Boosting Interpolation with Dynamic Localized Abstraction and Redundancy Removal

GIANPIERO CABODI, MARCO MURCIANO, SERGIO NOCCO, and STEFANO QUER
Politecnico di Torino

SAT–based Unbounded Model Checking based on Craig Interpolants is often able to overcome BDDs and other SAT–based techniques on large verification instances. Based on refutation proofs generated by SAT solvers, interpolants provide compact circuit representations of state sets, as they abstract away several nonrelevant details of the proofs. We propose three main contributions, aimed at controlling interpolant size and traversal depth. First of all, we introduce interpolant–based dynamic abstraction to reduce the support of computed interpolants. Subsequently, we propose new advances in interpolant compaction by redundancy removal. Finally, we introduce interpolant computation exploiting circuit quantification, instead of SAT refutation proofs. These techniques heavily rely on an effective application of the incremental SAT paradigm. The experimental results proposed in this paper are specifically oriented to prove properties, rather than disproving them, i.e., they target complete verification instead of simply hunting bugs. They show how this methodology is able to stretch the applicability of interpolant–based Model Checking to larger and deeper verification instances.

Categories and Subject Descriptors: B.6.3 [**Logic Design**]: Design Aids

General Terms: Algorithms, Verification

Additional Key Words and Phrases: Interpolant, abstraction, redundancy removal

ACM Reference Format:

Cabodi, G., Murciano, M., Nocco, S. and Quer, S. 2007. Boosting interpolation with dynamic localized abstraction and redundancy removal. ACM Trans. Des. Autom. Electron. Syst. 13, 1, Article 3 (January 2008), 20 pages. DOI = 10.1145/1297666.1297669 http://doi.acm.org/10.1145/1297666.1297669

1. INTRODUCTION

Symbolic model checking [Burch et al. 1994] is a method for verifying temporal properties of finite state systems which relies on a symbolic representation of sets, typically through Binary Decision Diagrams (BDDs) [Bryant 1986].

Author's address: Politecnico di Torino, Dipartimento di Automatica e Informatica, C.so Duca degli Abruzzi 24, 10129 Turin, Italy.

By contrast, bounded model checking (BMC) [Biere et al. 1999] can falsify temporal properties using Boolean satisfiability (SAT). Unlike BDD-based methods, BMC focuses on finding bugs of bounded length k, subsequently increasing the bound to search for longer traces. Given a design and a correctness property, it generates a Boolean formula by unrolling the model for k time frames, so that the formula is satisfiable if and only if there is a counterexample of length k. Using a modern SAT solver, this method is efficient in producing counterexamples [Bjesse et al. 2001] and it has been proved to be more robust and scalable than symbolic model checking methods based on BDDs. However, although BMC can find bugs in larger designs than other BDD-based methods, the verification process cannot be complete, since it only guarantees the correctness of a property for the given bound.

To extend the methodology to full verification, a novel approach was proposed by McMillan [2003], who used Craig interpolants for unbounded Model Checking. Modern SAT solvers are able to generate proofs of unsatisfiability, that is, refutation proofs. In our case, a refutation proof demonstrates that there is no counterexample up to a certain number of steps. Such a proof implies nothing about the truth of the property, but it can be used to produce overapproximations of the system reachable state space. Given this chance, it is possible to exploit refutation proofs of (unsatisfied) BMC runs, to compute overapproximate state sets. The approach can be viewed as an iterative refinement of proof-based abstractions to narrow down a proof to relevant facts. Its convergence is bound to the diameter of the state graph, and experimental tests on large circuits showed impressive results in several cases.

Our initial experiences with this Model Checking approach showed that interpolants can be very effective whenever they are able to converge at low depths (number of reachability iterations), and their sizes stay within tractable ranges. Unfortunately, this is not always the case. We thus propose three main contributions, under the general idea of compacting interpolant sizes and reducing traversal depths:

—A dynamic abstraction procedure, based on identifying optimal subsets of the state variables to be safely abstracted before each reachability step. The proposed abstraction generally reduces the number of iterations, thus anticipating the convergence of the process.
—An interpolant circuit optimization technique, based on redundancy removal under observability and External Don't Care conditions. The strategy we propose is applied on top of other optimization techniques [Mishchenko 2005] and effectively exploits the benefits of incremental SAT.
—An interpolant computation based on circuit quantification rather than on refutation proofs. We introduce this technique (combined with dynamic abstraction) in backward reachability, where circuit–based quantification is more effective.

Our techniques are implemented by successfully exploiting incremental SAT, that is, by properly formulating a set of SAT problems which are able to share large portions of their clause database. The main idea of incremental SAT is to share the clause database across different SAT solver runs, in such a way that the knowledge learned by the SAT engine, that is, the set of conflict clauses, can be safely reused.

Our experimental results focus on correct properties, and they show that the proposed methods improve the original one by making it faster, more robust and scalable. We show experiments where we are able to complete difficult instances, not achievable with previous techniques.

1.1 Related Work

SAT solvers have been applied in unbounded model checking in several ways.

For instance, they have been used in hybrid methods to detect fix-points, while the quantifier elimination required for image computation is performed by other means. Williams et al. [2000] adopted Boolean Expression Diagrams (BEDs), for quantifiers removal. Abdulla et al. [2000] adopted Reduced Boolean Circuits (RBCs), that is, a variant of BEDs, to represent formulas on which they performed existential quantifiers elimination through substitution, scope reduction, etc.

To extend BMC to full verification, a completeness check was proposed by Sheeran et al. [2000]. The authors provided a proof of correctness for safety properties based on the longest loop-free path between states. Unfortunately, the longest loop-free path can be exponentially longer than the diameter of the reachable state space (for example the longest loop-free path for n-bit counters is 2^n while the reachable diameter is 1).

To overcome this problem, McMillan [2002] adopted quantifier elimination through the enumeration of SAT solutions (*all-solutions SAT*). In this approach a SAT procedure is used to enumerate all state cube solutions, where efficient state pruning is achieved by the introduction of the so-called "blocking clauses." For each new solution, a blocking clause, that is, a clause representing the negation of the new solution, is added to the original problem database. As a consequence, each blocking clause prevents the SAT-solver from running into the same solution twice.

Following the same idea, Kang and Park [2003] used a two-level minimizer to reduce the growth of the CNF database due to the addition of new blocking clauses. However, since the number of required enumerations is bounded below by the size of a two-level prime and irredundant cover of the entire state set, quantifier elimination based on cube-by-cube enumeration tends to be expensive.

Ganai et al. [2004] extended the previous approaches by using "circuit cofactoring." The authors adopted a circuit graph model to represent state sets, and they used circuit-based cofactoring to capture a large set of states in every SAT enumeration step.

All the above methods potentially converge faster than Sheeran et al. [2000], yet they share the common issue of possible exponential state set representations.

A few recent works follow the interpolant idea. Silva [Marques-Silva 2005] proposes some effective optimizations for interpolant compaction and interpolant–based traversals. Transition relation abstraction, oriented to software model checking, is explored by McMillan in McMillan and Jhala [2005].

1.2 Outline

Section 2 introduces background notions on SAT–based and Craig interpolant model checking. Section 3 presents our first contribution, that is, interpolant-based dynamic abstraction. Section 4 describes circuit optimization by redundancy removal. Section 5 overviews the overall model checking procedure. Section 6 discusses the experiments we performed. Section 7 concludes with some summarizing remarks.

2. BACKGROUND

2.1 Model and Notation

In our notation, B indicates the Boolean space. Symbols $\wedge$, $\vee$, $\neg$, $\Rightarrow$, and $\Leftrightarrow$ are used for Boolean conjunction (AND), disjunction (OR), negation (NOT), implication, and coimplication respectively.

The automata we address are usually represented implicitly by Boolean formulas. The state space of the automaton is defined by an indexed set of Boolean variables $V = \{v_1, \dots, v_n\}$. Given a variable v_i, v_i/x_j indicates the substitution of v_i with x_j. Given a set of variables V, $V\backslash v_i$ indicates the same set V without the variable v_i. A state S is a corresponding vector $(s_1, \dots, s_n)$ of Boolean values. A state predicate P is a Boolean formula over V. We will write $P(X)$ to denote $P|_{v_i/x_i}$, that is, P with each v_i replaced by x_i. We also indicate next state variables with $V' = \{v'_1, \dots, v'_n\}$.

For our purposes, an automaton is a triple A = (I, T, F), where I is the set of initial states, T is the relation between the states, and F is the target set of states. Notice that, in our model, F is the negation of the property we want to verify.

As $T(V, V')$ is the set containing all the couples (current state V – next state V') such that there is at least an input value that lets the system evolve from state V to state V', the definition of an image [Burch et al. 1994] is straightforward. In fact, the image computation $\textsc{Img}(T, S)$ considers only those pairs of states in which the current one belongs to the current set S and returns the corresponding next state set S'.

$$\text{To}(V') = \textsc{Img}(T(V, V'),\ S(V)) = \exists_V (S(V) \wedge T(V, V')).$$

An overapproximate image $\textsc{Img}^+(T, S)$ is computed in such a way that, for every T and state set S, $\textsc{Img}(T, S)$ implies $\textsc{Img}^+(T, S)$. Notice that $\textsc{Img}^+(T, S)$ is not unique, as it depends on the approximation level.

With abuse of notation, in the rest of this paper, we make no distinction between the characteristic function of a set and the set itself.

2.2 Bounded Model Checking

SAT-based Bounded Model Checking (BMC) [Biere et al. 1999] considers only paths of bounded length k and builds a propositional formula f that is satisfiable iff there is a counter-example (a path from I to F) of the same length.

More specifically, a run of A (of length k) is a sequence of states $s_0, \ldots, s_k$ such that $I(s_0)$ is true, $T(s_i, s_{i+1})$ is true for all $0 \leq i < k$, and $F(s_k)$ is true. By introducing a new set of variables $X_i = \{x_{i1}, \ldots, x_{in}\}$, we will call

$$T_0^k(X_0, \ldots, X_k) = \bigwedge_{0 \leq i < k} T(X_i, X_{i+1}).$$

In BMC, the existence of a run is translated into a Boolean satisfiability problem for $0 \leq i \leq k$ as:

$$\textsc{Bmc}_0^k = I(X_0) \wedge T_0^k(X_0, \ldots, X_k) \wedge F(X_k).$$

If $\textsc{Bmc}_0^k$ is unsatisfiable, the property has no counterexample of length k, and a refutation proof RP can be produced. The technique works well in falsification and partial verification, whereas full verification is usually achieved by BMC with longer and longer bounds, possibly enhanced with inductive proofs.

2.3 Craig Interpolants in Model Checking

Given two inconsistent formulas A and B ($A \wedge B = 0$), an interpolant C is a formula such that:

(1) It is implied by A
(2) It is inconsistent with B, i.e., $C \wedge B$ is unsatisfiable
(3) It is expressed over the common variables of A and B.

Starting from a refutation proof (RP) of $A \wedge B$, an interpolant $C = \textsc{Itp}\,(A,\ B)$ is an AND/OR circuit that can be computed from RP. Albeit the computation can be performed in linear time with respect to the size of RP, the size of RP itself can be exponential compared to A and B.

A k-adequate overapproximate image $\textsc{Img}^+_{Adq}(T, S, F, k)$ is an $\textsc{Img}^+(T, S)$ that does not intersect any state on paths of length k to F. $\textsc{Img}^+_{Adq}(T, S, F, k)$ is undefined iff

$$\textsc{Img}(T(X_0, X_1), S(X_0)) \wedge T_1^{k+1}(X_1, \ldots, X_{k+1}) \wedge F(X_{k+1}) \neq 0.$$

A possible way of computing $\textsc{Img}^+_{Adq}$ is interpolation:

$$\textsc{Img}^+_{Adq}(T, S, F, k) = \textsc{Itp}(S(X_0) \wedge T(X_0, X_1),\ T_1^{k+1}(X_1, \ldots, X_{k+1}) \wedge F(X_{k+1})).$$

An image is called adequate if it is k-adequate for any k, that is, no path of any length can lead from a state within the image to states in F. Since the model is finite, a k-adequate image is adequate if $k \geq d$, where d is the diameter of the state transition graph.

McMillan [2003] proposed an effective fully SAT-based Unbounded Model Checking approach, exploiting interpolants generated from proofs of unsatisfiability. The algorithm is sketched in Figure 1.

While InterpolantMC is the entry point of the algorithm, routine FiniteRun takes care of the interpolant-based overapproximated traversal. The latter function may end up with three possible results:

—It returns "reachable" if it proves F reachable in k steps, hence the property has been disproved.

```
InterpolantMC (I, T, F)
    k = 0
    do
        res = FiniteRun (I, T, F, k)
        k = k + 1
    while (res = undecided)

FiniteRun (I, T, F, k)
    if (Sat($\text{Bmc}_0^{k+1}$))
        return (reachable)
    R = I
    while (true)
        To = $\text{Img}^{+}_{Adq}$ (T, R, F, k)
        if (To = undefined)
            return (undecided)
        if (To $\Rightarrow$ R)
            return (unreachable)
        R = R $\vee$ To
```

Fig. 1. Interpolant based verification.

—It returns "unreachable" if the approximate traversal using the Img^{+}_{Adq} image computation reached a fix-point. In this case the property has been proved.

—It returns "undecided" if F is intersected by the over-approximate state sets.

McMillan [2003] proved that the previous algorithm is sound and complete. In synthesis, let us assume I and F mutually unreachable: if $k < d$, a k-adequate set can produce a non-k-adequate image. In this case, the "undecided" result is returned and k is increased. Otherwise, when $k \geq d$, Img^{+}_{Adq} is adequate: this means that we only accumulate the (approximate) reachable state set, eventually reaching the fix-point and terminating.

According to Marques-Silva [2005], k can be incremented by the depth of the last FiniteRun execution to avoid a quadratic number of image computations.

2.4 Incremental SAT

Incremental SAT is a technique oriented to run a sequence of (related) SAT problems without destroying the learned clauses database. It was originally proposed by Whittemore et al. [2001], and then by Eén and Sörensson [2003b] for inductive problems of increasing depths. In the latter approach (the one we use, as opposed to the technique implemented in zChaff [Moskewicz et al. 2001]), the main idea is to forbid clause removal, just allowing the addition of new clauses across different SAT solver runs. This enables the reuse of the knowledge learned by the solver during all the previous instances, that is, to preserve conflict clauses. In order to virtually support clause removal, any temporary function f_i, required by the problem, is expressed in relational form, that is, $\rho_i \Rightarrow f_i$. The related SAT run is called by assuming $\rho_i = 1$ when f_i has to be taken into account, and $\rho_i = 0$ when it has to be ruled out. Incremental SAT can provide dramatic improvements, as long as the set of related

SAT problems share a great amount of learning, like the case of BMC runs with increasing depths.

3. INTERPOLATION AND ABSTRACTION

Let us start from the observation that interpolation basically provides an over–approximate image, k-adequate with respect to a given target set F or, in other words, it guarantees the unreachability of F through paths of length k.

Two desirable effects are related to the use of Craig interpolants in verification:

—They achieve overapproximation and variable existential quantification (i.e., image computation) at the same time.

—The overapproximation process is based on the property under check, so that the abstraction concentrates on relevant facts.

At the opposite side of the spectrum, problems may arise due to:

—*Deep traversals.* Let d^+ be the sequential depth of the overapproximate FINITERUN procedure, and d the diameter of the state transition graph. In general, $d^+ < d$; that is, overapproximation "bypasses" long state transition paths and converges faster than exact reachability. Nevertheless, $d^+ > d$ is also possible, whenever interpolants trigger long walks over chains of "unreachable" but "adequate" states.

—*Interpolant size.* It is generally difficult to predict the size of interpolants, as they come from refutation proofs, and their initial size is strongly related to the SAT solving process. It has been observed [McMillan 2003; Marques-Silva 2005] that interpolants are often highly redundant. Furthermore, different refutation graphs are possible for a given proof, producing different interpolant circuits. McMillan [McMillan and Jhala 2005] explored a full range of refutation graph transformations, and their resulting interpolant circuits, basically looking for tighter overapproximations.

3.1 Dynamic Abstraction

In order to face the indicated problems, we follow the general idea of abstraction by localization reduction [Kurshan 1994], as a well-known (and successful) way of analyzing overapproximate behaviors, by removing some details and/or mutual dependencies, such that the proof is still achievable.

Following McMillan [2003], we resort to proof-based abstraction, rather than counterexample-based refinement. Although interpolation itself can be viewed as an abstraction technique, we explicitly introduce localization reduction within the two nested iterations of interpolant-based model checking. We specifically consider overapproximation by state variable (existential) abstraction and we apply it:

—In the initial part of function FINITERUN (see Figure 1). For a given k value, we provide an abstract transition relation which is able to guarantee the consistency of the initial k-bound BMC instance. Notice that, in this case, unlike standard abstraction-refinement methods [Lin et al. 2003; Chauhan

```
DynAbstrBMC (S, T, F, k)
    // Exact BMC with bound k is UNSAT
    $\alpha = \{\}$
    for-each next state var $x'_i$
        $\alpha = \alpha \cup x'_i$
        $\Gamma = \textsc{Abstr}(T, \alpha)$
        // k–bound BMC check with $\Gamma$
        if ($\textsc{Sat}(S(X_0) \wedge \Gamma_0^k(X_0, \ldots, X_k) \wedge F(X_k))$)
            // BMC fails: Undo abstraction
            $\alpha = \alpha \setminus x'_i$
    return ($\alpha$)
```

Fig. 2. k–Step BMC safe dynamic abstraction.

```
DynAbstrAdq (S, Γ, F, k)
    // $k$–adequacy check for exact image
    if ($\textsc{Sat}(S(X_0) \wedge \Gamma(X_0, X_1) \wedge \Gamma_1^{k+1}(X_1, \ldots, X_{k+1}) \wedge F(X_{k+1})$)
        return (undecided)
    $\alpha = \{\}$
    for-each next state var $x'_i$
        $\alpha = \alpha \cup x'_i$
        $\widehat{\Gamma} = \textsc{Abstr}(\Gamma, \alpha)$
        // $k$–adequacy check for abstract image with $\widehat{\Gamma}$
        if ($\textsc{Sat}(S(X_0) \wedge \widehat{\Gamma}(X_0, X_1) \wedge \Gamma_1^{k+1}(X_1, \ldots, X_{k+1}) \wedge F(X_{k+1})$)
            // Not adequate: Undo abstraction
            $\alpha = \alpha \setminus x'_i$
    return ($\alpha$)
```

Fig. 3. k-Adequate dynamic abstraction.

et al. 2002], we do not refine previous abstractions. This task is performed by procedure DynAbstrBMC (see Figure 2).

—Within the traversal loop of function FiniteRun, at the $\textsc{Img}^{+}_{Adq}$ level, as a pre-processing step of standard interpolation. In this case, the abstraction is accomplished by procedure DynAbstrAdq (see Figure 3), where we consider abstraction itself as an interpolation process, able to produce k-adequate over-approximations.

Both our abstractions can be viewed as localization reductions, targeted to safely remove some state variables, still yielding unsatisfiability of a given SAT run.

We called our abstraction process *dynamic*, as we recompute transition relation abstractions on the fly. These two levels are analyzed in Section 3.2 and 3.3.

3.2 BMC-Based Abstraction

The aim of this abstraction is to generate an abstract transition relation, to be used within an entire FiniteRun over-approximate traversal. We do not adopt an abstraction-refinement scheme, as localization reduction is controlled by a k-bound BMC instance. Abstractions are tighter “by construction,” since we loop through increasing k values.

Let α be a subset of (present or next state) variables to be abstracted from the transition relation (we will show how to select α in Section 3.4). Given the α set, we compute an abstract transition relation Γ by localization abstraction through the ABSTR operator:

$$\Gamma(X, X') = \textsc{Abstr}(T(X, X'), \alpha).$$

The general formulation of the above abstraction is expressed by existential quantification:

$$\textsc{Abstr}(T(X, X'), \alpha) = \exists_\alpha T(X, X').$$

Due to the complexity of existential quantification over a circuit representation, we explicitly perform it only for next state variables quantification from functional transition relations. Whenever the transition relation comes from a circuit, it can be expressed as:

$$T = \bigwedge_i (x'_i \Leftrightarrow \delta_i(X)),$$

where $\delta_i(X)$ indicates a next state function as a formula over present state variables X. Existential quantification of any x'_i variable is achieved in a straightforward way by simply removing the corresponding component in T. In all other cases we defer quantification to the subsequent SAT solver run, by renaming the α variables to free, that is, pseudo-input, variables.

The BMC-based abstraction is done in such a way that Γ does not change the result of k-bound BMC:

$$\textsc{Sat}(I(X_0) \wedge \Gamma_0^k(X_0, \ldots, X_k) \wedge F(X_k)) = \textsc{Sat}\ (\textsc{Bmc}_0^k).$$

This abstraction is a preliminary step for an entire FINITERUN call. The intuition under this choice is that we exploit an abstract transition relation that still guarantees a correct k-bound BMC. As FINITERUN is called with increasing bounds the abstraction becomes tighter at each new call.

3.3 Adequate Abstraction

Given a formulation similar to the previous one, for the α variables and the Γ abstract transition relation, we operate a new dynamic abstraction at each image computation step. Starting from Γ, we look for new α variables to dynamically abstract away under the control of k-adequacy checks. Let us call $\widehat{\Gamma}$ the corresponding abstract transition relation:

$$\widehat{\Gamma}(X, X') = \textsc{Abstr}(\Gamma(X, X'), \alpha).$$

The performed abstraction is k-adequate iff

$$\begin{gathered}\textsc{Img}(\Gamma(X_0, X_1), S(X_0)) \wedge \Gamma_1^{k+1}(X_1, \ldots, X_{k+1}) \wedge F(X_{k+1}) = 0 \\ \Rightarrow \\ \textsc{Img}(\widehat{\Gamma}(X_0, X_1), S(X_0)) \wedge \Gamma_1^{k+1}(X_1, \ldots, X_{k+1}) \wedge F(X_{k+1}) = 0.\end{gathered}$$

Once $\widehat{\Gamma}$ has been computed, the FINITERUN procedure can:

—Compute exact images adopting $\widehat{\Gamma}$. As $\widehat{\Gamma}$ is a k-adequate overapproximation of Γ, this operation, per se, gives overapproximate results and it is an alternative to SAT based interpolation.

—Consider the abstraction as a pre-processing step of interpolant-based image computations that further approximate starting from refutation proofs of k-adequacy SAT checks.

We expect two main advantages from such an abstraction. First of all, we reduce the set of support, and hopefully the circuit size, of the image state set. Secondly, we reduce the number of iterations of the reachability process.

Intuitively, we start from the observation that minimal support is often related to lower circuit size. SAT solvers are unlikely to achieve minimum support of the refutation proofs by themselves, as the heuristics driving them are based on different targets. Although we generally expect that refutation proofs do not include variables that are not relevant for a proof, we don't expect minimal support interpolants.

A specifically targeted abstraction, paying some overhead, is more likely to get to such a minimal support interpolant. Moreover, shorter traversal depths come from abstraction by variable quantification, that projects the state transition graph over a subspace. This tends to produce more compact graphs and shorter state transition paths.

3.4 Selecting the Abstraction

Let us consider now the process of finding the α set of variables. Our target is to obtain a minimum-size interpolant. Unfortunately (to the best of our knowledge) Minimum-Cost Satisfiability Tools (MinCostSAT) [Manquinho and Marques-Silva 2002] have been studied, whereas the reduction of the unsatisfiability proof (from which an interpolant is extracted) has not been taken into account.

This is an optimization problem that we can face by trading off performance for optimality. Iterating through all candidate subsets, in order to find BMC-safe (or k-adequate) abstractions, and selecting the optimal one, is clearly too expensive. We chose a suboptimal greedy approach that incrementally builds the α set by looping through all next state variables.

Figures 2 and 3 show the procedures, for the BMC-based and adequacy-based abstractions, respectively.

In Figure 2, the generic x_i' variable is tentatively added to the abstraction set α (initially empty), and validated by a BMC instance. The result is sensitive to the chosen variable order.

A similar approach is used in Figure 3. In this case α is validated for k-adequacy. Given a state set S and a transition relation T, k-adequacy of their image with respect to F is tested by a SAT check. This is done by a preliminary test with Γ (to filter out undefined interpolants), then repeated for all candidate abstract transition relations $\widehat{\Gamma}$.

The process requires a linear number of SAT calls (related to the amount of state variables), which can still be too expensive. Incremental SAT is crucial for an efficient implementation, that is, to reduce the cost of the iterated SAT checks. For the sake of simplicity, we omit describing the incremental approach in DynAbstrBMC, whereas we give a brief formalization for the DynAbstrAdq case.

Let X be the set of common state variables shared by $\widehat{\Gamma}$ and Γ_1^{k+1}. In order to relationally express several $\widehat{\Gamma}$ (with different α sets), we use *wire cutting* instead of explicit existential quantification. We generate two sets of fresh variables $\widehat{x}$ and γ. Variable $\widehat{x}_i$ replaces x_i in $\widehat{\Gamma}$, so that now $\widehat{\Gamma}$ and Γ_1^{k+1} do not share state variables any more. Conditional variable sharing is captured by an additional term

$$\bigwedge_i (\gamma_i \Rightarrow (\widehat{x}_i \Leftrightarrow x_i)),$$

where $\gamma_i = 1$ means the existence of a wire connection between $\widehat{x}_i$ and x_i, $\gamma_i = 0$ means no connection (no relation) between them. In other words, $\gamma_i = 0$ is equivalent to the existential quantification of x_i in $\widehat{\Gamma}$ (as $\widehat{x}_i$ is an unconstrained free variable). We now express $\widehat{\Gamma}$ as follows:

$$\Gamma(x/\widehat{x}) \wedge \bigwedge_i (\gamma_i \Rightarrow (\widehat{x}_i \Leftrightarrow x_i)).$$

We now load all terms (including Γ in relational form) once and for all into the SAT solver (in terms of CNF clauses). The generic SAT call, to test k-adequacy of a given α set, is done incrementally. To do that, we first assume a proper values for all γ variables; that is, any γ_i is set to 0 if x_i belongs to the α set, 1 otherwise:

$$assumption = \bigcup_i (\gamma_i \Leftrightarrow \neg(x_i \in \alpha))$$

After that, each call to the SAT solver over a predicate P is constrained with this assumption, which we indicate as $\textsc{Sat}_{assumption}(P)$. Notice that *assumption* represents a set of literals instead of a set of variables; that is, we add and remove from *assumption* signed variables like v_i and $\neg v_i$.

A further improvement we adopted (omitted here for sake of simplicity) is the introduction of a time bound on SAT calls. This improves scalability, but leaves undecided k-adequacy tests. The undecided variables are further refined by a second loop, starting from the assumption that all those variables are abstracted. This set is then refined until the abstraction is proved k-adequate.

4. CIRCUIT COMPACTION

Circuit compaction is another way to attack the size of interpolant representations (interpolants are potentially highly redundant), as well as circuit-based state set representations in general.

Among the available possibilities, we concentrated on redundancy removal, whose basic steps consist in identifying circuit nodes replaceable by constant nodes. Although testing techniques have been successfully applied in this framework [Berkelaar and van Eijk 2002], as redundancies are related to untestable stuck-at faults, we resorted to SAT-based algorithms [Mishchenko and Brayton 2006b; Mishchenko and Brayton 2006a]. We expected major improvements from the recent developments in the SAT technology, the incremental approach, and an aggressive use of don't care based simplifications.

First of all, we looked at incremental SAT, as redundancy removal is essentially a set of SAT checks over several circuit nodes, to prove/disprove candidate substitutions with constant nodes. Given a circuit f (identified by the set of its outputs), and a subset N of its nodes (to be checked for redundancy) we build up a miter M for each redundancy test, comparing the outputs of the original f with the new ones, produced through the injection of a set of constants. More formally, being $n_i \in N$ the node to be checked for 0–redundancy, we do the following check:

$$\neg\text{S{\scriptsize AT}}(f \neq f(n_i/0)) \quad \Rightarrow \quad n_i \ is \ 0-redundant.$$

Where the $f(n_i/0)$ expression means that the n_i node is assigned the 0 constant value, and the dependency from its fan-in circuit is removed. We proceed dually for 1-redundancy. In order to apply incremental SAT over the set N of nodes, we basically need to express every constant injection in terms of variable decisions only. Subsequently, for each $n_i \in N$, we generate two new variables: c_i, selecting the proper constant value (0 or 1), and γ_i, selecting between nonredundancy ($\gamma_i = 1$, n_i connected to its original fan-in) and redundant behavior ($\gamma_i = 0$, n_i fed by c_i). Then, all the n_i nodes in N are replaced by ITE (γ_i, n_i, c_i). Now each redundancy check over N can be achieved incrementally, through proper assumptions on the γ and c variables.

We experimentally found that it was not worthwhile to work with too large N sets, for example, the whole set of f nodes. Different redundancies are not only related/implied each–other (so it might be unnecessary to prove all of them), but a given circuit can be recomputed and highly compacted, just after a subset of ITE redundancies is known. So we decided to work with "clusters" of nodes for the N set. Nodes are added to the cluster N following their topological depth under a size threshold constraint. Limiting the size of N allows us to control the overall number of variables for the SAT problem, and to rebuild f after redundancies over N are found, yet exploiting incremental SAT for all checks on a given N.

Our redundancy removal procedure is shown in Figure 4. Let f be the function (represented by a combinational network) we want to optimize, and Care be an external care condition. The RedRemoval procedure loops through successive redundancy removal iterations. First of all, procedure SelectCluster considers a set of candidate nodes of size Th. It sorts nodes by topological level, and it first considers nodes near the inputs. Then, the previously selected nodes are checked for redundancy. Then, at the end of each iteration, function RecomputeAIG optimize f exploiting the redundancies found. Finally, the procedure iterates by considering a wider set of nodes (Th is doubled) until no redundancy is found, or the upper bound for Th (maxTh) on tested nodes is reached.

A relevant contribution for stronger optimizations is provided by a careful and effective use of don't cares, which quite often allows finding far more redundancies than with the original circuit alone.

We exploit an External Don't Care (EDC) set optimization with the idea that already reached states can work as EDC conditions for further reachability

```
RedRemoval ($f$, Care)
    Th = minTh
    do
        $\widehat{f} = f$
        N = SelectCluster ($\widehat{f}$, Th)
        $assumption = \{\}$
        for each node $n_i \in$ N
            replace $n_i$ with ITE ($\gamma_i$, $n_i$, $c_i$)
            // Initially no redundancy assumed
            $assumption = assumption \cup \gamma_i$
        $M = (\widehat{f} \neq f) \wedge Care$
        for each node $n_i \in$ N
            // Try redundancy with constant 0
            $assumption = (assumption \backslash \gamma_i) \cup (\neg\gamma_i)$
            $assumption = assumption \cup (\neg c_i)$
            if ($\neg$Sat$_{assumption}$ (M))
                // Redundancy found
                Set $n_i = 0$ in $f$
            else
                // Try redundancy with constant 1
                $assumption = (assumption \backslash (\neg c_i)) \cup c_i$
                if ($\neg$Sat$_{assumption}$ (M))
                    // Redundancy found
                    Set $n_i = 1$ in $f$
                else
                    // Redundancy not found
                    $assumption = (assumption \backslash (\neg\gamma_i)) \cup \gamma_i$
        RecomputeAIG ($f$)
        Th = Th $\cdot$ 2
    while ((N $\leq$ maxTh) $\wedge$ (Redundancy Found))
```

Fig. 4. Redundancy removal under external care conditions.

steps (*Care* = ¬*EDC* = ¬*Reached*). The previous simplification was key for advances in BDD-based reachability, where the "restrict" cofactor [Coudert et al. 1989; Coudert and Madre 1991], was used to minimize the size of reachable state sets feeding image operators [Cho et al. 1990]. We then implicitly take into consideration both the Input Don't Care (IDC) set and the Output (or Observability) Don't Care (ODC) set, within our miter model. Historically, most practical redundancy removal approaches are quite often limited to the IDC set, as redundancy removal under IDCs preserves the circuit function: a node is replaced with a constant node whenever it is constant under all allowed inputs. Practically speaking, the miter includes a comparison output exactly on the redundant node n_i.

ODC conditions take into account more aggressive node transformations: they allow different values, for a given node, from the original and the transformed circuit, provided that they are not observable on the checked outputs. The main problem with this kind of redundancies is that two ODC-based redundancies are not mutually unrelated, that is, any accepted simplification modifies the circuit for the next check. This makes simulation-based pre-processing less effective, and efficient SAT processing crucial. Furthermore, the overall result depends on the order of redundancy checks. Incremental SAT is key to

```
FiniteRun_{Fwd} (I, T, F, k)
    if (Sat(Bmc$_0^k$))
        return (reachable)
    $\alpha$ = DynAbstrBMC (I, T, F, k)
    $\Gamma$ = Abstr($T, \alpha$)
    R = From = I
    while (true)
        $\beta$ = DynAbstrAdq (From, $\Gamma$, F, k)
        if ($\beta$ = undecided)
            return (undecided)
        $\widehat{\Gamma}$ = Abstr($\Gamma, \beta$)
        To = Img$^+_{Adq}$ ($\widehat{\Gamma}$, From, F, k)
        if (To $\Rightarrow$ R)
            return (unreachable)
        From = RedRemoval (To, $\neg$R)
        R = R $\vee$ To
```

Fig. 5. Forward traversal.

effectively handle ODC conditions, as it allows different checks to exploit common ODC conditions (and the related conflict clauses). Concerning the order of checks, we currently rely on topological heuristics, though we expect that further improvements are possible.

5. FORWARD/BACKWARD VERIFICATION

It is well known that forward or backward reachability can be selectively applied to get better results, as each one can overcome the other depending on the model under check. The choice of using one approach instead of the other depends on the depth of the forward/backward reachability, and the easiness to express the corresponding state sets. In our interpolant-based framework we implemented both the approaches. They share common features, and correspond to the main traversal scheme represented in Figure 1, with the addition of dynamically computed abstractions.

Forward verification is shown in Figure 5. Starting from T, it generates the α set and it obtains a first abstraction Γ from T. Then, within the loop, the set β is used at each iteration to generate a second abstraction $\widehat{\Gamma}$ starting from Γ. $\widehat{\Gamma}$ is employed within Img$^+_{Adq}$ images to achieve the desired overapproximation.

Backward reachability is similar, as the roles of the initial (I) and target (F) states are swapped, and the pre-image operator uses the transition relation in the reverse direction. However, it is possible to exploit different optimizations (see Figure 6).

We cannot generate the k-adequate Γ by component removal, as true quantification (before or within pre-image computation) is required. Nevertheless, we can apply state variable quantification by composition within the pre-image computation. This can drastically reduce the amount of variables to be quantified in pre-image, and allows us to adopt exact pre-image (using) Γ (at least partially). The LazyPreImg procedure performs exact circuit quantification, for primary inputs and α variables, whenever convenient (otherwise the variable is

```
FINITERUN_Bwd (I, T, F, k)
        if (SAT(BMC$_0^k$))
                return (reachable)
        $\alpha$ = DYNABSTRBMC (I, T, F, k))
        $\Gamma$ = ABSTR($T, \alpha$)
        R = From = F
        while (true)
                $\beta$ = DYNABSTRADQ (I, $\Gamma$, From, k)
                if ($\beta$ = undecided)
                        return (undecided)
                $\widehat{\Gamma}$ = ABSTR($\Gamma, \beta$)
                $\widehat{To}$ = LAZYPREIMG ($\widehat{\Gamma}$, From)
                if (more variables to quantify)
                        To = ITP ($\widehat{\text{To}}$, I $\wedge$ $\widehat{\Gamma}_0^k$)
                else
                        To = $\widehat{To}$
                if (To $\Rightarrow$ R)
                        return (unreachable)
                From = REDREMOVAL (To, $\neg$R)
                R = R $\vee$ To
```

Fig. 6. Backward traversal.

kept) with a method inspired by Abdulla et al. [2000] and Cabodi et al. [2005]. If we are able to fully complete exact pre-image with LAZYPREIMG, then we avoid SAT-based overapproximation by interpolant, thus providing a tighter pre-image. Otherwise we interpolate.

The peculiar aspect of the above procedure is to achieve quite often a full interpolation by dynamic abstraction and circuit quantification, without resorting to SAT-based Craig interpolants The procedure showed very good results in some experimental cases, where it was able to outperform forward interpolation, even though it is less general.

6. EXPERIMENTAL RESULTS

We compare interpolant-based Model Checking, with and without the optimizations described in this paper. Our procedures are implemented on top of the Minisat [Eén and Sörensson 2003a] SAT tool. We also use our own implementation of an AIG library, and we exploit the ABC tool [Mishchenko 2005] for different kinds of logic synthesis optimizations.

Our experiments ran on a Pentium Dual Core 3GHz Workstation with 3GByte of main memory, running Debian Linux. We performed extensive experiments on selected benchmarks, with both forward and backward interpolation, by specifically addressing proofs of correctness.

We present results on:

—Some standard benchmarks belonging to the VIS distribution.

—The Sun PicoJava II microprocessor as presented in McMillan [2003]. It includes 20 true safety properties with a number of state variables (after cone of influence reduction) ranging from about 50 to 350.

Table I. Forward CPU Time (in seconds) for the Different Strategies

Model	# FF	Fwd	+DA	+RR	+AR	+DA+RR+AR
$Ns2_1$	67	648	**201**	406	283	216
$Ns2_2$	67	791	**72**	481	398	101
$IndustrialA_1$	78	–	190	390	281	**105**
$IndustrialA_2$	79	378	**143**	308	228	199
$PicoJava_5$	88	49	34	**23**	57	70
Ns3	103	–	–	1606	629	**285**
Blackjack	103	534	1236	840	**305**	352
31_2_batch_1	122	1470	124	–	795	**63**
$Soap_1$	140	–	–	1163	906	**728**
$Soap_2$	140	–	–	–	**604**	860
$IndustrialB_1$	186	164	39	101	64	**32**
$IndustrialB_2$	202	556	–	–	**116**	142
$PicoJava_{16}$	290	**61**	228	201	100	116
Feistel	296	519	**133**	465	509	215
$PicoJava_6$	322	**130**	212	284	233	298
$PicoJava_{15}$	364	45	52	46	37	**22**
$IndustrialC_1$	377	782	**62**	224	175	144
$IndustrialC_2$	673	–	1453	–	**368**	521

"–" means time out after 1800 seconds.

Table II. Backward CPU Time (in seconds) for the Different Strategies

Model	# FF	Bwd	+DA	+RR	+AR	+DA+RR+AR
$Ns2_1$	67	–	453	**201**	–	464
$Ns2_2$	67	–	–	**397**	–	636
$IndustrialA_1$	78	1062	504	742	341	**125**
$IndustrialA_2$	79	241	144	185	**90**	127
$PicoJava_5$	88	53	24	32	180	**10**
Ns3	103	–	–	1404	–	**1141**
Blackjack	103	–	–	1168	–	**589**
31_2_batch_1	122	–	–	–	–	–
$Soap_1$	140	–	**304**	801	–	938
$Soap_2$	140	848	**144**	539	1075	298
$IndustrialB_1$	186	1534	653	750	721	**402**
$IndustrialB_2$	202	72	38	89	77	**28**
$PicoJava_{16}$	290	–	–	–	–	–
Feistel	296	–	1563	1228	–	**907**
$PicoJava_6$	322	–	–	1561	1062	**546**
$PicoJava_{15}$	364	–	–	–	–	–
$IndustrialC_1$	377	–	–	–	–	–
$IndustrialC_2$	673	231	51	35	43	**30**

"–" means time out after 1800 seconds.

—The IBM Formal Verification Benchmark Library [IBM 2003]. This library includes 75 circuits, each one with one property, with a size ranging from 95 to 917 memory elements.

—Some industrial circuits coming from STMicroelectronics.

Tables I and II include detailed results for the forward and backward approach respectively. They compare the running time of the different strategies we implemented, that is, the original forward and backward versions against the ones obtained with our optimizations.

We consider only runs that needed more than 20 second of CPU time (in both the forward and backward directions), and that we were able to complete, in less than 1800 seconds, with at least one strategy. Each line reports data on one single property. More lines labeled with the same circuit name indicate that more properties are verified on the same circuit. Bold characters show the best result for each property.

The two tables report the number of memory elements for each circuit (after COI extraction), and the verification time for the different interpolation strategies. The bottom-line strategy is represented by Fwd (Bwd), where we indicate the time for the original interpolation method [McMillan 2003] reimplemented in our tool in the forward and backward directions. +DA presents results by adopting dynamic abstraction at the image level (see Figure 3). +RR indicates the presence of redundancy removal. +AR includes abstraction refinements at the traversal level (see Figure 2). In all the strategies we adopted synthesis optimization techniques as delivered by the ABC tool [Mishchenko 2005].

Overall, we can do the following observations. Forward reachability is generally better than backward, though some cases exist where the backward verification is much faster than the forward one, for example, circuit IndustrialC_2. Redundancy removal has a larger impact on backward reachability. In the forward direction it represents, at least in some cases, just an overhead. The three contributions have different impact on different benchmarks. Even if combining all of them does not correspond to the best possible performance, it often produces a more robust verification framework. To this respect, we were able to complete 5 (7) problems in the forward (backward) direction with the Fwd+DA+RR+AR methodology, that we were not able to achieve with the original approach. On the other hand, circuits PicoJava_{16} and PicoJava_6 are the only two benchmarks for which the original (forward) strategy gives the best result.

Figures 7 and 8 show further information exactly on the same set of benchmarks presented in Tables I and II. More in details, Figure 7 reports some data concerning the number of reachability iterations (within the FiniteRun procedure) for the Fwd+DA+RR+AR method against the Fwd strategy. For the experiments that run out of time in Table I, we provide the results we obtained just before the time limit.

The figure plots the number of FiniteRun calls (external interpolation loop) and the number of FiniteRun inner iterations (inner interpolation loop). For the last one, we report both the maximum value (maximum number of inner iterations on all main iterations) and the total value (total number of inner iterations performed on all main iterations). Figure 7 clearly shows that most of the points are below the $x = y$ line; that is, the number of outer and inner iterations of FiniteRun is drastically reduced.

Figure 8 reports the maximum size of the support of the To set, for the same two strategies compared in Figure 7.

A direct comparison between the original and the proposed method shows that the latter one is able to reduce the support of the computed state sets, backing up the original claims of the paper.

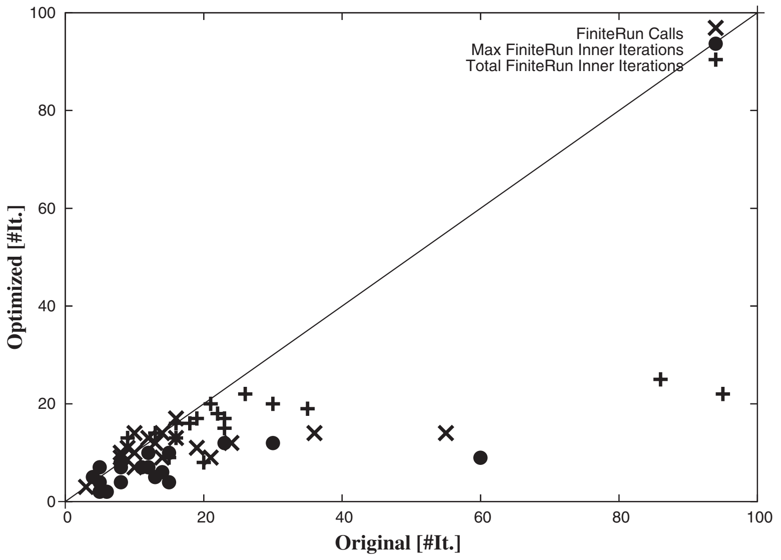

Fig. 7. Number of FiniteRun calls (external interpolation loop) and number of FiniteRun inner iterations (maximum and total values): Fwd + DA + RR + AR method against Fwd.

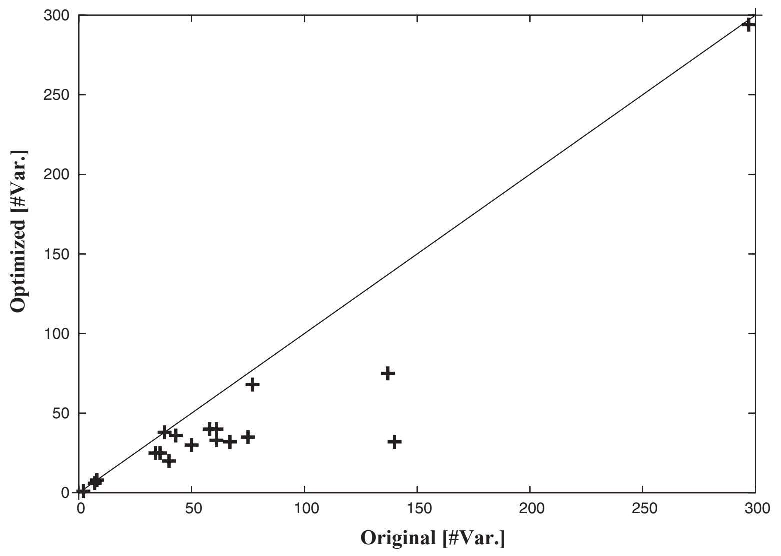

Fig. 8. Maximum support of the to set along the whole traversal process: Fwd + DA + RR + AR method against Fwd.

7. CONCLUSIONS

This paper shows how Craig interpolants, derived from SAT proofs, can be further optimized by:

—Extending the idea of interpolation to variable abstraction.

—Effectively exploiting redundancy removal under External and Observability Don't Care conditions.
—Circuit-based quantification adopted in the backward direction.

Our optimizations heavily rely on an efficient use of incremental SAT, which permits grouping several related problems with common learning.

To conclude, an interpolation approach still shows its main limits with sequentially deep verification instances correlated with large interpolant circuits. Solving the preceding problems seem to be the main challenge for future works in interpolant-based verification.

Another interesting options is to better integrate and combine interpolant verification with other state-of-the-art methods such as abstraction-refinements, inductive verification and automated generation of lemmas.

ACKNOWLEDGMENTS

The authors would like to thank K. L. McMillan for the source descriptions of the Sun PicoJava II microprocessor, and the VTT group of the STMicroelectronics, Agrate, Italy, for their industrial benchmarks.

REFERENCES

Abdulla, P. A., Bjesse, P., and Een, N. 2000. Symbolic reachability analysis based on SAT-solvers. In *Tools and Algorithms for the Construction and Analysis of Systems*, M. I. S. Susanne Graf, Ed. Vol. 1785. Springer-Verlag, Berlin, Germany, 411–425.

Berkelaar, M. and van Eijk, K. 2002. Efficient and effective redundancy removal for million-gate circuits. In *Proceedings of the Design Automation & Test in Europe Conference.* Paris, France, IEEE Computer Society, 1088–1088.

Biere, A., Cimatti, A., Clarke, E. M., Fujita, M., and Zhu, Y. 1999. Symbolic model checking using SAT procedures instead of BDDs. In *Proceedings of the 36th Design Automation Conference.* New Orleans, LO, IEEE Computer Society, 317–320.

Bjesse, P., Leonard, T., and Mokkedem, A. 2001. Finding bugs in an alpha microprocessor using satisfiability solvers. In *Proceedings of the International Conference on Computer Aided Verification*. Paris, France, G. Berry, H. Comon, and A. Finkel, Eds. Lecture Notes in Computer Science, vol. 2102. Springer-Verlag, 454–464.

Bryant, R. E. 1986. Graph–based algorithms for Boolean function manipulation. *IEEE Trans. Comput. C–35*, 8, 677–691.

Burch, J. R., Clarke, E. M., Long, D. E., McMillan, K. L., and Dill, D. L. 1994. Symbolic model checking for sequential circuit verification. *IEEE Trans. Comput.-Aid. Des. 13*, 4, 401–424.

Cabodi, G., Nocco, S., and Quer, S. 2005. Circuit based quantification: back to state set manipulation within unbounded model checking. In *Proceedings of the Design Automation and Test in Europe Conference.* Munich, Germany, IEEE Computer Society.

Chauhan, P., Clarke, E., Kukula, J., Sapra, S., Veith, H., and Wang, D. 2002. Automated abstraction refinement for model checking large state spaces using SAT based conflict analysis. In *Proceedings of the Formal Methods in Computer-Aided Design*. Portland, OR, M. D. Aagaard and J. W. O'Leary, Eds. Lecture Notes in Computer Science, vol. 2517. Springer, 35–51.

Cho, H., Hachtel, G., Jeong, S. W., Plessier, B., Schwarz, E., and Somenzi, F. 1990. ATPG aspects of FSM verification. In *Proceedings of the International Conference on Computer-Aided Design*. San Jose, CA, 134–137.

Coudert, O., Berthet, C., and Madre, J. C. 1989. Verification of sequential machines using Boolean function vectors. In *Proceedings of the International Federation for Information Processing Workshop on Applied Formal Methods for Correct VLSI Design*. Vol. 1. 111–128.

Coudert, O. and Madre, J. C. 1991. Symbolic computation of the valid states of the sequential machine: Algorithms and discussion. In *International Workshop on Formal Methods in VLSI Design*.

Eén, N. and Sörensson, N. 2003a. Minisat SAT Solver. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/.

Eén, N. and Sörensson, N. 2003b. Temporal induction by incremental sat solving. In *Proceedings of the 1st International Workshop on Bounded Model Checking (BMC)*. Boulder, CO.

Ganai, M. K., Gupta, A., and Ashar, P. 2004. Efficient SAT-based unbounded symbolic model checking using circuit cofactoring. In *Proceedings of the International Conference on Computer-Aided Design*. San Jose, CA. IEEE Computer Society.

IBM. 2003. Ibm formal verification benchmark library. http://www.haifa.il.ibm.com/projects/-verification/rb_homepage/benchmarks.html.

Kang, H. J. and Park, L. C. 2003. SAT-based unbounded symbolic model checking. In *Proceedings of the 40th Design Automation Conference.* Anaheim, CA. IEEE Computer Society, 840–843.

Kurshan, R. P. 1994. Computer aided verification of coordinating processes. Princeton University Press, Princeton, NJ.

Lin, B., Wang, C., and Somenzi, F. 2003. A satisfiability-based approach to abstraction refinement in model checking. In *1st International Workshop on Bounded Model Checking (BMC)*. Boulder, CO.

Manquinho, V. and Marques-Silva, J. 2002. Search pruning techniques in SAT–based branch-and-bound algorithms for the Binate covering problem. *IEEE Trans. Comput.-Aid. Des. 21*, 505–516.

Marques-Silva, J. 2005. Improvements to the implementation of interpolant–based model checking. In *Proceedings of the Computer Aided Verification*. Edinburgh, Scotland, UK. D. Borrione and W. Paul, Eds. Lecture Notes in Computer Science, vol. 3725. Springer, 367–370.

McMillan, K. L. 2002. Applying SAT methods in unbounded symbolic model checking. In *Proceedings of the Computer Aided Verification*. Copenhagen, Denmark, E. Brinksma and K. G. Larsen, Eds. Lecture Notes in Computer Science, vol. 2404. Springer, 250–264.

McMillan, K. L. 2003. Interpolation and SAT-based model checking. In *Proceedings of the Computer Aided Verification*. Boulder, CO, W. A. H. Jr. and F. Somenzi, Eds. Lecture Notes in Computer Science, vol. 2725. Springer, 1–13.

McMillan, K. L. and Jhala, R. 2005. Interpolation and SAT-based model checking. In *Proceedings of the Computer Aided Verification*. Edinburgh, Scotland, UK. T. Ball and R. B. Jones, Eds. Lecture Notes in Computer Science, vol. 3725. Springer, 39–51.

Mishchenko, A. 2005. ABC: A system for sequential synthesis and verification, http://www.eecs.berkeley.edu/~alanmi/abc/.

Mishchenko, A. and Brayton, R. K. 2006a. Improvements to combinational equivalence checking. In *Proceedings of the International Conference on Computer-Aided Design*. San Jose, CA. ACM Press.

Mishchenko, A. and Brayton, R. K. 2006b. Scalable logic synthesis using a simple circuit structure. In *Proceedings of the International Workshop on Logic Synthesis*. Lake Tahoe, CA.

Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference*. Las Vegas, NV. IEEE Computer Society.

Sheeran, M., Singh, S., and Stålmarck, G. 2000. Checking safety properties using induction and SAT solver. In *Proceedings of the Formal Methods in Computer-Aided Design*. Austin, TX, W. A. Hunt and S. D. Johnson, Eds. Lecture Notes in Computer Science, vol. 1954. Springer-Verlag, 108–125.

Whittemore, J., Kim, J., and Sakallah, K. 2001. Satire: A new incremental satisfiability engine. In *Proceedings of the 38th Design Automation Conference*. ACM Press, 542–545.

Williams, P. F., Biere, A., Clarke, E. M., and Gupta, A. 2000. Combining decision diagrams and SAT procedures for efficient symbolic model checking. In *Proceedings of the Computer Aided Verification*. Chicago, IL, E. A. Emerson and A. P. Sistla, Eds. Lecture Notes in Computer Science, vol. 2102. Springer-Verlag. 124–138.

Received June 2006; revised June 2007; accepted July 2007