Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 174 (2007) 31–43
www.elsevier.com/locate/entcs

Interpolant Learning and Reuse in SAT-Based Model Checking

Joao Marques-Silva[1]

School of Electronics and Computer Science
University of Southampton, Southampton, UK

Abstract

Bounded Model Checking (BMC) is one of the most paradigmatic practical applications of Boolean Satisfiability (SAT). The utilization of SAT in model checking has allowed significant performance gains and, as a consequence, a large number of commercial verification tools now include SAT-based model checkers. Recent work has provided SAT-based BMC with completeness conditions, and this is generally referred to as unbounded model checking (UMC). Among the existing approaches for SAT-based UMC, the utilization of interpolants is among the most effective. Despite their success, interpolants have only been used for identifying a fixed point of the set of reachable states. This paper extends the utilization of interpolants in SAT-based model checking. This is achieved by observing that, under reasonable assumptions, interpolants can be *reused*, i.e. computed interpolants can be reused at later stages of the model checking process. The paper develops conditions for validity of interpolant reuse. In addition, the paper outlines a new fixed point condition, alternative to the existing interpolant-based fixed point condition. Preliminary practical experience on interpolant learning and reuse is reported.

Keywords: Boolean Satisfiability, Bounded Model Checking, Interpolants.

1 Introduction

The utilization of Boolean Satisfiability (SAT) in Model Checking has been the subject of extensive research in recent years. The main result of this effort has been a number of fairly competitive incomplete and complete SAT-based model checking algorithms [3,4,5,20,21,26,27]. Moreover, SAT-based model checking has also been rapidly adopted by industry, and a number of vendors have included SAT-based Model Checking in their tools.

The utilization of SAT in model checking was first proposed in the form of Bounded Model Checking (BMC) [3], where a counterexample is searched for increasing unfoldings of a finite state automaton. The original BMC work has been shown to be extremely useful for finding counter-examples but, unless the recurrence

[1] Email:jpms@ecs.soton.ac.uk

1571-0661
doi:10.1016/j.entcs.2006.12.021

(or the reachability) diameter of the automaton is known [2], the BMC procedure is incomplete.

Different solutions have been proposed for ensuring the completeness of BMC [26,5,17,16,21], the most promising of which is arguably based on the utilization of interpolants [21].

This paper reviews the utilization of interpolants in SAT-based unbounded model checking and proposes the learning and reuse of computed interpolants with the purpose of allowing increased search pruning for subsequent calls to the SAT solver during the model checking process. The paper shows that different interpolants can be computed and used in different contexts. Moreover, the paper outlines a fixed point condition alternative to the one proposed in‘[21].

We note that the main objectives of the paper are to investigate the conditions for interpolant reuse, and to propose an alternative interpolant-based fixed point condition. However, our experimental results suggest that interpolant reuse may not yield improvements on industrial examples. A more effective implementation, as well as a more careful selection of which interpolants to reuse, may yield more effective interpolant reuse.

The paper is organized as follows. The next section provides a necessarily brief perspective on SAT solvers and related concepts. Afterwards, Section 3 reviews SAT-based model checking, including bounded and unbounded model checking. Section 4 develops conditions for reusing learnt interpolants. Initial practical experience is summarized in Section 5 and Section 6 concludes the paper.

2 Preliminaries

Propositional formulas are defined over finite sets of Boolean variables $X = \{x_1, x_2, \ldots\}$, $W = \{w_1, w_2, \ldots\}$, X_1, X_2, etc., where each variable can be assigned value 1 (TRUE) or 0 (FALSE). In what follows propositional formulas are represented by $\psi_1, \psi_2, \ldots$. When relevant other subscripts can be used, e.g. ψ_a, ψ_b, etc. For specific cases, letters and names representing predicates are also used for denoting the associated propositional formulas, examples include I, T, F, P, Q and BMC. When referring to propositional formulas in conjunctive normal form (CNF), we associate with each propositional formula $\psi_a(X_a)$ a CNF formula $\varphi_a(X_a, U_a)$, where U_a denotes a set of auxiliary Boolean variables. Formulas in CNF consist of a conjunction of clauses (each clause represented by ω_i), where each clause consists of a disjunction of literals (represented by l_j). When used in an expression, a propositional formula ψ is interpreted as a predicate, and so corresponds to $\psi = 1$. Similarly, when the propositional formula $\neg\psi$ is used in an expression, it corresponds to $\psi = 0$.

We consider model checking of LTL safety properties G ψ_S. A finite state automaton $M = (I, T, F)$ is assumed, where I is a predicate defined on state variables, T is the state transition relation, and F represents the failing property (i.e. $F = \neg\psi_S$), defined on state variables. Moreover, the utilization of predicates I, T or F assumes an underlying automaton $M = (I, T, F)$. As mentioned above, for simplicity the propositional formulas associated with these predicates are repre-

sented with the same letters, I, T and F.

It will also be necessary to map propositional formulas from one set of variables to another set of variables. The notation $\psi(Y/Y_k)$ is used to denote that the propositional formula ψ, defined over the set of variables Y, is mapped into the set of variables Y_k. Moreover, state variables are preferably represented as set Y, Y_k when referring to the state variables in time step k. Boolean circuit variables are preferably represented as sets X or W, respectively X_k and W_k for variables in time step k, and finally auxiliary variables used in the CNF representation are preferably represented as sets W or Z.

2.1 Boolean Satisfiability Solvers

The remarkable evolution of Boolean Satisfiability (SAT) solvers over the last decade [19,23,14] has motivated the application of SAT in model checking. The most effective SAT solvers are based on backtrack search [9] and share a number of key techniques, including:

- Unit clause rule, also referred as Boolean constraint propagation, that consists of the identification of implied variable assignments [10].
- Clause learning, consisting of learning new clauses in presence of conflicts during the execution of backtrack search. A few techniques related with clause learning are the utilization of unique implication points (UIPs) [19], and non-chronological backtracking [19].
- Memory efficient lazy data structures [23].
- Adaptive branching heuristic, usually derived from the VSIDS heuristic [23].
- Utilization of search restarts [15], by using some completeness criterion.

Because modern backtrack search SAT solvers learn clauses, it is straightforward to track all the learned clauses, and use these clauses for constructing a resolution refutation (or unsatisfiability proof) of the original formula [29].

2.2 SAT-Related Concepts

This subsection addresses a number of byproducts of modern SAT solvers, which are required for the utilization of interpolants in SAT-based model checking. For this purpose, we review *proof traces*, *unsatisfiable cores* and *unsatisfiability proofs*.

As mentioned above, modern SAT solvers learn clauses. For unsatisfiable instances, the original clauses and the learned clauses can be used for generating a resolution-based unsatisfiability proof [29]. Modern SAT solvers can be instructed for generating a *proof trace*, which associates with each learned clause ω, all the clauses that explain the creation of ω [29].

Given a proof trace Γ, where the final traced clause is the empty clause $\bot$, we can identify, in linear time on the size of the proof trace, a subset of the original set of clauses which is itself unsatisfiable [29]. This subset is referred to as an *unsatisfiable core*.

Moreover, and given a proof trace Γ, generated by a SAT solver, it is possible to create a resolution-based unsatisfiability proof in time and size linear on the size of the proof trace.

Definition 2.1 [Unsatisfiability Proof [21]] A proof of unsatisfiability Π for a set of clauses φ is a directed acyclic graph (V_Π, E_Π), where V_Π is a set of clauses, such that:

- For every $\omega \in V_\Pi$, either
  - $\omega \in \varphi$, and ω is a root, or
  - ω has two predecessors, ω_1 and ω_2, such that ω is the resolvent of ω_1 and ω_2 (the variable v used for resolving ω_1 with ω_2 is referred to as the *pivot* variable of the resolution step), and
- the empty clause $\bot$ is the unique leaf.

2.3 Craig Interpolants

Assume a propositional formula $\psi_A(Y, X)$, defined over the sets of variables Y and X, and a propositional formula $\psi_B(Y, W)$, defined over the sets of variables Y and W. If $\psi_A(Y, X) \land \psi_B(Y, W)$ is unsatisfiable, then there exists a propositional formula $\psi_P(Y)$, defined over the set of variables Y, such that $\psi_A(Y, X) \rightarrow \psi_P(Y)$ is a tautology and $\psi_B(Y, W) \land \psi_P(Y)$ is unsatisfiable. The propositional formula $\psi_P(Y)$ is referred to as an *interpolant* for $\psi_A(Y, X)$ and $\psi_B(Y, W)$ [8]. Recent work has shown that an interpolant can be constructed in linear time on the size of a resolution refutation of $\psi_A(Y, X) \land \psi_B(Y, W)$ [25].

In what follows we outline McMillan's interpolant construction [21], even though Pudlák's construction [25] could also be considered. Regarding the propositional formulas $\psi_A(Y, X)$ and $\psi_B(Y, W)$, and associated CNF formulas, respectively $\varphi_A(Y, X, U)$ and $\varphi_B(Y, W, V)$, variables in set Y are referred to as *global* variables, whereas variables in sets X and U are *local* to $\varphi_A(Y, X, U)$, and the variables in sets W and V are *local* to $\varphi_B(Y, W, V)$. Further, let $g(\omega)$ denote the literals corresponding to global variables in clause ω.

Definition 2.2 [Interpolant [21]] Let (φ_A, φ_B) be a pair of clause sets and let Π be a proof of unsatisfiability of $\varphi_A \cup \varphi_B$, with leaf vertex $\bot$. For each vertex $\omega \in V_\Pi$, let ψ_ω be a Boolean formula, such that:

- If ω is a root then
  - if $\omega \in \varphi_A$ then $\psi_\omega = g(\omega)$,
  - else $\psi_\omega = \textsc{True}$
- else, let ω_1, ω_2 be the predecessors of ω and let v be their pivot variable
  - if v is local to φ_A, then $\psi_\omega = \psi_{\omega_1} \lor \psi_{\omega_2}$,
  - else $\psi_\omega = \psi_{\omega_1} \land \psi_{\omega_2}$

The Π-interpolant of (φ_A, φ_B), denoted $\textsc{Itp}(\Pi, \varphi_A, \varphi_B)$ is $\psi_\bot$.

The interpolant $\textsc{Itp}(\Pi, \varphi_A, \varphi_B)$ has size linear on the size of the unsatisfiability proof [25,21].

Algorithm 1 Organization of BMC

BMC($M = (I, T, F)$, λ, ι, μ)
1 $j \leftarrow 0$
2 $k \leftarrow \lambda$
3 **while** $k \leq \mu$
4 **do** $\varphi \leftarrow \textsc{Cnf}(\textsc{Bmc}_j^k(M), W)$
5 **if** $\textsc{Sat}(\varphi)$
6 **then return false** ▷ Found counterexample
7 $k \leftarrow k + \iota$
8 **return true**

3 SAT-Based Model Checking

This section overviews the work on using SAT in model checking, emphasizing the initial work on Bounded Model Checking (BMC) and the more recent work on Unbounded Model Checking (UMC).

3.1 Bounded Model Checking

The generic Boolean formula associated with SAT-based BMC is the following [2,3,27]:

$$\textsc{Bmc}_j^k(M) = I(Y_0) \wedge \left(\bigwedge_{0 \leq i < k} T(Y_i, Y_{i+1}) \right) \wedge \left(\bigvee_{j \leq i \leq k} F(Y_i) \right) \tag{1}$$

This formula represents the unfolding of the state machine for k time steps, where $I(Y_0)$ represents the initial state, $T(Y_i, Y_{i+1})$ represents the transition relation between states Y_i and Y_{i+1}, and $F(Y_i)$ represents the failing property in time step i. Given the Boolean formula $\textsc{Bmc}_j^k(M)$, it is straightforward to generate a CNF formula φ, by applying Tseitin's [28] or the structure preserving [24] transformations, and by using additional auxiliary Boolean variables. This formula can then be evaluated by a SAT solver.

The typical organization of BMC for safety properties is illustrated in Algorithm 1. The details regarding the sets of variables associated with each propositional formula are omitted, but are clear from the context. Experimental evidence has confirmed SAT-based BMC to be an extremely competitive technique, that has been widely applied in industrial settings [2,7,12].

In order to describe the work on UMC and the reusing of interpolants, the following predicates are extensively used:

$$\textsc{Unfold}_r^s(M) = I(Y_{-r}) \wedge \left(\bigwedge_{-r \leq i < s} T(Y_i, Y_{i+1}) \right) \tag{2}$$

$$\textsc{Tran}_s^t(M) = \bigwedge_{s \leq i < t} T(Y_i, Y_{i+1}) \tag{3}$$

$$\text{PROP}_v^u(M) = \left(\bigwedge_{u \leq i < u+v} T(Y_i, Y_{i+1}) \right) \wedge \left(\bigvee_{u \leq i \leq u+v} F(Y_i) \right) \tag{4}$$

Hence, we can express the BMC formula in terms of these predicates:

$$\begin{aligned} \text{BMC}_j^k(M) &= \text{UNFOLD}_0^j(M) \wedge \text{PROP}_{k-j}^j(M) \\ &= \text{UNFOLD}_0^0(M) \wedge \text{TRAN}_0^j(M) \wedge \text{PROP}_{k-j}^j(M) \end{aligned} \tag{5}$$

3.2 Unbounded Model Checking

A key difficulty with BMC is its inability for proving that there is no counterexample for a given safety property G ψ_S. Unless the recurrence (or the reachability) diameter [2] of an automaton is known, it is not possible to establish the value of the upper bound (UB) used in Algorithm 1; in the case the recurrence diameter is known, BMC becomes complete. In general the recurrence diameter of an automaton is not known, and so BMC is incomplete. As a result, in recent years different approaches have been proposed for ensuring the completeness of SAT-based model checking. We refer to these approaches as *Unbounded Model Checking* (UMC) [20,21]. The first UMC SAT-based approach was proposed by Sheeran et al. in [26] and extended in [4]. Additional techniques include [5,20,13,22,21,16]. The induction-based approach of Sheeran et al. [26] requires unfolding the state machine for the largest simple path between any two reachable states in the worst case. However, the largest simple path between any two reachable states can be exponentially larger than the reachability diameter. Alternatively, Chauhan et al. [5] and Glusman et al. [13] propose refinement techniques based on elimination of false counterexamples. Another approach based on iterative abstraction is proposed by Gupta et al. in [16]. More recently, McMillan and Amla [22] propose the utilization of proof-based abstraction, even though the proposed approach is not fully SAT-based. According to experimental data from [21], the utilization of interpolants in SAT-based model checking is the most effective approach. We detail the utilization of interpolants in the next section.

3.3 Interpolant-Based Unbounded Model Checking

Recent work on SAT-based Unbounded Model Checking has addressed the utilization of interpolants [21], with quite promising experimental results. This section reviews McMillan's interpolant-based UMC algorithm [21].

The definition of the BMC proposition formula is modified slightly with respect to (1):

$$\begin{aligned} \text{PREF}_l(M) &= I(Y_{-l}) \wedge \left(\textstyle\bigwedge_{-l \leq i < 0} T(Y_i, Y_{i+1}) \right) \\ &= \text{UNFOLD}_l^0(M) \end{aligned} \tag{6}$$

$$\begin{aligned} \text{SUFF}_j^k(M) &= \left(\textstyle\bigwedge_{0 \leq i < k} T(Y_i, Y_{i+1}) \right) \wedge \left(\textstyle\bigvee_{j \leq i \leq k} F(Y_i) \right) \\ &= \text{TRAN}_0^j(M) \wedge \text{PROP}_{k-j}^j(M) \end{aligned} \tag{7}$$

Algorithm 2 UMC Algorithm

```
UMC(M = (I, T, F))
 1  k ← 0
 2  if SAT(I ∧ F)
 3     then return false ▷ Counterexample found
 4          while true
 5               do status = CHECKFIXPOINT(M, k)
 6                  if status = false
 7                     then return false ▷ Counterexample found
 8                     else  if status = true
 9                                then return true ▷ Property proved
10                  k ← k + 1 ▷ Unfold further
```

Hence, the BMC formula becomes:

$$\text{BMC}_j^k(M) = \text{PREF}_1(M) \wedge \text{SUFF}_j^k(M) \tag{8}$$

The above equation corresponds to the one proposed by McMillan [21], where the separation between prefix and suffix identifies the set of variables with respect to which interpolants are to be computed.

The SAT-based model checking algorithm can be organized into two main phases: a BMC loop, where the circuit is unfolded, and a fixed point checking step, that checks for the existence of a counterexample and where the existence of a fixed-point is tested. Observe that the second phase requires the iterative computation of interpolants until a fixed-point is reached or a true or (possibly) false counterexample is identified. The organization of the BMC loop is outlined in Algorithm 2, whereas the organization of fixed point checking step is outlined in Algorithm 3.

For the BMC loop there is no upper bound on the number of unfoldings, since the algorithm is now complete. The increment of k is not required to be 1. In fact, feedback from the fixed point checking procedure can be used for increasing k by values larger than 1 [18]. In addition, observe that the fixed point checking procedure consists of iterative computation of interpolants, where for iteration m the interpolant represents an abstraction of the reachable states in m time steps [21]. At each iteration of the UMC fixed point checking procedure, the existence of a fixed-point is tested. The fixed-point is reached when the abstraction of the reachable states in m time steps contains only states already included in the abstractions of the reachable states in less than m time steps. Finally, observe that the algorithm sets $j = 0$, because interpolants are computed with respect to Y_0.

4 Interpolant Learning and Reuse

This section develops conditions for reusing computed interpolants, and consists of two main parts. Conditions for interpolants representing over-approximations of the set of reachable states, and conditions for interpolants representing over-

Algorithm 3 Fixed point identification in SAT-based UMC

```
CheckFixpoint(M = (I, T.F), k)
 1  R ← I
 2  while true
 3        do M' ← (R, T, F)
 4           A ← Cnf(Pref_1(M'), W_1)
 5           B ← Cnf(Suff_0^k(M'), W_2)
 6           (isSat, Γ) ← Sat(A ∪ B)
 7           if isSAT
 8              then if R = I
 9                      then return false
10                      else  return abort
11           ▷ A ∪ B is unsat
12           Π ← UnsatProof(Γ)
13           P ← Itp(Π, A, B)
14           R' ← P(Y/Y_0)
15           C ← Cnf(¬R, W_3)
16           D ← Cnf(R', W_4)
17           (isSat, Γ) ← Sat(C ∪ D)
18           if not isSAT
19              then return true
20           R ← R ∨ R'
```

approximations of the set of states satisfying the failing property. We should note that the work on interpolant reuse is largely motivated by previous (and successful) work on clause reuse [27]. Clause reuse has been used extensively in BMC and is widely regarded as a key technique [27,12].

The main motivation is to develop conditions which enable computed interpolants to be reused. Hence, the following definition is used extensively.

Definition 4.1 A Boolean formula ψ_N is said to be *usable* for Boolean formula ψ_B iff $\psi_B \rightarrow \psi_N$.

Hence, ψ_N preserves satisfiability of the original formula and so we get the following straightforward result:

Proposition 4.2 *Let ψ_N be usable for ψ_B. Then ψ_B is satisfiable iff $\psi_B \wedge \psi_N$ is satisfiable.*

In order to generalize the computation of interpolants, equation (5) is modified as follows:

$$\textsc{Bmc}_j^k(M) = \textsc{Unfold}_0^k(M) \wedge \textsc{Prop}_j^k(M) \tag{9}$$

Observe that the new equation differs from (5) and (8). In equation (9) the failing property is checked for only in the last j time steps for an unfolding of $k + j$ time

steps [2]. (This approach is also used for example in [7,26,12].) For simplicity we assume $j = 0$; generalization for $j > 0$ is simple.

The standard interpolants used in [21] are referred to as *direct interpolants*. It is also possible to compute *reverse interpolants* by exchanging the sets A and B in the definition of interpolant. *Direct* interpolants are computed as described in McMillan's work [21] (see also the previous section), but relaxing the 1 time step unfolding for A. For computing an interpolant after r time steps from I and $t = k-r$ time steps from F, the propositional formulas for A and B become:

$$A = \textsc{Cnf}(\textsc{Unfold}_0^r(M), W_1) \tag{10}$$

$$B = \textsc{Cnf}(\textsc{Tran}_{k-t}^k(M) \wedge \textsc{Prop}_0^k(M), W_2) \tag{11}$$

The interpolant computed with A and B above will be denoted P_t^r. It is also possible to compute an interpolant by replacing I with another interpolant P_v^u:

$$A = \textsc{Cnf}(P_v^u(Y_0) \wedge \textsc{Tran}_0^r(M), W_1) \tag{12}$$

$$B = \textsc{Cnf}(\textsc{Tran}_{k-t}^k(M) \wedge \textsc{Prop}_0^k(M), W_2) \tag{13}$$

And the new interpolant is denoted P_t^{u+r}.

Consequently, P_t^r, with $r, t \geq 0$, denotes the direct interpolant computed with a (possibly virtual) unfolding of r time steps from the initial state, and t time steps until the failing property is checked for. Hence, P_t^r represents an *over-approximation* of the set of states reachable in r time steps and an *under-approximation* of the set of states which do not satisfy the failing property in t time steps.

Reverse interpolants are computed by interchanging the definitions of A and B in (10) and (11), and will be denoted by Q_t^r. Hence, Q_t^r, $r, t \geq 0$, denotes the reverse interpolant computed with an unfolding of r time steps from the initial state, and (possibly virtual) t time steps until the failing property is checked for. Hence, Q_t^r represents an *under-approximation* of the set of states that are not reachable in r time steps and an *over-approximation* of the set of states which satisfy the failing property in t time steps. From (10) and (11) we obtain:

$$A = \textsc{Cnf}(\textsc{Tran}_{k-t}^k(M) \wedge \textsc{Prop}_0^k(M), W_3) \tag{14}$$

$$B = \textsc{Cnf}(\textsc{Unfold}_0^r(M), W_4) \tag{15}$$

The interpolant computed with A and B above will be denoted Q_t^r. It is also possible to compute an interpolant by replacing F with another interpolant Q_v^u:

$$A = \textsc{Cnf}(\textsc{Tran}_{k-t}^k(M) \wedge Q_v^u(Y_k)) \tag{16}$$

$$B = \textsc{Cnf}(\textsc{Unfold}_0^r(M), W_4) \tag{17}$$

And the new interpolant is denoted Q_{t+v}^r.

Given the definitions of direct and reverse interpolants, we can now establish conditions for interpolant reuse in SAT-based model checking.

Theorem 4.3 *Let* $\textsc{Bmc}_j^k(M)$ *be given by (9), and direct interpolants* P_t^r *be computed with (10) and (11). Then the following holds:*

[2] The automaton is assumed to be stuttering closed [6,21].

(i) $P_t^r(Y_r)$ *is usable for* $\text{BMC}_j^k(M)$, *with* $t \geq 0$ *and* $0 \leq r \leq k$.

(ii) $\neg P_t^r(Y_{k-t})$ *is usable for* $\text{BMC}_j^k(M)$, *with* $r \geq 0$ *and* $0 \leq t \leq k$.

Proof.

(i) If $\text{BMC}_j^k(M)$ is satisfiable, then $\text{UNFOLD}_0^r(M)$, with $r \leq k$ is also satisfiable and Y_r represents a state reachable in r time steps. By definition, $P_t^r(Y_r)$ represents an over-approximation of the states reachable in r time steps. Hence, $P_t^r(Y_r)$ holds for any assignment to the variables in Y_r representing a state reachable in r time steps. Thus, $\text{BMC}_j^k(M) \rightarrow P_t^r(Y_r)$, with $r \leq k$. By definition, $P_t^r(Y_r)$ is usable for $\text{BMC}_j^k(M)$, with $r \leq k$. Observe that there is no upper bound on the value of t.

(ii) Observe that $P_t^r(Y_{k-t})$ represents an under-approximation of the states which do not satisfy the failing property in t time steps. Hence, $P_t^r(Y_{k-t}) \rightarrow \neg\text{BMC}_j^k(M)$ with $t \leq k$. Consequently, $\text{BMC}_j^k(M) \rightarrow \neg P_t^r(Y_{k-t})$. By definition, $P_t^r(Y_{k-t})$ is usable for $\text{BMC}_j^k(M)$, with $t \leq k$. Observe that there is no upper bound on the value of r.

□

Theorem 4.4 *Let* $\text{BMC}_j^k(M)$ *be given by (9), and reverse interpolants* Q_t^r *be computed with (14) and (15). Then the following holds:*

(i) $Q_t^r(Y_{k-t})$ *is usable for* $\text{BMC}_j^k(M)$, *with* $r \geq 0$ *and* $0 \leq t \leq k$.

(ii) $\neg Q_t^r(Y_r)$ *is usable for* $\text{BMC}_j^k(M)$, *with* $t \geq 0$ *and* $0 \leq r \leq k$.

Proof. The proof is similar to the proof for Theorem 4.3.

(i) If $\text{BMC}_j^k(M)$ is satisfiable, then $\text{TRAN}_{k-t}^k(M) \wedge \text{PROP}_k^k(M)$, with $t \leq k$ is also satisfiable and Y_{k-t} represents a state that satisfies the failing property in t time steps. By definition, $Q_t^r(Y_{k-t})$ represents an over-approximation of the states that satisfy the failing property in t time steps. Thus, $\text{BMC}_j^k(M) \rightarrow Q_t^r(Y_{k-t})$, with $t \leq k$. By definition, $Q_t^r(Y_{k-t})$ is usable for $\text{BMC}_j^k(M)$, with $t \leq k$. Observe that there is no upper bound on the value of r.

(ii) Observe that $Q_t^r(Y_r)$ represents an under-approximation of the states that are unreachable r in time steps. Hence, $Q_t^r(Y_r) \rightarrow \neg\text{BMC}_j^k(M)$, with $r \leq k$. Consequently, $\text{BMC}_j^k(M) \rightarrow \neg Q_t^r(Y_r)$. By definition, $Q_t^r(Y_r)$ is usable for $\text{BMC}_j^k(M)$, with $r \leq k$. Observe that there is no upper bound on the value of t.

□

Remark 4.5 Even though we describe the most general setting for learning and reusing interpolants, the specific interpolants computed in the standard interpolant-based fixed point condition [21] are also usable according to the conditions of Theorems 4.3 and 4.4. Hence, interpolant reuse can be readily integrated in a standard interpolant-based UMC flow.

Remark 4.6 The conditions of Theorems 4.3 and 4.4 can be used in *any* BMC/UMC setting, independently of whether a fixed point is used and whether

Instance	w/o interpolants	w/ interpolants
6-bit counter	1.51	5.29
7-bit counter	16.38	61.03
8-bit counter	236.90	784.81
I1	7.08	7.11
I2	31.36	36.96
I3	38.36	60.60
I4	52.45	58.25
I5	150.54	157.81

Table 1
Results with and without interpolant reuse

it is based on interpolants.

Remark 4.7 It is straightforward to conclude that reverse interpolants can be used for developing a fixed point condition alternative to the one of [21]. Algorithm 3 can easily be adapted for using reverse interpolants, computed one time step from the time step at which the property is checked for. Similarly to image computation approaches in BDD-based symbolic model checking, the advantages of this alternative fixed point condition are expected to depend on the actual automaton.

5 Experimental Results

The practical experience reported in this section respects a preliminary SAT-based model checking prototype. The prototype represents interpolants as Reduced Boolean Circuits (RBCs) [1]. The backend SAT solver is MiniSAT [11]. The implementation of interpolant computation is still preliminary and, currently, different interpolants do not share structure. Even though each interpolant is generated with the rules of [1], each different interpolant is maintained with a separate RBC manager, and so common nodes among different interpolants are not shared. Moreover, the utilization of interpolants was evaluated in a standard BMC loop, and so interpolants were solely computed for search pruning purposes. Iinterpolants were computed with respect to the last time step and reused in the last time step. As a result, reused interpolants serve for preventing sets of unwanted states to be reached.

Table 1 shows preliminary results from interpolant reuse. The first set of instances represent standard counters, for which counterexample exists. The second set of instances represent industrial problem instances, for which a counterexample also exists. As can be concluded, the utilization of interpolants does not yield improvements to the run times. For the first set of (artificial) examples the results

are worse than for the second set of (industrial) examples. As mentioned above, the setup for the utilization of interpolants is certainly not the most adequate. We considered a simple BMC loop, where interpolants are solely used for search pruning purposes. The reuse of interpolants in a UMC setting is expected to provide more competitive results, since the interpolants have be computed for checking the fixed point condition.

6 Conclusions and Future Work

This paper develops conditions for learning and reusing of interpolants in SAT-based model checking. Computed interpolants can be used for requiring states from a set of states or for preventing states from a set of states. Besides interpolant reuse, an alternative fixed-point condition is also proposed, based on reverse (as opposed to direct) interpolants.

The preliminary results are not positive, albeit the implementation is still very preliminary. Moreover, the experimental setup chosen was not beneficial for the reuse of interpolants. Instead of an interpolant-based UMC algorithm, where interpolants need to be computed, our experiments consisted of a standard BMC loop, where computed interpolants were solely used for search pruning purposes.

A few drawbacks of the current implementation have been identified. Examples include the lack of structure sharing between different interpolants, and the fact that interpolants were computed solely for interpolant reuse and not for checking the existence of a fixed point. Integration of these improvements is expected to yield more promising results for interpolant reuse.

Acknowledgments

This work has been partially supported by European project IST-033709 VERTIGO.

References

[1] P. A. Abdulla, P. Bjesse, and N. Eén. Symbolic reachability analysis based on SAT solvers. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2000.

[2] A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu. *Advances in Computers*, chapter Bounded Model Checking. Academic Press, 2003.

[3] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 193–207, March 1999.

[4] P. Bjesse and K. Claesen. SAT-based verification without state space traversal. In *International Conference on Formal Methods in Computer-Aided Design*, 2000.

[5] P. Chauhan, E. Clarke, J. Kukula, S. Sapra, H. Veith, and D. Wang. Automated abstraction refinement for model checking large state spaces using SAT based conflict analysis. In *International Conference on Formal Methods in Computer-Aided Design*, 2002.

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

[7] F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Y. Vardi. Benefits of bounded model checking at an industrial setting. In *International Conference on Computer-Aided Verification*, 2001.

[8] W. Craig. Linear reasoning: A new form of the Herbrand-Gentzen theorem. *Journal of Symbolic Logic*, 22(3):250–268, 1957.

[9] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the Association for Computing Machinery*, 5:394–397, July 1962.

[10] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, July 1960.

[11] N. Een and N. Sorensson. An extensible SAT solver. In *Sixth International Conference on Theory and Applications of Satisfiability Testing*, May 2003.

[12] N. Een and N. Sorensson. Temporal induction by incremental SAT solving. In *Workshop on Bounded Model Checking*, volume 89 of *ENTCS*, 2003.

[13] M. Glusman, G. Kamhi, S. Mador-Haim, R. Fraer, and M. Vardi. Multiple-counterexample guided iterative abstraction refinement. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, April 2003.

[14] E. Goldberg and Y. Novikov. BerkMin: a fast and robust SAT-solver. In *Design, Automation and Test in Europe Conference*, pages 142–149, March 2002.

[15] C. P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *National Conference on Artificial Intelligence*, pages 431–437, July 1998.

[16] A. Gupta, M. Ganai, Z. Yang, and P. Ashar. Iterative abstraction using SAT-based BMC with proof analysis. In *International Conference on Computer-Aided Design*, November 2003.

[17] H.-J. Kang and I.-C. Park. SAT-based unbounded symbolic model checking. In *Design Automation Conference*, pages 840–843, June 2003.

[18] J. P. Marques-Silva. Improvements to the implementation of interpolant-based model checking. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, October 2005.

[19] J. P. Marques-Silva and K. A. Sakallah. GRASP: A new search algorithm for satisfiability. In *International Conference on Computer-Aided Design*, pages 220–227, November 1996.

[20] K. L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *International Conference on Computer-Aided Verification*, July 2002.

[21] K. L. McMillan. Interpolation and SAT-based model checking. In *International Conference on Computer-Aided Verification*, 2003.

[22] K. L. McMillan and N. Amla. Automatic abstraction without counterexamples. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, April 2003.

[23] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Engineering an efficient SAT solver. In *Design Automation Conference*, pages 530–535, June 2001.

[24] D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, September 1986.

[25] P. Pudlák. Lower bounds for resolution and cutting planes proofs and monotone circuit computations. *Journal of Symbolic Logic*, 62(3):981–998, 1997.

[26] M. Sheeran, S. Singh, and G. Stalmarck. Checking safety properties using induction and a SAT solver. In *International Conference on Formal Methods in Computer-Aided Design*, 2000.

[27] O. Strichman. Tuning SAT checkers for bounded model checking. In *International Conference on Computer-Aided Verification*, July 2000.

[28] G. S. Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic, Part II*, pages 115–125, 1968.

[29] L. Zhang and S. Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Design, Automation and Test in Europe Conference*, pages 10880–10885, March 2003.