A Survey of Recent Advances in SAT-Based Formal Verification

Mukul R Prasad[1], **Armin Biere**[2], **Aarti Gupta**[3]

[1] Fujitsu Labs. of America, Sunnyvale, CA, USA
[2] Johannes Kepler University, Linz, Austria
[3] NEC Labs. America, Princeton, NJ, USA

The date of receipt and acceptance will be inserted by the editor

Abstract Dramatic improvements in SAT solver technology over the last decade, and the growing need for more efficient and scalable verification solutions have fueled research in verification methods based on SAT solvers. This paper presents a survey of the latest developments in SAT-based formal verification, including incomplete methods such as bounded model checking, and complete methods for model checking. We focus on how the surveyed techniques formulate the verification problem as a SAT problem, and how they exploit crucial aspects of a SAT solver, such as application-specific heuristics and conflict-driven learning. Finally, we summarize the noteworthy achievements in this area so far, and note the major challenges in making this technology more pervasive in industrial design verification flows.

Key words: Verification, SAT, Model Checking, QBF, ATPG

1 Introduction

Functional verification of digital hardware designs has become one of the most expensive and time-consuming components of the current product development cycle. Symbolic model checking based on BDDs [22, 72] have come a long way since their introduction more than a decade ago. However, they are still incapable of handling the largest problems encountered in current industrial practice. Reduction in feature size coupled with the recent move towards IP-based design has led to dramatic increases in the size and complexity of systems that are being designed, thereby posing new challenges for functional verification methods. Hence there is a growing need to investigate and develop more robust and scalable verification methods based on novel and alternative technologies.

Verification methods based on SAT solvers have recently emerged as a promising solution. Dramatic improvements in

SAT solver technology over the past decade have led to the development of several powerful SAT solvers [45, 71, 77, 105]. Verification methods based on these solvers have been shown to push the envelope of functional verification in terms of both capacity and efficiency, as reported in several academic and industrial case studies [4, 16, 19, 31]. This has fueled further interest and intense research activity in the area of SAT-based verification.

This paper surveys the recent developments in SAT-based formal verification techniques and methodologies. The work surveyed falls primarily in the category of property verification or model checking methods since such has been the focus of most recent works on SAT-based verification. For other verification applications of SAT methods, such as combinational equivalence checking, the interested reader is referred to [47, 68].

Additionally, there is an interesting body of work based on applying SAT to richer types of specifications and logics, which, due to lack of space can not be covered in this short survey. Here is a list of recent relevant topics, which may serve as a starting point for the interested reader: quantifier free fragments of first order logic [9, 87, 100], Presburger Arithmetic [97], monadic second order logic [6], object oriented software specifications [60].

1.1 Organization

The survey is organized as follows. Section 2 briefly reviews the SAT problem, basic SAT algorithms and advanced features of modern SAT solvers, and model checking. Section 3 discusses work on *bounded model checking (BMC)* including ways of strengthening SAT-based BMC with BDD-based analysis and several industrial case studies comparing SAT-BMC with traditional BDD-based symbolic model checking. Section 4 reviews techniques that implement complete methods for model checking based on state-space search, inductive reasoning and abstraction-refinement.

Recently there have been some successful attempts at using sequential ATPG tools for model checking. These are surveyed in Section 5. Another recent development has been the use of *Quantified Boolean Formulae (QBF)* solvers, a generalization of SAT, to solve model checking problems. The state-of-the-art in QBF solving and its applications to verification are discussed in Section 6. We conclude the paper in Section 7 with a summary of the major achievements in SAT-based verification to date and some thoughts on the future prospects and challenges for SAT-based verification.

2 Background

2.1 The Boolean Satisfiability Problem

The Boolean Satisfiability (SAT) problem is a well-known constraint satisfaction problem, with many applications in the fields of VLSI Computer-Aided Design and Artificial Intelligence. Given a propositional formula φ, the Boolean Satisfiability problem posed on φ is to determine whether there exists a variable assignment under which φ evaluates to true. Such an assignment, if it exists, is called a *satisfying assignment* for φ and φ is called *satisfiable*. Otherwise φ is said to be *unsatisfiable*. The SAT problem is known to be NP-Complete [42]. However, in practice, there has been tremendous progress in SAT solvers technology over the years, summarized in a recent survey [107]. Earlier work in the context of theorem proving is covered in [63].

Most SAT solvers use a *Conjunctive Normal Form (CNF)* representation of the Boolean formula. In CNF, the formula is represented as a conjunction of clauses, each clause is a disjunction of literals, and a literal is a variable or its negation. Note that in order for the formula to be satisfied, each clause must also be satisfied, *i.e.*, evaluate to true. There exist polynomial algorithms (*e.g.*, [81,98]) to transform an arbitrary propositional formula into a *satisfiability equivalent* CNF formula that is satisfiable if and only if the original formula is satisfiable. Similarly a Boolean circuit may be encoded as a satisfiability equivalent CNF formula using the method of [65]. Alternatively, for SAT applications arising from the circuit domain, the SAT solver may be modified to work directly on the Boolean circuit representation.

2.2 SAT Solvers

Most modern SAT solvers are based on the *Davis-Putnam-Logemann-Loveland (DPLL) algorithm* [32,33], which performs a branching search with backtracking. The DPLL algorithm is sound and complete, *i.e.*, it finds a solution if and only if the formula is satisfiable. In this section, we summarize the main features of modern DPLL-based SAT solvers. This provides the context for enhancements targeted at verification applications, discussed in the rest of the paper.

Probabilistic SAT solvers, including WALKSAT [85] and GSAT [86], are based on stochastic local search instead of

```
sat-solve()
    if preprocess() = CONFLICT then
        return UNSAT;
    while TRUE do
        if not decide-next-branch() then
            return SAT;
        while deduce() = CONFLICT do
            blevel ⟸ analyze-conflict();
            if blevel = 0 then
                return UNSAT;
            backtrack (blevel);
        done;
    done;
```

Figure 1. DPLL-based SAT Solver

DPLL. They have their strength on random SAT instances, but in practice do not work well on structured instances obtained from real verification problems.

The basic skeleton of DPLL-based SAT solvers is shown in Figure 1, adapted from the GRASP work [71]. The initial step consists of some preprocessing, during which it may be discovered that the formula is unsatisfiable. The outer loop starts by choosing an unassigned variable, and a value to assign to it (decide-next-branch). If no such variable exists, a solution has been found. Otherwise, the variable assignments deducible from this decision are made (using deduce), through a procedure called *Boolean Constraint Propagation (BCP)*. It typically consists of iterative application of the *unit clause rule*, which is invoked whenever a clause becomes a unit clause, *i.e.*, all but one of its literals are false and the remaining literal is unassigned. According to the rule, the last unassigned literal is *implied* to be true – this avoids the search path where the last literal is also false, since such a path cannot lead to a solution. A *conflict* occurs when a variable is implied to be true as well as false. If no conflict is discovered during BCP, then the outer loop is repeated, by choosing the next variable for making a decision. However, if a conflict does occur, *backtracking* is performed within an inner loop in order to undo some decisions and their implications. If all decisions need to be undone (*i.e.*, the backtracking level *blevel* is 0), the formula is declared unsatisfiable since the entire search space has been exhausted.

The original DPLL algorithm used chronological backtracking, *i.e.*, it would backtrack up to the most recent decision, for which the other value of the variable had not been tried. However, modern SAT solvers use *conflict analysis* techniques (shown as (analyze-conflict) in the figure), to analyze the reasons for a conflict. Conflict analysis is used to perform *conflict-driven learning* and *conflict-driven backtracking*, which were incorporated independently in GRASP [71] and rel-sat [11]. Conflict-driven learning consists of adding *conflict clauses* to the formula, in order to avoid the same conflict in the future. Conflict-driven backtracking allows non-
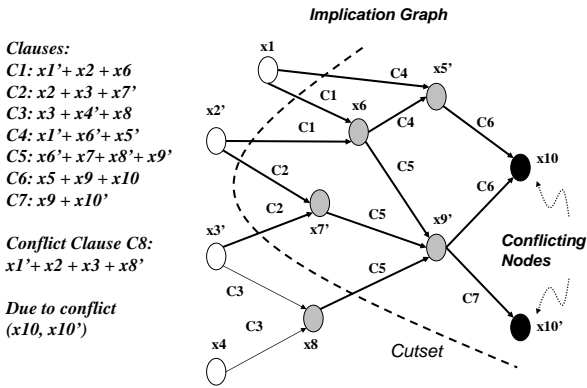
Figure 2. Conflict Analysis using an Implication Graph

chronological backtracking, *i.e.*, up to the closest decision which caused the conflict. These techniques greatly improve the performance of the SAT solver on structured problems.

The essential component of conflict analysis is an *implication graph* [71,106], which captures the current state of the SAT solver. A small example of an implication graph is shown in Figure 2, where the original SAT problem consists of clauses $C1 - C7$, as shown on the left. In an implication graph, nodes represent assignments to variables. For example, node $x1$ represents $x1 = 1$, and node $x5'$ represents $x5 = 0$.

Edges in an implication graph represent clauses, which cause implications due to source nodes on sink nodes. For example, when $x1 = 1$ and $x2 = 0$, clause $C1$ causes an implication $x6 = 1$. This is shown as two edges – between $x1$ and $x6$, and between $x2'$ and $x6$ – both marked with clause $C1$ as shown. Nodes with no incoming edges, such as $x1$, denote decision assignments (shown as white nodes in the figure). A *conflict* is indicated when there are two nodes in the graph with opposite values assigned to the same variable. In this example, a conflict is indicated by nodes $x10$ and $x10'$, which are called conflicting nodes. Conflict analysis takes place by following back the edges from the conflicting nodes, up to any edge cutset which separates the conflicting nodes from the decision nodes. An example cutset is shown by the dashed line in the figure. A conflict clause is derived from the variables feeding into the chosen cutset to capture the reasons for the conflict. It also corresponds to a resolution on all the clauses associated with the edges traversed up to the cutset. In this example, conflict clause $C8$ is derived as shown, corresponding to the observation that a partial assignment ($x1 = 1, x2 = 0, x3 = 0, x8 = 1$) always leads to a conflict. For conflict-driven learning, the derived clause $C8$ is added to the clause database in order to avoid the same conflict in the future.

Many other advances have been made in these basic components which comprise the DPLL-based SAT solver – deci-

sion engine (heuristics for choosing decision variables and values), deduction engine (data structures and heuristics for performing BCP and detecting conflicts), diagnosis engine (heuristics for conflict-driven learning). Some of these are described in the remainder of this section.

2.2.1 CNF-based SAT Solvers

An interesting property of CNF representations was first exploited by Zhang in the SATO SAT solver [105], to improve the performance of BCP. It proposed the use of head and tail pointers to point to non-false literals in the list representation of a clause, and maintained the *strong invariant* that all literals before the head pointer, and all literals after the tail pointer, are false. Clearly, detection of a unit clause during BCP becomes easy, *i.e.*, when the head and tail pointers coincide on an unassigned literal. The main advantage is that the clause status is updated *only* when either of the head/tail literals is assigned a false value during BCP. In particular, this eliminates an update when any of the other literals in the clause is assigned a value. When the head/tail literal is assigned a false value during BCP, the associated pointer needs to be moved to another non-false literal if it exists. This is facilitated by the strong invariant. However, during backtracking, the head/tail pointers may need to be moved back again, in order to maintain the strong invariant.

A different tradeoff was proposed by Moskewicz *et al.* in the Chaff SAT solver [77]. Its BCP scheme, known as *two literal watching with lazy update*, is also based on tracking only two literals per clause during BCP. However, Chaff maintains a *weak invariant*, whereby the two watched literals are required to be non-false, but there is no ordering requirement with respect to other false literals. Again, detection of a unit clause during BCP is easily performed by checking whether both watched pointers coincide, and whether clause updates on assignment to other literals are eliminated. Note that due to the weaker invariant, more work than SATO may be required during BCP, to search for a non-false literal when one of the two watched literals is assigned a false value. However, the weaker invariant ensures that no additional work is required during backtracking. This tradeoff has been shown to work better in practice.

Chaff also proposed a useful decision heuristic that prioritizes the literals that appear in recent conflict clauses. Recall that conflict clauses are added due to conflict-driven learning, which is very beneficial for SAT solvers on structured problems. This was taken a step further by Goldberg and Novikov in the BerkMin SAT solver [45], which prioritizes all literals involved in the conflict analysis, and not just those that appear in the conflict clause. The performance improvement due to these decision heuristics is additional testament to the importance of conflict-driven learning in practice.

More recently, additional information recorded during conflict analysis has been used very effectively to provide a proof when a formula is determined to be unsatisfiable by the SAT solver. This proof can be independently checked to verify the SAT solver itself [46,109]. These techniques can also be

easily adapted to identify a subset of clauses from the original problem, called the *unsatisfiable core* [109, 75], such that these clauses are sufficient for implying unsatisfiability. The use of such techniques in verification applications are described in more detail in Section 4.

2.2.2 Circuit-based SAT Solvers

SAT has many applications in the logic circuit domain, such as automatic test pattern generation (ATPG), verification, timing analysis, *etc.* The Boolean reasoning problem is typically derived from the circuit structure. This has also led to interest in circuit-based SAT solvers [38, 44, 64]. These work directly on the circuit structure, and use circuit specific heuristics to guide the search. In general, attempts to include circuit structure information into CNF-based SAT solvers have been unsuccessful due to significant overhead.

Among verification applications, recently Kühlmann *et al.* [64] focused on SAT techniques for a simple, uniform gate-level representation of circuits, and their integration with other useful techniques like BDD sweeping and dynamic circuit transformation. Circuit-based BCP is performed by a single table lookup per gate, in contrast to CNF-based updates for potentially three equivalent clauses. This improves the BCP performance. However, there is no effective way to perform conflict-driven learning. The bottleneck is that conflict clauses correspond to large *OR-tree* circuits.

2.2.3 Hybrid SAT Solvers

More recently, there has been an effort by Ganai *et al.* [41] to combine the relative benefits of CNF-based and circuit-based SAT solvers. In particular, their hybrid SAT solver incorporates efficient circuit-based BCP techniques, along with conflict analysis techniques of CNF-based solvers. The original circuit problem is represented as a simple gate-level netlist, while the learned conflict clauses are represented in CNF. The BCP engine consists of table lookups for the gates, and a Chaff-style two-literal watching scheme for conflict clauses. Note that since the clauses for a simple gate are short clauses (3-literals or less), a single table lookup is cheaper than multiple clause updates. On the other hand, since conflict clauses tend to be much longer, a two-literal watching scheme (which avoids multiple updates) is more useful than multiple table lookups for its many literals. This enables consistent speedups in the BCP performance. Their hybrid representation of the Boolean problem allows exploitation of both circuit-based and CNF-based decision heuristics.

Another effort by Lu *et al.* [69] used these ideas along with additional conflict-driven learning, in order to improve the SAT solver performance. The idea is to use cheaper methods (such as simulation) to find candidate pairs of co-related signals in the given circuit. Then the inequivalence of the co-related signals is added as a constraint to the SAT problem. Since this constraint is likely to be conflicting, it provides additional opportunities for the SAT solver to perform conflict-driven learning, which can potentially improve its performance on larger problems.

2.3 Model Checking

With the introduction of *bounded model checking* [15] it became clear that SAT can be used for *model checking* [27]. One can even argue, that currently one of the main driving forces behind SAT research is its application to model checking. The purpose of this section is to give a short overview on the history and terminology of model checking. More details can be found in the text book [28] or the survey [30].

The target of model checking is the verification of sequential properties of dynamic systems. A dynamic system has a state component which changes over time. Typical systems are sequential circuits, which contain delay elements, such as flip-flops and latches. Verification of sequential circuits is also the main application area of this survey.

Model checking, in the first place, is only applicable to finite systems. However, if suitable *finite abstractions* can be found, then some classes of infinite systems can be checked as well. Applications of infinite model checking are real-time systems, modeled as timed automata [3], or even system software [8]. Further, model checking has also been applied successfully in the context of telecommunication protocols and cache coherence protocols. It must be noted, that SAT has mainly been used for model checking sequential circuits. However, it is apparent that the verification of more general systems can also benefit from SAT technology.

Sequential properties are usually represented in temporal logic [36]. Formulas of temporal logic try to express system behavior over time. There are various variants of temporal logic, such as *Linear Temporal Logic (LTL)* or *Computation Tree Logic (CTL)*, which usually require dedicated algorithms. In this paper we focus on *simple safety properties*, also often called invariants, written in CTL as **AG**p. This formula specifies, that for all executions paths, globally in all states along the path, the property p holds. Alternatively, it states the property that $\neg p$, read as *not p*, which could be some catastrophic system state, can not be reached. Note that for finite systems, many practically relevant properties can be translated into simple safety properties [84].

Originally [27] model checking used an explicit representation of states. A typical implementation [55] of this type of *explicit model checking* stores individual states in a large hash table, memorizing the states reached during a depth first traversal of the state space. Since the number of states of even small systems can be very large, *e.g.*, a 128 bit shift register has 2^{128} states, this method does not scale, in particular for sequential circuits. One solution to this so called *state explosion problem* is *symbolic model checking* [72], which operates on sets of states instead of individual states and represents sets of states symbolically in a compact form.

In the past, Binary Decision Diagrams (BDDs) [21] and variants have frequently been used as representation for sets of states. They also allow efficient computation of Boolean

$\underline{\text{model-check}^{\mu}_{\text{forward}}} (I, T, B)$
   $S_C = \emptyset; S_N = I;$
   while $S_C \neq S_N$ **do**
     $S_C = S_N;$
     if $B \cap S_C \neq \emptyset$ **then**
       return "found error trace to bad states";
     $S_N = S_C \cup Img(S_C);$
   done;
   return "no bad state reachable";

Figure 3. Forward least fixpoint algorithm for safety properties

$\underline{\text{model-check}^{\nu}_{\text{backward}}} (I, T, G)$
   $S_C = $ "all states"; $S_P = G;$
   while $S_C \neq S_P$ **do**
     $S_C = S_P;$
     $S_P = S_C \cap \mathbf{AX}(S_C);$
   done;
   if $I \Rightarrow S_C$ **then return** "only good states reachable";
   else return "found error trace to bad states";

Figure 4. Backward greatest fixpoint algorithm for safety properties

operations. In particular BDDs allow an efficient implementation of the image operation *Img*, which lies at the core of the breadth first search in symbolic model checking. It calculates the states reachable in one step via the transition relation T from the current set of states S_C, by implicitly conjoining the BDD representing S_C with the BDD representing T and projecting the result onto the next state variables Y (after eliminating the current state variables X and primary input variables W).

$$Img(Y) \equiv \exists X, W. \; S_C(X) \wedge T(X, Y, W) \qquad (1)$$

In the context of circuits, we additionally assume that the transition relation is deterministic. As shown above, it may however depend on primary inputs, encoded by a vector W of Boolean variables, which also need to be quantified during image computation. In the terminology of program verification, *Img* calculates the strongest post condition of a given predicate.

A basic algorithm for symbolic model checking simple safety properties can then be formulated as in Figure 3. It represents sets of states symbolically, and searches in breadth first order from the initial states to the bad states. Let B be the set of bad states, in which p does not hold, and I the set of initial states.

This *forward model checking* algorithm, starts at the initial states and searches forward along the transition relation. In the literature one can also find *backward model checking* algorithms. They rely on a dual operation to the *Img* operation *PreImg*, or equivalently the CTL operator **EX**. It calculates the set of previous states S_P that may reach the given set of current states S_C in one step:

$$PreImg(X) \equiv \exists Y, W. \; S_C(Y) \wedge T(X, Y, W)$$

A *backward model checking* algorithm can be obtained from the forward algorithm by, in essence, exchanging B with I and *Img* with *PreImg*. In practice, forward traversal usually is much faster [58, 57, 54, 17]. The reason may be, that unreachable states do not have to be visited and BDDs behave much better. However, not all temporal properties, for instance **EX**$p \wedge$ **EX**q or **AG EX**p, can be handled with *Img* computation only. In certain cases backward traversal is better. For instance, if the property p is an inductive invariant,

as defined in Section 4.2. In this case the backward fixpoint computation terminates after one *PreImg* computation. A general strategy is to try backward and forward traversal in parallel.

Both symbolic model checking algorithms presented so far can be interpreted as calculating a least fixpoint [22]. Dual formulations exist for greatest fix points. For backward traversal, the CTL operator **AX** (also known as the weakest pre condition operator *wp*) replaces *PreImg*:

$$\mathbf{AX}(X) \equiv \forall Y, W. \; T(X, Y, W) \rightarrow S_C(Y)$$

It calculates the set of previous states S_P that lead to a state in the current set of states S_C, independent of the values at the primary inputs. A backward model checking algorithm for simple safety properties, based on greatest fix point calculation and on the **AX** operator, can be formulated as in Figure 4. Here, G denotes the set of *good states*, *i.e.*, the states in which p holds.

SAT technology can be used for implementing all parts of these algorithms. One option is to unroll the loop in model-check$^{\mu}_{\text{forward}}$ only a finite number of times, omitting the termination checks. This, in essence, is the main idea behind bounded model checking, the topic of the next section. We will come back to backward traversal calculating greatest fix points in Section 4.1.2.

3 Bounded Model Checking

Bounded Model Checking based on SAT methods was introduced by Biere *et al.* in [14, 15, 26] and is rapidly gaining popularity as a complementary technique to BDD-based symbolic model checking. Given a temporal logic property $\mathcal{P}$ to be verified on a finite transition system M, the essential idea is to search for counter-examples to $\mathcal{P}$ in the space of all executions of M whose length is bounded by some integer k.

The problem is formulated by constructing the following propositional formula:

$$\varphi^k = I \wedge \bigwedge_{i=0}^{k-1} T_i \wedge (\neg \mathcal{P}^k) \qquad (2)$$

where I is the characteristic function for the set of initial states of M, T_i is the characteristic function of the transition
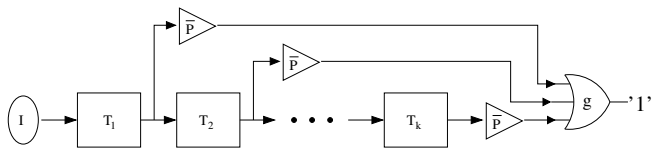
Figure 5. Bounded Model Checking

relation of M for time step i. Thus, the formula $I \land \bigwedge_{i=0}^{k-1} T_i$ precisely represents the set of all executions of M of length k or less, starting with a legal initial state. $\neg\mathcal{P}^k$ is a formula representing the condition that $\mathcal{P}$ is violated by a bounded execution of M of length k or less. Hence, φ^k is satisfiable if and only if there exists an execution of M of length k or less that violates property $\mathcal{P}$. φ^k is typically translated to CNF and solved by a conventional SAT solver.

The formula $\neg\mathcal{P}^k$ may be used to express both safety and liveness properties. Liveness properties of the form $\mathbf{AF}p$ are checked by having $\neg\mathcal{P}^k$ represent a loop within a bounded execution of length at most k, such that p is violated on each state in the loop. However, the more common application of BMC is for the purpose of checking safety properties of the form $\mathbf{AG}p$ (p is some propositional expression). In this case expression (2) reduces to:

$$\varphi^k = I \land \bigwedge_{i=0}^{k-1} T_i \land (\bigvee_{i=0}^{k} \neg P_i) \tag{3}$$

where P_i is the expression p in time step i. Thus, this formula can be satisfied if and only if for some i $(i \le k)$ there exists a reachable state in time step i in which p is violated. Figure 5 shows a circuit representation of this equation, where the block $\overline{P}$ denotes a combinational circuit block computing $\neg P_i$ as a function of the state variables of time step i.

A typical application of BMC consists of iteratively executing the above formulation for increasing values of k until either a property violation is discovered or some user specified limit on k or the computing resources (memory, runtime) is exceeded.

Recent research has improved upon both the technology and methodology of the basic BMC method described above in several ways. These improvements are discussed below.

3.1 Structural Pruning during CNF Generation

Many techniques use some kind of structural processing to generate a more compact CNF for the BMC problem, with the hope that the resulting SAT problem is easier for the SAT solver to solve.

The *bounded cone of influence (BCOI)* reduction [16] is an improvement on the classical *cone of influence (COI)* reduction used in traditional model checking. The intuition is that over a bounded time interval we need not consider every state variable in the classical COI at every time step. Specifically, in Figure 5, the BCOI reduction would extract the transitive fanin cone of the gate g and construct the BMC-CNF

only from this sub-circuit. In our experience the BCOI reduction is cheap and easy to apply and can occasionally provide significant improvements over the simple COI reduction.

Ganai *et al.* [39] use binary AND-INVERTER graphs [64] to represent the transition relation of the system as well as the unrolled transition relation used for the BMC problem (Figure 5). The graph is compressed, as it is built, by using an efficient functional hashing scheme across two levels of logic, as well as term re-writing techniques. The CNF for the BMC problem is generated from this compressed representation. SAT results from earlier BMC runs are used to set appropriate $\overline{P}$ nodes (Figure 5) to 0 and then re-hash the circuit graph to obtain further compression. Such techniques work extremely well in practice especially if the logic level circuit used for the verification has been generated through a quick on-the-fly synthesis from an RTL description.

3.2 Decision Variable Ordering of the SAT solver

Variable ordering has long been recognized as a key determinant of the performance of SAT solvers. The earliest works on SAT-BMC were based on SAT solvers such as GRASP and SATO which used variable ordering heusristics such as the *DLIS* heuristic [70]. Strichman [95] proposed a static variable ordering scheme specifically targeted for BMC problems which improved upon the default DLIS ordering. The static order was generated from a BFS-like traversal of the unrolled circuit graph used for BMC.

However, recent results [88] show that the conflict-driven variable ordering heuristics used in modern SAT solvers (*e.g.*, the VSIDS heuristic in zchaff [77]) outperform any *fully* static BMC-specific variable ordering scheme, such as the one proposed in [95]. A slight tuning of these heuristics for the BMC problem [88] can further enhance the performance. On the other hand, BMC tools using circuit-based SAT solvers, *e.g.*, [41,59,64] essentially use some variant of the *J-frontier justification* heuristic popularly used in sequential ATPG tools.

While the above heuristics work fairly well for a SAT solver in a BMC setting, they do not specifically exploit any key aspects of the BMC problem to customize and target the SAT search for BMC. Since the SAT solver's runtime dominates the overall performance of the BMC tool, this topic could be an interesting avenue for future research.

3.3 Addition of Constraints to the SAT problem

The technique of learning *conflict clauses* during search has dramatically enhanced the efficacy of modern SAT solvers. Motivated by this, several other specialized static and dynamic learning techniques have been developed for the BMC problem. The learned constraints can be added as CNF clauses to the SAT problem being solved, with the hope of speeding up the solution process.

The technique of *constraints sharing* [96] proposed by Strichman is based on the observation that since BMC is an iterative process whereby the problem is repeatedly solved
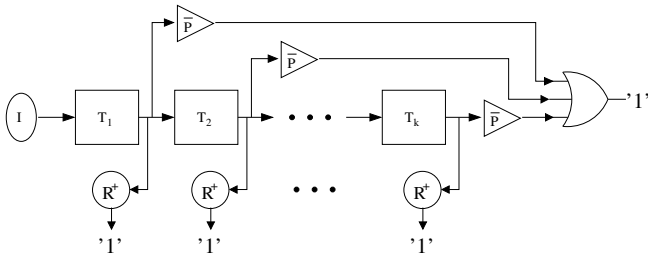
Figure 6. Improving BMC Using Reachability Over-approximation

for increasing values of the bound k, conflict clauses learned by the SAT solver in one run can potentially be used for subsequent runs instead of having to re-learn them. Specifically, any conflict clause derived *exclusively* from the sub-formula $\Phi_k = I \wedge \bigwedge_{i=0}^{k-1} T_i$ can be re-used (*i.e.*, added a-priori to the CNF) in future BMC runs with higher values of k. This technique is a specific instance of *incremental satifiability* techniques, with applications in BMC [92] and other general classes of SAT problems [61, 102]. Generally, this technique has been found to offer speed-ups of upto $2\times$ or more with negligible overhead.

A related technique called *constraints replication* [95] first identifies conflict clauses c, derived from the sub-formula $\bigwedge_{i=0}^{k-1} T_i$ alone, and creates new clauses by replacing literals of c by their time-frame shifted versions, which are then added a priori to CNFs of subsequent BMC runs. This technique is not very effective in practice, mainly due to the large overhead caused by addition of too many replicated clauses.

Recent work by Gupta *et al.* [49] proposed learning conflict clauses from BDDs, and adding them dynamically to the problem during the SAT search. The learned clauses correspond to paths to the $'0'$ terminal in a BDD representation, denoting unsatisfiable assignments on the path variables. These BDDs are created on-the-fly for heuristically selected small regions (*i.e.*, sub-circuits) in the unrolled design for BMC. They proposed several heuristics to keep the overhead low, while increasing the usefulness of the added clauses, and demonstrated significant speedups in BMC performance.

Another technique that draws upon BDD technology is the work of Cabodi *et al.* [23]. The basic idea is to use BDD-based approximate reachability analysis to *quickly* compute a *succinct* and coarse over-approximation, R^+ of the reachable state-space of a design. The BDD representing the characteristic function of R^+ is then asserted as constraints on the transition boundary between each successive pair of time-frames $i, i+1$, as shown in Figure 6. The BDDs are converted to CNF constraints which are conjoined with the BMC formulation of Equation 2. This technique does indeed have an overhead and is therefore useful primarily for larger, more difficult BMC problems. In such cases speed-ups of upto an order of magnitude have been observed.

3.4 Methodology Improvements to BMC

Although, BMC is by its intent an incomplete, bug-finding method rather than a complete verification method, a given property can be certified to be true if no counter-examples are found through BMC, upto the *sequential depth* of the circuit[15]. The sequential depth of a circuit is length of the *longest* of the shortest-paths from the initial state(s) to other reachable states of the system.

There have been a few attempts at computing or estimating the sequential depth of a circuit, to use as a target depth for BMC. Yen*et al.* [104] proposed a heuristic method based on a sampling of the state-space through random simulation. However, since the method can report an under-approximation or an over-approximation of the true sequential depth it does not provide a viable solution. Mneimneh and Sakallah [76] formulate the problem as a logical inference problem on Quantified Boolean formulas (QBF) (see also Section 6) and present a SAT-based procedure for solving the generated QBF. Unfortunately, this technique although precise, does not offer a scalable solution. Baumgartner *et al.* [10] present a structural approach based on traversing the circuit netlist to identify components with known sequential depth and using these to compute the overall sequential depth. Despite the above attempts, the problem of efficiently computing or tightly over-approximating the sequential depth of industrial size, arbitrary sequential circuits largely remains an open problem.

It is well known that different propositional encodings of the same problem can result in dramatically different run-times on a given SAT solver. The approach of *Binary Time-frame Expansion* proposed by Fallah [37] provides a different propositional encoding of the check for violation of the property in various time-frames of an unrolled circuit. The proposed encoding has been demonstrated to improve the SAT solver runtimes over the traditional formulation of Eq. 2 provided the BMC instance is sufficiently deep (typically $k \geq 100$).

3.5 Industrial Application of BMC

Several successful attempts at applying SAT-based BMC technology to industrial problems have been reported over the past few years. The original proponents of BMC reported a case study [16] where they applied BMC based on the SAT solvers SATO [105] and GRASP [71] to verify safety properties on 5 control units from the *PowerPC^{TM}* microprocessor. BMC was found to significantly outperform the BDD-based CMU SMV model checker for several of the benchmarks. Bjesse *et al.* [19] reported a significant increase in bug-finding speed and efficiency by their application of SAT-BMC (based on GRASP and CAPTAIN PROVE [90] SAT solvers), to check safety properties in the memory sub-system of the Alpha microprocessor.

A recent comprehensive analysis with respect to the performance and capacity of BMC is presented in [31]. The authors compare Intel's BDD-based model checker, *Forecast*

(adapted for BMC) with a SAT-based BMC tool, *Thunder* on several benchmarks taken from Intel's Pentium 4 processor. Their evaluation yields an interesting tie between the performance of *untuned* Thunder and *tuned* Forecast. They conclude that the real productivity gains from SAT-based BMC are obtained by obviating the need for user ingenuity and tuning effort that would be needed to obtain a *comparable* performance from a BDD-based BMC. They also report success in using SAT-based BMC on large benchmarks, that are well beyond the capacity of BDD-based tools.

A more recent study [4] compares the performance of BDD-based, SAT-based and explicit state BMC on a wide variety of industrial property checking benchmarks including both safety and liveness properties on hardware and software designs. Interestingly, they conclude that SAT-BMC is most effective at finding bugs at shallow depths (< 50) while BDD-based methods should be the method of choice for finding deep counter-examples. They also find that explicit-state BMC based on random simulation can give comparable performance to SAT-BMC in finding shallow, easier bugs for safety properties.

The general understanding and consensus in the community is that SAT-BMC tools require minimal tuning effort and work particularly well on large designs where bugs need to be searched at shallow to medium depths. In other instances it *may be possible* to extract comparable or better performance from BDD-based model checkers or other algorithms.

4 SAT-based Unbounded Model Checking

In this section we describe verification efforts that have used SAT solvers for unbounded symbolic reachability analysis, *i.e.*, methods that can prove the correctness of a property on a design as well as find counter-examples for failing properties. The method may or may not be complete. The surveyed methods fall into three categories. The first set of techniques have their roots in BDD-based symbolic state space search where the use of BDDs has been partially or completely replaced with SAT solvers. The second category comprises methods based on inductive reasoning. Inductive techniques are sound but usually incomplete in that they may not be able to prove every correct property. The third category of methods are abstraction-refinement frameworks, where SAT-based BMC is used primarily for abstraction or refinement, and is supplemented by other techniques for obtaining proofs on smaller abstract models. These frameworks also provide completeness, and offer better scalability due to effective use of abstraction. In principle, completeness can also be achieved by making the transition from SAT to QBF as is explained in Section 6.

4.1 SAT-based State Space Search

Due to the success of SAT solvers in bounded model checking, there has been growing interest in their use for *unbounded*

model checking. Here, the crucial non-trivial operation is quantifier elimination, which converts a QBF to a propositional Boolean formula. This is shown below for the image operation, which forms the computational core of symbolic methods for forward model checking, as explained in Section 2.3.

$$S_N(Y) = \exists X, W, Z. \ S_C(X) \wedge T(X, Y, W, Z) \qquad (4)$$

In this equation, the variable sets X, Y, W, Z, denote the present state, next state, input, and internal (needed for a CNF representation) variables, respectively; and S_N, S_C, and T denote the next states, the current states, and the transition relation, respectively.

4.1.1 Combination of SAT with Decision Diagrams

Abdulla *et al.* [1] formulate the checks for property satisfaction and fixpoints as SAT problems, to be solved by standard SAT solvers. The SAT problems comprise combinations of formulas S_*, representing sets of states. These are obtained by using rewriting rules for eliminating the existential quantifier in the image/pre-image operations (shown in Equation 4). The most effective rule is an *inlining rule*, which substitutes an expression for a variable to be quantified; while the most expensive is rewriting the existential quantification as a disjunction, which can result in a size blowup. They use *Reduced Boolean Circuits (RBCs)* to represent the Boolean formulas, which can be exponentially more succinct than BDDs, but are semi-canonical. A similar effort was made by Williams *et al.* [103] to use SAT solvers for CTL model checking. They too used a substitution rule very effectively for elimination of the existential quantifier. They used *Boolean Expression Diagrams (BEDs)* [5], which are closely related to RBCs, for representation of the Boolean formulas,. In addition to using standard SAT solvers to check satisfiability of BEDs, they also used the conversion of BEDs to standard BDDs. Since this conversion can blow up in practice, they used various heuristics to reduce the size of BEDs.

A different approach was taken by Gupta *et al.* [52], which integrates BDD-based techniques tightly into the SAT decision procedure. They represent the transition relation T in CNF, and the set of reachable states S_* as BDDs. For image computation, quantifier elimination is performed by using SAT techniques to enumerate all solutions to the CNF formula, and by projecting each solution on the set of image variables (Y). The search for solutions is also constrained by the BDD for S_P, using a technique called *BDD Bounding*, whereby any partial solution in SAT which is inconsistent with the BDD is regarded as a conflict. This technique is also used effectively to avoid repeating image set solutions by bounding against the current S_N. They also generate BDD-based subproblems on-the-fly, under a partially explored path in SAT. Though their procedure can be used to perform cube enumeration in SAT alone, the use of BDD subproblems is highly beneficial in handling large designs. This image computation procedure was enhanced in [51] by adding circuit structure information to the CNF formula, in order to dynamically detect and remove redundant clauses. Partition-based

SAT decision heuristics [53] were used to further improve its performance.

4.1.2 Purely SAT-based Techniques

An approach using purely SAT-based techniques was proposed by McMillan [73], for performing backward symbolic model checking (see Figure 4 in Section 2.3). It is based on computing the CNF formula equivalent to $\mathbf{AX}p$, where p is an arbitrary Boolean formula, by enumerating all satisfying assignments using a SAT solver. Variables are universally quantified by simply dropping the associated literals from the resulting CNF. Note that this forms the dual of projection for existentially quantified variables in a Disjunctive Normal Form using cubes, as used by other researchers, *e.g.*, [52,80]. Each satisfying cube is used to derive a *blocking clause* which contributes to the set of solutions, and is also added to the current database of clauses in order to avoid repetition of the solutions. The procedure for deriving a blocking clause exploits circuit structure information to re-arrange the implication graph (described in Section 2.2) when a solution (*i.e.*, a satisfying assignment) is found by the SAT solver. This re-arrangement can be viewed as a *cube enlargement* technique, which allows a larger solution cube to be captured in each enumeration by the SAT solver. The overall approach works well for designs where the sets of states can be represented compactly in CNF, and where cube enumeration with blocking clauses does not blow up.

Another model checking approach based on use of SAT techniques and *Craig interpolants* has been proposed in [74]. Given an unsatisfiable Boolean problem, and a proof of unsatisfiability derived by a SAT solver, a Craig interpolant can be efficiently computed to characterize the interface between two partitions of the Boolean problem. In particular, when no counterexample exists for depth k in BMC, *i.e.*, the SAT problem for depth k is found to be unsatisfiable, a Craig interpolant is used to obtain an over-approximation of the set of states reachable from the initial state in 1 step (or any fixed number of steps). This provides an approximate image operator, which can be used iteratively to compute an over-approximation of the set of reachable states, *i.e.*, till a fixpoint is obtained. If at any point, the over-approximate set is found to violate the given property, then the depth k is increased for BMC, till either a true counterexample is found, or the over-approximation converges without violating the property. The main advantage of the interpolant-based method is that it does not require an enumeration of satisfying assignments by the SAT solver. Indeed, the proof of unsatisfiability is used to efficiently compute the interpolant, which serves directly as the over-approximate state set. In practice too, this method has been shown to work better than other BDD-based and SAT-based complete methods. However, if the focus is only on finding bugs, *e.g.*, *falsification*, then, in the current version, it can not be faster than BMC alone.

More recently, a SAT-based quantification technique using circuit cofactoring has been proposed by Ganai *et al.* [40]. They too use a SAT solver to enumerate solutions, but they use circuit cofactoring after each enumeration to capture a larger set of new state cubes per enumeration, in comparison to cube-wise enumeration techniques. Note that in general a cofactor can capture not just a single cube, but several cubes. This is greatly beneficial in reducing the total number of solutions enumerated by SAT, sometimes by several orders of magnitude, in comparison to approaches based on blocking clauses (described above). They also use an efficient circuit graph representation for the solution states [64], which is more robust than CNF-based or BDD-based representations, and use a hybrid SAT solver [41] to directly work on these representations. Ganai et al.'s quantification technique can be used to compute exact image/pre-image state sets, unlike the interpolant-based technique (described above) which computes approximate state sets. It has been used in SAT-based unbounded symbolic model checking to handle many difficult industry examples, which could not be handled by either BDDs or blocking-clause based SAT approaches.

4.2 SAT-based Inductive Reasoning

Inductive reasoning can be a cheap and efficient means of verifying properties, rather than simply finding counter-examples as in BMC. Inductive reasoning has previously been used, with some success, for various verification problems, including property checking using technologies such as BDDs. The inductive proof for verifying a property $\mathcal{P} = \mathbf{AG}p$ can be derived using a SAT solver by checking the formulas ϕ_{base} (the base case) and ϕ_{induc} (the induction step) for unsatisfiability.

$$\phi_{base} = I \wedge \neg P_0 \tag{5}$$
$$\phi_{induc} = P_k \wedge T(k, k+1) \wedge (\neg P_{k+1})$$

If ϕ_{induc} is unsatisfiable the property $\mathcal{P}$ is called an *inductive invariant*. Both formulas, if unsatisfiable, provide a sufficient (but not necessary) condition for verifying $\mathcal{P}$. However, the above form of induction, known as *simple induction*, is not powerful enough to verify many properties.

Two recent works [18,89] have proposed the use of more powerful forms of induction known as *induction with depth* and *unique states induction* to verify safety properties. For induction with depth n the formulas of Equation 5 become:

$$\phi_{base}^n = I \wedge \left(\bigwedge_{i=0}^{n-1} T(i, i+1) \right) \wedge \bigvee_{i=0}^{n} \neg P_i \tag{6}$$
$$\phi_{induc}^n = \left(\bigwedge_{j=k}^{k+n} P_j \right) \wedge \left(\bigwedge_{i=k}^{k+n} T(i, i+1) \right) \wedge \neg P_{k+n+1}$$

Essentially, induction with depth corresponds to strengthening the induction hypothesis, by imposing the original induction hypothesis (P_k in ϕ_{induc}, Equation 5) on n consecutive time-frames. This can be further strengthened by requiring that the states appearing on each time-frame be unique

(*unique states induction*). This restriction results in a complete method for simple safety properties. However, the induction length may be as long as the recurrence diameter [15], which in most cases is much longer than the sequential depth. Further, the number of constraints needed to enforce the state uniqueness is quadratic in the depth of unrolling, *i.e.*, the induction depth, resulting in very large CNFs. In recent work [35], Eén *et al.* partly address this issue by proposing an iterative method for induction. The induction hypothesis starts off without any uniqueness constraints, which are gradually added in successive iterations till the induction proof goes through. The efficiency of the method is further improved by using an *incremental SAT* mechanism that allows sharing of conflict clauses (recorded by the SAT solver) between successive iterations of induction.

Another variant of this line of research is the work by Gupta *et al.* [48], which is similar to the work by Cabodi *et al.* [23], discussed in Section 3.3. As in [23], BDD-based techniques are used to *efficiently* compute a *succinct* over-approximation R^+ of the reachable states of a design. This is used to strengthen the induction hypothesis by imposing R^+ as an additional reachability invariant. In particular, it constrains the state values that are allowed to appear at the starting state of the induction step (or at the interfaces between each successive pair of time-frames). Note that in contrast to [23], the constraints here are not redundant, but are added to strengthen the induction hypothesis, which might be too weak with the property alone. This frequently allows induction proofs to go through successfully. A related line of research is based on generating an inductive invariant to be used as over-approximation for the reachable states in the context of sequential equivalence checking [18,99,94].

One of the original papers on SAT-BMC [16] had proposed the use of simple induction as a cheap and simple first pass to apply to all property checking instances before resorting to more comprehensive verification/falsification methods. The above powerful variants of induction undoubtedly enlarge the range of properties verifiable through inductive reasoning. At the same time they can produce very large SAT formulas which are very resource intensive to solve. Hence the real utility of these methods would only be brought out by a good verification methodology that uses them with the right trade-off between verification power and efficiency, and in the right balance with BDD-based verification techniques. Recent work by Li *et al.* [67] points in this direction as well. In this work the authors use SAT-based unique-states induction with depth as the model checking method in an abstraction refinement framework (discussed in the next section). They observe that the efficacy of SAT-based induction is considerably enhanced when used within such a framework. Further, even within this framework the SAT-based induction exhibits complementary strengths compared to a traditional BDD-based model checker, underscoring the need for a combined proof technique.

4.3 SAT-based Abstraction-Refinement Frameworks

In order to handle large designs, there has been a great deal of interest in the use of abstraction and refinement techniques for verification. Most efforts are refinement-based approaches, where starting from a small abstract model of the concrete design, counterexamples found on these models are used to refine them iteratively until either a conclusive result is obtained by conservative model checking, or the resources are exhausted [79]. One of the first attempts to use SAT solvers for counterexample guided abstraction refinement (CEGAR) was described by Clarke *et al.* [29]. In their approach, the SAT solver is used to check whether a counterexample trace found during model checking of the abstract model is spurious or not, by effectively checking its satisfiability on the concrete design. If it is spurious, ILP (integer linear programming) and machine learning techniques are used to perform the refinement. In a subsequent effort [25], they used SAT-based techniques for performing this refinement as well. In particular, they proposed heuristics using the SAT decision scores to pick refinement candidates among hidden (abstracted away) latches. A more interesting technique used ideas similar to a SAT solver's proof of unsatisfiability, in order to identify latches that are *sufficient* to exclude the spurious counterexample.

Another recent method for counterexample guided abstraction refinement has been proposed by Wang *et al.* [101]. They use BDDs to represent multiple abstract counterexamples, which are checked for satisfiability on the concrete design using a SAT solver interfaced with BDD constraints [48]. Rather than refining each counterexample individually, they propose a game-theoretic refinement procedure, that attempts to exclude multiple counterexamples simultaneously. In practice, their method performs better than other methods based on refining a single counterexample at a time.

One reason for the popularity of counterexample guided abstraction refinement approaches has been a lack of techniques that could extract relevant information from a relatively large concrete design. This is changing now with the use of proof analysis techniques for SAT solvers. These techniques can be easily used to identify a set of clauses from the original problem, called the *unsatisfiable core* [109,75], such that the clauses are sufficient for implying unsatisfiability. These unsatisfiable cores form the basis of two recent independent efforts on abstraction methods using SAT-based BMC [75,50]. In both methods, an abstract model is obtained from the unsatisfiable core, identified from an unsatisfiable BMC instance at depth k. This abstract model has the useful property that it does not have any counterexamples of depth less than or equal to k. The basis for abstraction is the intuition that after k is large enough, the corresponding abstract model may exclude counterexamples of all lengths. The usefulness of the abstraction stems from the empirical evidence that for typical verification applications, the unsatisfiable cores and the corresponding abstract models are much smaller than the concrete designs. There are minor differences in the abstraction methods used by these two ap-

proaches – McMillan and Amla use a gate-level abstraction, while Gupta *et al.* use a latch-level abstraction. However, the major differences are in their application settings.

In McMillan and Amla's approach, a proof of correctness is attempted for the abstract model derived from BMC on the concrete design itself. If a counterexample is found, the BMC depth k is increased, till either a true counterexample is found on the concrete design, or correctness of the derived abstract model is proved. They demonstrated many successful applications of their approach on various benchmark examples. However, this approach runs into scalability problems if either the abstract model is too large for unbounded verification, or SAT-based BMC cannot be completed on the concrete model at the increased depth k.

In contrast, the approach by Gupta *et al.* [50] proposes an *iterative abstraction* framework, which is targeted at iteratively reducing the size of the abstract models, starting from the concrete model. In each iteration, BMC is performed for increasing depths on the chosen model. If there is no counterexample (up to some heuristically chosen depth), a proof-based abstraction procedure is used to abstract the model further for the next iteration. In this framework, a proof of correctness on an abstract model in any iteration, guarantees correctness on the concrete design. On the other hand, a counterexample may require a refinement (if it is spurious), based on either the specific counterexample, or a deeper BMC analysis on a less abstract model from a previous iteration. In practice, the abstraction loop is iterated up to convergence in the size of the abstract model. The successive reductions in abstract model sizes, typically by two orders of magnitude across all iterations, was crucial for successful verification of large industry designs.

In practice, refinement-based approaches which start from a small abstract model may require many iterations before converging on a model where the proof succeeds. More frequently, the size of the refined abstract model grows monotonically larger, on which unbounded verification methods fail to complete. On the other hand, abstraction-based approaches which start from the given concrete model may need to handle much larger models. However, note that they do not require complete verification on these larger models for the purpose of abstraction. We believe there is likely to be more activity in exploring useful combinations of these approaches.

5 ATPG-based Model Checking

Concurrent with the development of SAT methods for model checking there has been a growing interest in applying tools for automatic test pattern generation (ATPG) of sequential circuits to the model checking problem. ATPG tools for sequential circuits (abbreviated as *sequential ATPG* tools in the sequel) are designed to search for input sequences to the given circuit that can test for the presence of a certain fault. A sub-task in this process involves performing a search on the space of input sequences of the sequential machine for an
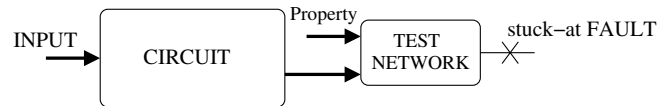

Figure 7. Sequential ATPG for Model Checking

input sequence that will excite a certain signal in the circuit to logic 1 or 0. In this respect the core problems solved by sequential ATPG algorithms are very similar to those expressed by SAT formulations of circuit problems. Historically, the key difference between ATPG algorithms and CNF-based SAT solvers has been that the former perform branch-and-bound search on a structural circuit representation rather than a CNF database. This allows ATPG tools to implement efficient decision heuristics based on the circuit structure and to model and deal with real-world circuit primitives, such as tri-state buses and high-impedance logic values. Further, unlike SAT-based bounded model checking, sequential ATPG tools do not need to explicitly replicate the circuit structure when performing sequential reasoning. More importantly, the latter can be used to implement *complete* property checking algorithms while the former essentially perform a bounded check. On the other hand, sequential ATPG tools have traditionally lacked techniques such as conflict-based learning and efficient Boolean constraint propagation (implication generation) which are the main sources of power and efficiency of modern SAT solvers.

Boppana *et al.* [20] were the first to recognize the relative advantages of sequential ATPG solvers over conventional CNF-SAT solvers and to employ a sequential ATPG tool to check safety properties on circuits. The basic idea of this formulation, as shown in Figure 7, is to construct a test network, based on the property, and place a fault on the output of this test network such that this fault is testable if and only if there exists a counter-example for the specified property. A key contribution of this work was to recognize that most sequential designs have *synchronizing sequences* which cause the FSM (finite state machine) to reach a specific state (say s_0) regardless of the starting state. Using this, the checking of safety properties of the form **AG EF**p can be reduced to verifying **EF**$_{s_0}p$. The authors showed that since there is no explicit storage of states in each time-frame (like BDD-based model checkers) a sequential ATPG-based model checker could outperform conventional BDD-based model checkers in several cases.

In another work [2], the authors propose a method of *bounded* model checking using a sequential ATPG tool. Unlike the formulation of [20] where the test network is a combinational network based on the property p, this work supports both safety and liveness properties and the test network is a monitor FSM based on both the property and the bound n, the number of time-steps. A fault specified on the output of this monitor is testable if and only if the given property has a counter-example within n time-steps of the initial state. The authors report impressive speed-ups and memory savings, compared to a Cadence-SMV BMC based on zChaff.

In related work, Sheng *et al.* [91] have successfully used a sequential ATPG tool based on simulation and genetic algorithms for checking safety properties. They observe that such a tool, while not suitable for *verification* per se, can be very effective in finding bugs. Huan and Cheng [56] have used a combination of structural, word-level ATPG and modular arithmetic constraint solving techniques to check safety properties.

While a recent experimental comparison between SAT and ATPG based BMC approaches [78] found no real performance gap between the two formulations, our experience has shown that model checking approaches based on ATPG tools and CNF-SAT solvers really have orthogonal strengths. Thus, ATPG-based model checkers can be superior to SAT-BMC on certain benchmarks and vice versa. Current research is aimed at producing a tool that combines the benefits of both types of engines. The hybrid SAT solver [41] discussed in Section 2.2 is one such attempt. Another significant step in this direction has been reported by Iyer *et al.* in the SATORI solver [59]. SATORI is a complete algorithm for sequential Boolean reasoning. It is based on algorithms and techniques available in modern sequential ATPG tools which have been augmented with *some flavor* of the techniques (*e.g.*, efficient BCP and conflict-driven learning) available in modern SAT solvers.

6 QBF

Checking the satisfiability of the more expressive logic of *Quantified Boolean Formulae* (QBF), is equivalent to symbolic reachability and thus sequential property checking. This fact, in principle, can be used to obtain a QBF-based model checking algorithm. In the future, QBF may play the same role for sequential property checking or model checking, as SAT does today for combinational property checking. Therefore, we briefly explain the connection between QBF and model checking and show how SAT based techniques for unbounded model checking relate to QBF. We conclude the section with an overview on the state-of-the-art of algorithms and implementations for solving QBF, which naturally are very similar to those used for SAT.

6.1 QBF for Model Checking

The logic of Quantified Boolean Formulae (QBF) is a generalization of propositional logic, the input language of SAT solvers, that allows Boolean variables to be quantified. Typical examples are the following two formulas

$$\forall x[\exists y[x \leftrightarrow y]] \quad \text{and} \quad \exists y[\forall x[x \leftrightarrow y]],$$

the first formula being *true* and the second evaluating to *false*. This already highlights the most important difference between QBF and SAT: the quantification order of the variables in which the formula is evaluated matters. Note, that a propositional formula is an instance of a QBF formula with only existentially quantified variables.

Checking satisfiability for QBF, which we also abbreviate as QBF, generalizes SAT and is a PSPACE complete problem [93]. QBF is expected to have exponential complexity and to be strictly harder than NP. It has been observed in [83] that the symbolic reachability problem is PSPACE complete as well. Thus there exist polynomial reductions from sequential property checking to QBF and vice versa. For our purpose of using QBF solvers for sequential property checking we give a translation of checking safety properties to checking satisfiability of QBF. The other direction, *i.e.*, translating QBF to symbolic reachability, can be obtained from [34] in combination with [84].

Consider the forward safety checking algorithm of Figure 3 in Section 2.3. Here, the check to see if a bad state (violating the safety property) is contained in the set of states reached in the current iteration can be formulated as a SAT problem, since all the quantifiers introduced are of the same type. Only in the termination check an alternation of quantifiers occurs.

If we use QBF to represent sets of states symbolically we get a complete procedure. Instead of performing an image computation in each iteration of the algorithm, as in classical BDD based algorithms [22,72] or in modern SAT based fixpoint algorithms [73] (see Section 4.1.2) the image can be represented symbolically with QBF by introducing quantifiers without eliminating them. Then the termination check of the fixpoint algorithm becomes an instance of checking satisfiability of QBF (see Section 4 of [80]). Regardless of the fact that the implementation of QBF solvers is still in its infancy, this approach has the drawback that the number of iterations of the fixpoint algorithm can only be bounded by the diameter of the model and is therefore exponential in the size of the model, in the worst case.

Starting from iterative squaring [22] an even linear reduction can be obtained. The original idea of iterative squaring is to compute the transitive closure T^* of the transition relation T according to the following equation:

$$T^{2i}(s,t) \equiv \exists s'[T^i(s,s') \wedge T^i(s',t)] \tag{7}$$

Let the model consists of n state bits (flip-flops). Then $T^* \equiv T^m$ with $m = 2^n$. Thus after n applications of the above equation T^* is obtained. If the first argument of T^* can be restricted to the set of initial states, the resulting expression is the characteristic function of the set of reachable states. In order to keep the exposition simple, we assume that the transition relation contains the identity relation. This is equivalent to assuming that the model may always stall.

Originally BDDs would have been used for representing T^i, with the problem that an additional set of variables is involved when eliminating the quantifier in Equation 7 compared to standard image computation as in Equation 1. Additionally, the approximations to the transitive closure of the transition relation can not be kept in partitioned form, which results in very large BDDs. Moreover there is no obvious way to restrict the computation to the reachable state space. These problems may be the reasons that iterative squaring

was not used much in practice when using BDDs. In the context of SAT, iterative squaring was successfully applied in [103], by assuming determinism except for the choice of the initial state.

In principle, in order to get a QBF based algorithm, we only need to apply the above equation n times and add the initial state constraints to obtain a propositional formula that represents all reachable states. However, the parameter lists of the two occurrences of T^i in the body of the quantifier are different, and thus applying the equation usually involves doubling the size of the formula. Additionally, there is almost no chance for sharing common subformulae. With BDDs T^i may potentially be represented by a small BDD. Also one can hope that the quantifier eliminations are cheap. But directly using the above equation to generate QBF is not better than unrolling the transition relation as in BMC.

By using QBF as presented in [83] (see also [82]) the effect of copying T^i twice can be avoided. We introduce a new universally quantified variable c, that determines which of the two parameter lists should be used.

$$T^{2i}(s,t) \equiv \exists s'[\forall c[\exists l,r[(c \to \overbrace{(l,r)=(s,s')}^{\text{use left parameter list}}) \wedge$$
$$(\overline{c} \to \underbrace{(l,r)=(s',t)}_{\text{use right parameter list}}) \wedge T^i(l,r)]]]$$

Now applying this equation n times merely introduces $3n$ existentially quantified state bit vectors and n universally quantified variables and needs only one copy of the transition relation. The result is not propositional anymore, since it involves $2n$ alternations of quantifiers but it is quadratic in the number of state bits n. It still needs to be seen, whether this formulation of symbolic reachability is beneficial in practice, but clearly there is a potential for an exponential speedup, compared to current algorithms.

6.2 QBF Solvers

The efficiency of the approach described above relies on efficient implementations of QBF solvers. We briefly give an overview on the state-of-the-art in QBF solvers. For more details the interested reader is referred to the report on the evaluation of QBF solvers for the SAT'2003 conference [12]. First note that techniques for image computation in model checking can be interpreted as QBF decision procedures, since they essentially provide a quantifier elimination procedure. As already described, these techniques are based on BDDs as in traditional symbolic model checking [22], based on structural methods [99,94], directly based on SAT [73,74], or based on combination of SAT and other decision diagrams[1,52,103].

An enumeration based explicit QBF decision procedure, which mentions applications to model checking, and is similar to the SAT based image computation of McMillan [73] discussed in Section 4.1.2, can be found in [80]. Related to the explicit quantifier expansion in [1,103] the structural QBF

algorithm in [7] expands quantifiers by copying and substitution:

$$\forall x[f] \equiv f[0/x] \wedge f[1/x]$$

Earlier attempts [24] for QBF are based on DPLL. In essence, an algorithm similar to that of Figure 1 can be used. There are two major differences. First, decision variables can only be picked in the same order as they occur in the quantifier prefix. Second, for a universal variable x it is required that both sub-problems, assigning $x = 0$ and $x = 1$, have to return *satisfiable*.

Recently, several groups [108,43,66] independently applied techniques from the SAT domain to QBF, such as the conflict-driven learning and conflict-driven backtracking techniques discussed in Section 2.2. However, QBF provides the opportunity to learn not only conflict clauses, but also models for speeding up the search for universally quantified variables. Finally, there is the notion of q-resolution [62], which in principle gives a complete decision procedure for QBF, but, for the same reason as the resolution-based Davis & Putnam procedure [33], requires too much memory in practice. However, more recently it has been observed in [13] that the combination of q-resolution with expansion can lead to an efficient QBF decision procedure, which in many cases outperforms DPLL style solvers.

The structure of the QBF problem is much richer than the simpler SAT problem. More optimizations are possible and probably also necessary. Algorithms and tools are not as mature as for SAT. Implementations of QBF solvers are steadily improving. A large set of benchmarks is available[1] and a standard input format exists (QDIMACS).

As mentioned earlier, the restriction on the order of decision variables is the major difference between QBF and SAT. In practice, this also seems to severely restrict the size of the problems that can be handled. However, a QBF formulation of a problem may be exponentially more succinct. Thus there is a potential for an exponential speedup using QBF solvers, which may have a large impact in the context of sequential property checking.

7 SAT-based Verification: Achievements & Challenges

There has been significant progress in the area of SAT-based verification over the last decade. However, much remains to be done to make this technology more pervasive in industrial design verification flows. In the following we discuss the notable achievements and major challenges in SAT-based verification.

7.1 Achievements

The single most important achievement of SAT-based verification has been its emergence as an orthogonal technology to BDD-based model checking techniques (bounded and unbounded). This means that there are several instances where

---

[1] http://www.qbflib.org

one technology significantly out-performs the other and vice versa. Further, SAT-based techniques have been found to be less sensitive to the problem size and typically require much less user tuning of parameters. Hence, such methods are capable of verifying much larger systems than those typically handled by BDDs, and of enhancing productivity by obviating the need for user ingenuity and tuning effort. Bounded Model checking (BMC) based on SAT methods has been found to be particularly effective at generating counterexamples for hard-to-find bugs at short to medium depths (upto depth $50 - 60$) of sequential behavior.

7.2 Challenges

While SAT-based verification methods have proven to be orthogonal to their BDD counterparts there is very little fundamental understanding of their respective strengths. In this respect, a major challenge is to develop a *verification methodology which employs both SAT and BDD methods* in a manner best suited to utilize their respective strengths.

The strength of SAT-based verification techniques lies primarily in falsification. BDD-based symbolic model checking continues to be the de-facto standard for verifying properties. Several attractive techniques for *SAT-based unbounded model checking (UMC)* have been proposed in recent years, including methods for SAT-based state space traversal, inductive reasoning and iterative abstraction refinement surveyed in Section 4. As it currently stands this body of research is rich in promising ideas but somewhat immature. For example, among the several variants of induction, the most scalable ones (*e.g.*, simple induction) are too weak to prove most properties while the most comprehensive ones (*e.g.*, unique states induction with depth) may not be applicable to the largest designs. Therefore, further research is needed to develop SAT-based unbounded model checking into a viable alternative to BDD-based symbolic model checking.

SAT-based BMC is currently used as a falsification technique. However, as pointed out the original proponents of SAT-based BMC [26], the technique can be used to formally verify properties by performing the BMC check upto the sequential circuit depth (or some over-approximation thereof). Recently, there have been several attempts at the problem of *computing the sequential depth* of a given system. Tight over-approximations would also be valuable. But the inherent intractability of the problem has frustrated attempts at finding a general, scalable solution. Nevertheless, such a solution, if discovered, would greatly enhance both the efficacy and applicability of SAT-based BMC techniques.

This issue lies at the very core of the SAT-BMC formulation. Since the BMC formulation uses an explicit unrolling of time-frames, the generated SAT formulas can become too large, and hence unsolvable for large sequential depths. A desirable solution to this is the development of a sequential reasoning engine that implements all the features of modern SAT solvers but does not require explicit unrolling of the circuit. A first attempt on *implicit time-frame unrolling* has been made

in the SATORI solver [59] which partly integrates ideas from modern SAT solvers and sequential ATPG tools.

As the simplicity of the SAT problem helped to improve algorithms and implementations for combinational property checking, including BMC, QBF may play the same role for (complete) sequential property checking. Thus improvements to the capacity of QBF solvers will have a large impact on sequential property verification, in addition to providing an important research topic on its own.

While it is clear that SAT-based verification techniques will continue to make inroads into current verification methodology and tools, in our opinion the satisfactory solution of some of the issues discussed above will be crucial in making this technology more widely applicable. In any case, the next few years promise to be an interesting time for researchers working in this area as well for tool developers seeking to integrate this technology into the next generation of verification tools in industrial practice.

References

1. Parosh A. Abdulla, Per Bjesse, and Niklas Eén. Symbolic Reachability Analysis Based on SAT-Solvers. In Susanne Graf and Michael Schwartzbach, editors, *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1785 of *Lecture Notes in Computer Science*, pages 411–425. Springer, March 2000.

2. Jacob A. Abraham, Vivekananda M. Vedula, and Daniel G. Saab. Verifying Properties Using Sequential ATPG. In *Proceedings of the International Test Conference (ITC)*, pages 194–202, October 2002.

3. Rajeev Alur. Timed Automata. In Nicolas Halbwachs and Doron Peled, editors, *Proceedings of the 11th International Conference on Computer Aided Verification (CAV)*, volume 1633 of *Lecture Notes in Computer Science*, pages 8–22. Springer, July 1999.

4. Nina Amla, Robert Kurshan, Kenneth McMillan, and Ricardo Medel. Experimental Analysis of Different Techniques for Bounded Model Checking. In Hubert Garavel and John Hatcliff, editors, *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2619 of *Lecture Notes in Computer Science*, pages 34–48. Springer, April 2003.

5. Henrik R. Andersen and Henrik Hulgaard. Boolean Expression Diagrams. *Information and Computation*, 179(2):194–212, December 2002.

6. Abdelwaheb Ayari and David Basin. Bounded model construction for monadic second-order logics. In E. Allen Emerson and A. Prasad Sistla, editors, *Proceedings of the 12th International Conference on Computer-Aided Verification (CAV)*, number 1855 in Lecture Notes in Computer Science, pages 99–113. Springer, July 2000.

7. Abdelwaheb Ayari and David Basin. QUBOS: Deciding Quantified Boolean Logic using Propositional Satisfiability Solvers. In Mark Aagard and John W. O'Leary, editors, *Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, volume 2517 of *Lecture Notes in Computer Science*, pages 187–201. Springer, 2002.

8. Thomas Ball and Sriram K. Rajamani. The SLAM project: Debugging system soft-ware via static analysis. In *Proceedings of the 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 1–3. ACM, January 2002.

9. Clark W. Barrett, David L. Dill, and Aaron Stump. Checking Satisfiability of First-Order Formulas by Incremental Translation to SAT. In Ed Brinksma and Kim G. Larsen, editors, *Proceedings of the 14th International Conference on Computer Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, pages 236–249. Springer, July 2002.

10. Jason Baumgartner, Andreas Kuehlmann, and Jacob A. Abraham. Property Checking via Structural Analysis. In Ed Brinksma and Kim G. Larsen, editors, *Proceedings of the 14th International Conference on Computer Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, pages 151–165. Springer, July 2002.

11. Roberto J. Bayardo and Robert C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 203–208, July 1997.

12. Daniel Le Berre, Laurent Simon, and Armando Tachella. Challenges in the QBF Arena: the SAT'03 Evaluation of QBF Solvers. In Enrico Giunchiglia and Armando Tacchella, editors, *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 2919 of *Lecture Notes in Computer Science*, pages 468–485. Springer, May 2004.

13. Armin Biere. Resolve and Expand. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, May 2004.

14. Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Masahiro Fujita, and Yunshan Zhu. Symbolic Model Checking using SAT procedures instead of BDDs. In *Proceedings of the 36th Design Automation Conference (DAC)*, pages 317–320, June 1999.

15. Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic Model Checking without BDDs. In Rance Cleaveland, editor, *Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer, March 1999.

16. Armin Biere, Edmund Clarke, Richard Raimi, and Yunshan Zhu. Verifying safety properties of a PowerPC microprocessor using symbolic model checking without BDDs. In Nicolas Halbwachs and Doron Peled, editors, *Proceedings of the 11th International Conference on Computer Aided Verification (CAV)*, volume 1633 of *Lecture Notes in Computer Science*, pages 60–71. Springer, July 1999.

17. Armin Biere, Edmund M. Clarke, and Yunshan Zhu. Multiple State and Single State Tableaux for Combining Local and Global Model Checking. In Ernst-Rüdiger Olderog and Bernhard Steffen, editors, *Correct System Design, Recent Insight and Advances*, volume 1710 of *Lecture Notes in Computer Science*, pages 163–179. Springer, 1999.

18. Per Bjesse and Koen Claessen. SAT-based Verification without State Space Traversal. In Warren A. Hunt Jr. and Steven D. Johnson, editors, *Proceedings of the 3rd International Conference on Formal Methods in Computer Aided Design (FM-CAD)*, volume 1954 of *Lecture Notes in Computer Science*, pages 372–389. Springer, November 2000.

19. Per Bjesse, Tim Leonard, and Abdel Mokkedem. Finding Bugs in an Alpha Microprocessor Using Satisfiability Solvers.

In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Proceedings of the 13th International Conference on Computer Aided Verification (CAV)*, volume 2102 of *Lecture Notes in Computer Science*, pages 454–464. Springer, July 2001.

20. Vamsi Boppana, Sreeranga P. Rajan, Koichiro Takayama, and Masahiro Fujita. Model Checking Based on Sequential ATPG. In Nicolas Halbwachs and Doron Peled, editors, *Proceedings of the 11th International Conference on Computer-Aided Verification (CAV)*, volume 1633 of *Lecture Notes in Computer Science*, pages 418–430. Springer, July 1999.

21. Randal E. Bryant. Graph Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C(35):677–691, August 1986.

22. Jerry R. Burch, Edmund M. Clarke, David E. Long, Kenneth L. McMillan, and David L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(4):401–424, April 1994.

23. Gianpiero Cabodi, Sergio Nocco, and Stefano Quer. Improving SAT-based Bounded Model Checking by Means of BDD-based Approximate Traversals. In *Proceedings of the Design Automation and Test in Europe (DATE)*, pages 898–903, March 2003.

24. Marco Cadoli, Andrea Giovanardi, and Marco Schaerf. An Algorithm to Evaluate Quantified Boolean Formulae. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*, pages 262–267, July 1998.

25. Pankaj Chauhan, Edmund M. Clarke, James Kukula, Samir Sapra, Helmut Veith, and Dong Wang. Automated Abstraction Refinement for Model Checking Large State Spaces using SAT based Conflict Analysis. In Mark Aagaard and John W. O'Leary, editors, *Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design (FM-CAD)*, volume 2517 of *Lecture Notes in Computer Science*, pages 33–51, November 2002.

26. Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded Model Checking Using Satisfiability Solving. *Formal Methods in System Design*, 19(1):7–34, July 2001. Kluwer Academic Publishers.

27. Edmund M. Clarke and E. Allen Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In Dexter Kozen, editor, *Proceedings of the Workshop on Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1982.

28. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 2000.

29. Edmund M. Clarke, Anubhav Gupta, James Kukula, and Ofer Strichman. SAT-based Abstraction Refinement Using ILP and Machine Learning Techniques. In Ed Brinksma and Kim G. Larsen, editors, *Proceedings of the 14th International Conference on Computer Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, pages 265–279. Springer, July 2002.

30. Edmund M. Clarke and Bernd-Holger Schlingloff. Model Checking. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 24, pages 1635–1790. Elsevier and MIT Press, 2001.

31. Fady Copti, Limor Fix, Ranan Fraer, Enrico Giunchiglia, Gila Kamhi, Armando Tacchella, and Moshe Y. Vardi. Benefits of Bounded Model Checking in an Industrial Setting. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Proceedings of the 13th International Conference on Computer Aided Ver-*

ification (CAV), volume 2102 of *Lecture Notes in Computer Science*, pages 436–453. Springer, July 2001.

32. Martin Davis, George Logemann, and Donald Loveland. A Machine Program for Theorem-Proving. *Communications of the ACM*, 5(7):394–397, July 1962.

33. Martin Davis and Hilary Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM*, 7(3):201–215, July 1960.

34. Francesco M. Donini, Paolo Liberatore, Fabio Massacci, and Marco Schaerf. Solving QBF with SMV. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 578–589, 2002.

35. Niklas Eén and Niklas Sörensson. Temporal Induction by Incremental SAT Solving. In Ofer Strichman and Armin Biere, editors, *Proceedings of the First International Workshop on Bounded Model Checking (BMC)*, volume 89 of *Electronic Notes in Theoretical Computer Science*. Elsevier, July 2003.

36. E. Allen Emerson. *Temporal and modal logic*, volume B, chapter 16, pages 995–1072. MIT Press, 1990.

37. Farzan Fallah. Binary Time-Frame Expansion. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, pages 458–464, November 2002.

38. Hideo Fujiwara and Takeshi Shimono. On the acceleration of test generation algorithms. *IEEE Transactions on Computers*, C-32:1137–1144, December 1983.

39. Malay K. Ganai and Adnan Aziz. Improved SAT-based Bounded Reachability Analysis. In *Proceedings of the 15th International Conference on VLSI Design (VLSID)*, pages 729–734, January 2002.

40. Malay K. Ganai, Aarti Gupta, and Pranav Ashar. Efficient SAT-based Unbounded Symbolic Model Checking using Circuit Cofactoring. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, November 2004.

41. Malay K. Ganai, Lintao Zhang, Pranav Ashar, and Aarti Gupta. Combining Strengths of Circuit-based and CNF-based Algorithms for a High Performance SAT Solver. In *Proceedings of the 39th Design Automation Conference (DAC)*, pages 747–750, June 2002.

42. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman, San Francisco, 1979.

43. Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Learning for Quantified Boolean Logic Satisfiability. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, pages 649–654, July 2002.

44. Prabhakar Goel. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Transactions on Computers*, C-30:215–222, March 1981.

45. Evgueni Goldberg and Yakov Novikov. BerkMin: a Fast and Robust Sat-Solver. In *Proceedings of Design Automation and Test in Europe (DATE)*, pages 142–149, March 2002.

46. Evgueni Goldberg and Yakov Novikov. Verification of Proofs of Unsatisfiability for CNF Formulas. In *Proceedings of the Design Automation and Test in Europe (DATE)*, pages 886–891, March 2003.

47. Evgueni Goldberg, Mukul R. Prasad, and Robert K. Brayton. Using SAT for Combinational Equivalence Checking. In *Proceedings of Design Automation and Test in Europe (DATE)*, pages 114–121, March 2001.

48. Aarti Gupta, Malay Ganai, Chao Wang, Zijiang Yang, and Pranav Ashar. Abstraction and BDDs Complement SAT-based BMC in Di*Ver*. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Proceedings of the 15th International Conference on Computer-Aided Verification (CAV)*, volume 2725 of *Lecture Notes in Computer Science*, pages 206–209. Springer, July 2003.

49. Aarti Gupta, Malay Ganai, Chao Wang, Zijiang Yang, and Pranav Ashar. Learning from BDDs in SAT-based Bounded Model Checking. In *Proceedings of the 40th Design Automation Conference (DAC)*, pages 824–829, June 2003.

50. Aarti Gupta, Malay Ganai, Zijiang Yang, and Pranav Ashar. Iterative Abstraction Using SAT-based BMC with Proof Analysis. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, pages 416–423, November 2003.

51. Aarti Gupta, Anubhav Gupta, Zijiang Yang, and Pranav Ashar. Dynamic Detection and Removal of Inactive Clauses in SAT with Application in Image Computation. In *Proceedings of the 38th Design Automation Conference*, pages 536–541, June 2001.

52. Aarti Gupta, Zijiang Yang, Pranav Ashar, and Anubhav Gupta. SAT Based State Reachability Analysis and Model Checking. In Warren A. Hunt and Steven D. Johnson, editors, *Proceedings of the 3rd International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, volume 1954 of *Lecture Notes in Computer Science*, pages 354–371, November 2000.

53. Aarti Gupta, Zijiang Yang, Pranav Ashar, Lintao Zhang, and Sharad Malik. Partition-Based Decision Heuristics for Image Computation Using SAT and BDDs. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, pages 286–292, November 2001.

54. Thomas A. Henzinger, Orna Kupferman, and Shaz Qadeer. From *Pre*-historic to *Post*-modern symbolic model checking. In Alan J. Hu and Moshe Y. Vardi, editors, *Proceedings of the 10th International Conference on Computer-Aided Verification (CAV)*, volume 1427 of *Lecture Notes in Computer Science*, pages 195–206. Springer, July 1998.

55. Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

56. Chung-Yang Huan and Kwang-Ting Cheng. Using Word-Level ATPG and Modular Arithmetic Constraint-Solving Techniques for Assertion Property Checking. *IEEE Trnsactions on Computer-Aided Design*, 20(3):381–391, March 2001.

57. Hiroaki Iwashita and Tsuneo Nakata. Forward Model Checking Techniques Oriented to Buggy Designs. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pages 400–404, November 1997.

58. Hiroaki Iwashita, Tsuneo Nakata, and Fumiyasu Hirose. CTL model checking based on forward state traversal. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pages 82–87, November 1996.

59. Madhu K. Iyer, Ganapathy Parthasarathy, and Kwang-Ting Cheng. SATORI-A Fast Sequential SAT Engine for Circuits. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pages 320–325, November 2003.

60. Daniel Jackson and Mandana Vaziri. Finding bugs with a constraint solver. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, pages 14–25, August 2000.

61. Joonyoung Kim, Jesse Whittemore, and Karem Sakallah. On Solving Stack-Based Incremental Satisfiability Problems. In

Proceedings of the International Conference on Computer Design (ICCD), pages 379–382, October 2000.

62. Hans Kleine Büning, Marek Karpinski, and Andreas Flgel. Resolution for quantified boolean formulas. *Information and Computation*, 117(1):12–18, February 1995.

63. Hans Kleine Büning and Theodor Lettmann. *Propositional logic: Deduction and Algorithms*, volume 48 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1999. ISBN-0-521-63017-7.

64. Andreas Kuehlmann, Viresh Paruthi, Florian Krohm, and Malay K. Ganai. Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(12):1377–1394, December 2002.

65. Tracy Larrabee. Test Pattern Generation Using Boolean Satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(1):4–15, January 1992.

66. Reinhold Letz. Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas. In Uwe Egly and Christian G. Fermüller, editors, *Proceedings of International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)*, volume 2381 of *Lecture Notes in Computer Science*. Springer, July 2002.

67. Bing Li, Chao Wang, and Fabio Somenzi. A Satisfiability-based Approach to Abstraction Refinement in Model Checking. In *Proceedings of the First International Workshop on Bounded Model Checking (BMC)*, volume 89 of *Electronic Notes in Theoretical Computer Science*. Elsevier, July 2003.

68. Feng Lu, Li-C. Wang, Kwang-T. Cheng, John Moondanos, and Ziyad Hanna. A Signal Correlation Guided ATPG Solver And Its Applications For Solving Difficult Industrial Cases. In *Proceedings of the 40th Design Automation Conference (DAC)*, pages 436–441, June 2003.

69. Feng Lu, Li-C. Wang, Kwang-Ting Cheng, and Ric Chung-Yang Huang. A Circuit SAT Solver with Signal Correlation Guided Learning. In *Proceedings of the Design Automation and Test in Europe (DATE)*, pages 892–897, March 2003.

70. João P. Marques-Silva. The Impact of Branching Heuristics in Propositional Satisfiability Algorithms. In *Proceedings of the 9th Portuguese Conference on Artificial Intelligence (EPIA)*, September 1999.

71. João P. Marques-Silva and Karem A. Sakallah. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999.

72. Kenneth L. McMillan. *Symbolic Model Checking: An approach to the State Explosion Problem*. Kluwer Academic Publishers, 1993.

73. Kenneth L. McMillan. Applying SAT Methods in Unbounded Symbolic Model Checking. In Ed Brinksma and Kim G. Larsen, editors, *Proceedings of the 14th International Conference on Computer-Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 250–264. Springer, July 2002.

74. Kenneth L. McMillan. Interpolation and SAT-based Model Checking. In Jr. Warren A. Hunt and Fabio Somenzi, editors, *Proceedings of the 15th Conference on Computer-Aided Verification (CAV)*, volume 2725 of *Lecture Notes in Computer Science*, pages 1–13. Springer, July 2003.

75. Kenneth L. McMillan and Nina Amla. Automatic abstraction without counterexamples. In Hubert Garavel and John Hatcliff, editors, *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2619 of *Lecture Notes in Computer Science*, pages 2–17. Springer, April 2003.

76. Maher Mneimneh and Karem Sakallah. SAT-Based Sequential Depth Computation. In *Proceedings of the First International Workshop on Constraints in Formal Verification*, September 2002.

77. Matthew H. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC)*, pages 530–535, June 2001.

78. Ganapathy Parthasarthy, Chung-Yang Huang, and Kwang-Ting Cheng. An Analysis of ATPG and SAT algorithms for Formal Verification. In *Proceedings of the 6th International High-Level Design Validation and Test Workshop (HLDVT)*, pages 177–182, November 2001.

79. Robert P. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*. Princeton University Press, 1995.

80. David Plaisted, Armin Biere, and Yunshan Zhu. A Satisfiability Procedure for Quantified Boolean Formulae. *Discrete Applied Mathematics*, 130(2):291–328, August 2003.

81. David Plaisted and Steven Greenbaum. A Structure-preserving Clause Form Translation. *Journal of Symbolic Computation*, 2(3):293–304, September 1986.

82. Jussi Rintanen. Partial implicit unfolding in the davis-putnam procedure for quantified boolean formulae. In *International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, 2001.

83. W. J. Savitch. Relational between nondeterministic and deterministic tape complexity. *Journal of Computer and System Sciences*, 4:177–192, 1970.

84. Viktor Schuppan and Armin Biere. Efficient reduction of finite state model checking to reachability analysis. *Software Tools for Technology Transfer (STTT)*, 5(1-2):185–204, March 2004.

85. Bart Selman, Henry A. Kautz, and Bram Cohen. Noise Strategies for Improving Local Search. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI)*, pages 337–343, July 1994.

86. Bart Selman, Hector J. Levesque, and David Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI)*, pages 440–446, July 1992.

87. Sanjit A. Seshia, Shuvendu K. Lahiri, and Randal E. Bryant. A Hybrid SAT-Based Decision Procedure for Separation Logic with Uninterpreted Functions. In *Proceedings of the 40th Design Automation Conference (DAC)*, pages 425–430, June 2003.

88. Ohad Shacham and Emmanuel Zarpas. Tuning the VSIDS decision heuristic for bounded model checking. In *Proceedings of the 4th International Workshop on Microprocessor Test and Verification (MTV)*, pages 75–79, May 2003.

89. Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking safety properties using induction and a SAT-solver. In Warren A. Hunt Jr. and Steven D. Johnson, editors, *Proceedings of the 3rd International Conference on Formal Methods in Computer Aided Design (FMCAD)*, volume 1954 of *Lecture Notes in Computer Science*, pages 108–125. Springer, November 2000.

90. Mary Sheeran and Gunnar Stålmarck. A tutorial on Stålmarck's proof procedure for propositional logic. *Formal Methods in System Design*, 16(1):23–58, January 2000.

91. Shuo Sheng, Koichiro Takayama, and Michael S. Hsiao. Effective Static Property Checking Using Simulation-Based

ATPG. In *Proceedings of the 39th Design Automation Conference (DAC)*, pages 813–818, June 2002.

92. Ofer Shtrichman. Sharing Information Between Instances of Propositional Satisfiability (SAT) Problems, Jan 2000. US patent (Disclosure number: IL8-2000-0070).

93. L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proceedings of the 5th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 1–9, 1973.

94. Dominik Stoffel and Wolfgang Kunz. Record & Play: A Structural Fixed Point Iteration for Sequential Circuit Verification. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, pages 394 – 399, November 1997.

95. Ofer Strichman. Tuning SAT Checkers for Bounded Model Checking. In E. Allen Emerson and A. Prasad Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification (CAV)*, volume 1855 of *Lecture Notes in Computer Science*, pages 480–494. Springer, July 2000.

96. Ofer Strichman. Pruning Techniques for the SAT-based Bounded Model Checking Problem. In Tiziana Margaria and Thomas F. Melham, editors, *Proceedings of the 11th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME)*, volume 2144 of *Lecture Notes in Computer Science*, pages 58–70. Springer, September 2001.

97. Ofer Strichman. On Solving Presburger and Linear Arithmetic with SAT. In Mark Aagaard and John W. O'Leary, editors, *Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, volume 2517 of *Lecture Notes in Computer Science*, pages 160–170. Springer, November 2002.

98. G. S. Tseitin. On the Complexity of Derivation in Propositional Calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part II*, volume 8 of *Seminars in Mathematics*, pages 234–259, Leningrad, 1968. V.A. Steklov Mathematical Institute. English Translation: Consultants Bureau, New York, 1970, pages 115 – 125.

99. C. A. J. van Eijk. Sequential Equivalence Checking without State Space Traversal. In *Proceedings of the Design Automation and Test in Europe (DATE)*, pages 618–623, February 1998.

100. Miroslav N. Velev and Randal E. Bryant. Effective Use of Boolean Satisfiability Procedures in the Formal Verification of Superscalar and VLIW Microprocessors. *Journal of Symbolic Computation (JSC)*, 35(2):73–106, February 2003.

101. Chao Wang, Bing Li, HoonSang Jin, Gary D. Hachtel, and Fabio Somenzi. Improving Ariadne's Bundle by Following Multiple Threads in Abstraction Refinement. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, pages 408–415, November 2003.

102. Jesse P. Whittemore, Joonyoung Kim, and Karem A. Sakallah. SATIRE: A New Incremental Satisfiability Engine. In *Proceedings of the 38th Design Automation Conference (DAC)*, pages 542–545, June 2001.

103. Poul F. Williams, Armin Biere, Edmund M. Clarke, and Anubhav Gupta. Combining Decision Diagrams and SAT Procedures for Efficient Symbolic Model Checking. In E. Allen Emerson and A. Prasad Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification (CAV)*, volume 1855 of *Lecture Notes in Computer Science*, pages 124–138. Springer, July 2000.

104. Chia-Chih Yen, Kuang-Chien Chen, and Jing-Yang Jou. A Practical Approach to Cycle Bound Estimation for Property Checking. In *Proceedings of 11th International Workshop on Logic & Synthesis (IWLS)*, pages 149–154, June 2002.

105. Hantao Zhang. SATO: An Efficient Propositional Prover. In William McCune, editor, *Proceedings of the 14th International Conference on Automated Deduction (CADE)*, volume 1249 of *Lecture Notes in Computer Science*, pages 272–275. Springer, July 1997.

106. Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz, and Sharad Malik. Efficient Conflict Driven Learning in a Boolean Satisfiability Solver. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, pages 279–285, November 2001.

107. Lintao Zhang and Sharad Malik. The Quest for Efficient Boolean Satisfiability Solvers. In Ed Brinksma and Kim G. Larsen, editors, *Proceedings of the 14th International Conference on Computer Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, pages 17–36. Springer, July 2002.

108. Lintao Zhang and Sharad Malik. Towards Symmetric Treatment of Conflicts And Satisfaction in Quantified Boolean Satisfiability Solvers. In Pascal Van Hentenryck, editor, *Proceedings of 8th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 2470 of *Lecture Notes in Computer Science*, pages 200–215. Springer, 2002.

109. Lintao Zhang and Sharad Malik. Validating SAT Solvers using an Independent Resolution-based Checker: Practical Implementations and Other Applications. In *Proceedings of the Design Automation and Test in Europe (DATE)*, pages 880–885, March 2003.