{N+L}\}$	$\{I_1, \ldots, I_L, \ldots, I_{N+L}\}$ $S_i \preceq_{N+L} I_i$ After checking bounds 1 to $N+L$	$\{J_1^N, J_2^N, \ldots, J_L^N\}$ $S_i \preceq_{N+L} J_i^N$, $(1 \leq i \leq L)$ L iterations at bound N if possible

TABLE I: The correlation between the interpolants computed by ISB and IB to the sets computed using SMC

IB uses $\varphi^{1,N}$. Second, the way the interpolants are computed is different. While ISB computes the sets I_j incrementally and refines them after each iteration of BMC as part of the BMC loop, IB recomputes the sets whenever the bound is increased regardless of previous runs using a different BMC call for each iterpolant. ISB can be viewed as an addition to BMC's loop. The addition is the extraction of an interpolation-sequence at each iteration and the check for a fixpoint. Indeed, after N iterations of the BMC loop in ISB, there are N sets of reachable states $I_1, \ldots, I_N$ and $S_j \preceq_N I_j$. On the other hand, IB consists of two nested loops. The outer loop iterates through the bounds while the inner loop calculates the sets of reachable states. If the outer loop is at a higher bound, $N > 1$ and the inner loop performs L iterations then there are L sets of states $J_1^N, \ldots, J_L^N$ such that each has the property $S_i \preceq_{N+L} J_i^N$ $(1 \leq i \leq L)$. Table I summarizes the above.

Having said that, clearly IB can compute, at a given bound, as many sets as needed as long as no counterexample is found (not necessarily a real counterexample). On the other hand, for a bound N, ISB can only compute N sets but it does not require recurrent BMC calls for each bound (only one is needed). By that, 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 the advantage. This fact is reflected in the results.

As was mentioned before, when a counterexample exists the over-approximated sets of reachable states are not needed. For properties that can be falsified there exists a minimal bound N such that for this bound there exists a path that violates the property. Both algorithms have to hit that bound in order to find the counterexample. Here, ISB has a clear advantage over IB. After each BMC run on the original model, IB executes BMC runs on modified models. This means that there are 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 our experiments, these kind of properties always favored ISB.

V. Experimental Results

The proposed algorithm has been checked on various models taken from two of Intel's future CPU designs. The characteristics of the checked models appear in Table II. The 136 properties chosen for the experiments were all real safety properties used to verify the correctness of the designs. The

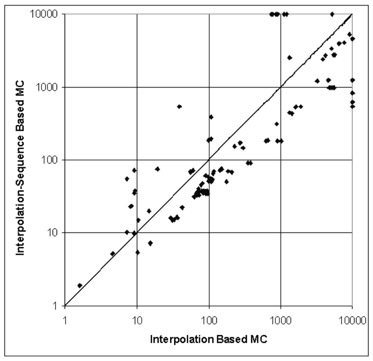

Fig. 6: Runtime on Intel's next micro-architecture releases

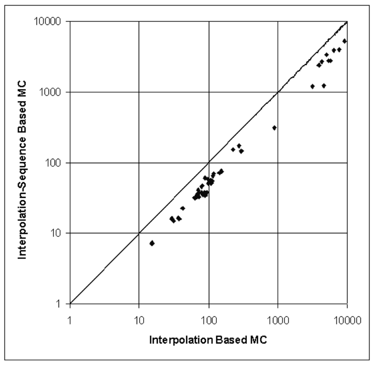

Fig. 7: Runtime of falsified properties.

cone of influence for the properties contains thousands of state variables and tens of thousands of gates and signals. The properties vary in that some are *true* and some are *false*. During all checks, a timeout of 10,000 seconds has been set. If after the given timeout the property cannot be verified nor falsified, the process terminates. If the process terminates with no conclusive answer (*Verified* or *Falsified*), it reports that the result is *Bounded* with the highest bound at which the property is known to be non-violated. Both algorithms were implemented within Intel's verification system using a SAT-based model checker using Intel's in-house SAT solver *Eureka*. Experiments were conducted on systems with a dual core Xeon 5160 processors (Core 2 micro-architecture) running

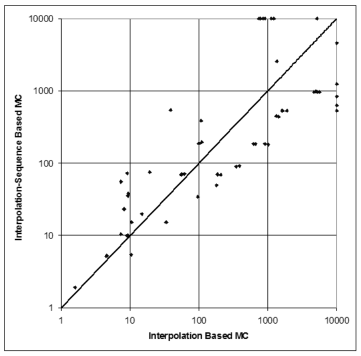

Fig. 8: Runtime of verified properties.

Name	♯ Latches	♯ Inputs	♯ Gates
M_1	3611	3	84570
M_2	4968	2079	133255
M_3	12806	402	89392
M_4	1672	459	11195
M_5	19213	305	146717

TABLE II: Models used for testing

Name	♯ Vars	ISB B	IB B	ISB ♯ I	IB ♯ I	ISB ♯BMC	IB ♯BMC	ISB Time [s]	IB Time [s]
f_1	3406	16	15	136	80	16	80	970	5518
f_2	1753	9	8	45	40	9	40	91	388
f_3	1753	7	6	28	28	7	28	49	179
f_4	1753	16	15	136	94	16	94	473	1901
f_5	3406	6	5	21	13	6	13	68	208
f_6	1761	2	1	3	2	2	2	5	4
f_7	3972	3	1	6	3	3	3	19	14
f_8	2197	3	1	6	3	3	3	10	7
f_9	1629	23	6	276	39	23	39	2544	1340
f_{10}	4894	5	1	15	3	5	3	635	101

TABLE III: Verified properties and their running parameters. ♯ *Vars* stands for the number of state variables in the cone of influence. *B* - bound at convergence, ♯ *I* - number of interpolants computed, ♯BMC - number of calls to BMC algorithm and *Time [s]* - the runtime in seconds

at 3.0GHz (4MB L2 cache) with 32GB of main memory. Operating system running on the system is Linux SUSE.

Fig 6 shows the runtime in seconds of running two interpolation based methods. Each point represents a property from the set of chosen properties. The X axis represents runtime for IB while the Y axis represents the runtime using ISB. We can see that the results vary. All falsified properties (total of 67) favor ISB. Fig 7 shows the runtime for the falsified properties. Fig 8 shows runtime for true properties. There are five properties that can be verified by ISB and not by IB (due to timeout) and two properties that can be falsified using ISB while cannot be falsified using IB. On the other hand, there are seven properties that cannot be verified by ISB but can be verified by IB. The rest of the properties (57 total) are all verified by both algorithms.

A more accurate analysis of the algorithms is shown in Table III that presents running parameters (number of state variables in the cone of influence, bound at convergence, number of interpolants computed, number of calls to BMC and runtime) on various properties for both IB and ISB. For some cases, even though IB converges at a lower bound, and computes less interpolants than ISB, ISB still converges faster by means of runtime. This is due to the fact that BMC calls are computationally heavier than the extraction of the interpolants. Since IB issues more calls to BMC than ISB in these cases, the influence on its runtime is noticeable. Through all our experiments, when convergence for IB could be achieved only at high bounds, ISB always performed better while for convergence at lower bounds, IB is the better performer. This result is supported by the analysis presented in the previous section.

The overall performance, when summarized, are in favor for ISB with 30% improvement in runtime. The total runtime for ISB was 128491 seconds while for IB it was 168745 seconds.

VI. Conclusion

We presented a method that uses interpolation-sequence for SAT-based unbounded model checking. Unlike the interpolation-based model checking algorithm presented in [10], our method does not require successive BMC runs in order to compute an over-approximation of the reachable states. Instead, it is part of the original BMC loop with the addition of interpolation-sequence extraction. It uses a single BMC run for a given bound N to extract information about the reachable states after N transitions or less. The experiments show a clear advantage to ISB when the properties are falsified. In case of true properties, the results vary such that some favor our methods while others favor the method of [10]. The overall performance favored our algorithm.

Further investigation can be made in order to characterize the type of properties (when the properties are true) suitable for each method and by that obtain a better understanding of the difference between the two methods. In addition, we believe that the over-approximated sets of reachable states computed using our method at the N-th iteration can be used to simplify the BMC run for bound $N+1$.

References

[1] N. Amla, X. Du, A. Kuehlmann, R. P. Kurshan, and K. L. Mcmillan. An analysis of sat-based model checking techniques in an industrial environment. In *CHARME'05*.

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

[3] A. Biere, A. Cimatti, E.M. Clarke, O. Strichman, and Y. Zhu. *Bounded Model Checking*, volume 58 of *Advances in Computers*. Elsevier, 2003.

[4] A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu. Symbolic Model Checking Without BDDs. In *TACAS'99*.

[5] Jerry R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *LICS'90*.

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

[7] William Craig. Linear reasoning. a new form of the herbrand-gentzen theorem. *J. Symb. Log.*, 22(3), 1957.

[8] R. Jhala and K.L. McMillan. Interpolant-Based Transition Relation Approximation. In *CAV'05*.

[9] O. Kupferman and M.Y. Vardi. Model checking of safety properties. In *CAV'99*.

[10] K.L. McMillan. Interpolation and SAT-based Model Checking. In *CAV'03*.

[11] K.L. McMillan. Lazy Abstraction with Interpolants. In *CAV'06*.

[12] K.L. McMillan and N. Amla. Automatic abstraction without counterexamples. In *TACAS'03*.

[13] Amir Pnueli. The temporal logic of programs. In *FOCS'77*.

[14] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a sat-solver. In *FMCAD'00*.