

A survey of recent advances in SAT-based formal verification

Mukul R. Prasad¹, Armin Biere², Aarti Gupta³

¹Fujitsu Laboratories of America, Sunnyvale, CA, USA

²Johannes Kepler University, Linz, Austria

e-mail: biere@inf.ethz.ch

³NEC Laboratories of America, Princeton, NJ, USA

Published online: 25 January 2005 – © Springer-Verlag 2005

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.

Keywords: 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 toward 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 that, due to lack of space, cannot 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 Sect. 5. Another recent development has been the use of *quantified Boolean formula (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 Sect. 6. We conclude the paper in Sect. 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, summarized in a recent survey [107], in SAT solver technology over the years. 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 DPLL. They are strong 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 Fig. 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

```

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;

```

Fig. 1. DPLL-based SAT solver

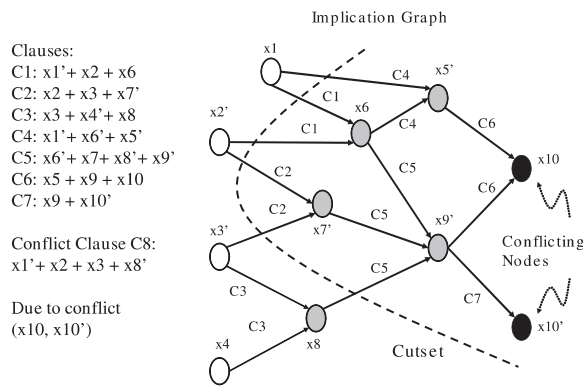


Fig. 2. Conflict analysis using an implication graph

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 nonchronological backtracking, i.e., up to the closest decision that 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 Fig. 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 that 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 that comprise the DPLL-based SAT solver – decision 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 nonfalse 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 nonfalse 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 nonfalse, 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 nonfalse 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* [75, 109], such that these clauses are sufficient for implying unsatisfiability. The use of such techniques in verification applications are described in more detail in Sect. 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 corelated signals in the given circuit. Then the inequivalence of the corelated 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 could 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 textbook [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 that 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)*, that usually require dedicated algorithms. In this paper we focus on *simple safety properties*, also often called invariants, written in CTL as $\mathbf{AG}p$. This formula specifies that for all execution 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, cannot 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 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).

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

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 Fig. 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 \mathbf{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 [17, 54, 57, 58]. 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 $\mathbf{EXP} \wedge \mathbf{EX}q$ or $\mathbf{AG} \mathbf{EX}p$, can be handled with Img computation only. In certain cases backward traversal is better, for instance, if property p is an inductive invariant, as defined in Sect. 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 \mathbf{AX} (also known as the weakest precondition 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 , independently of the values at the primary inputs. A backward model checking algorithm for simple safety properties, based on greatest fixpoint calculation and on the \mathbf{AX} operator, can be formulated as in Fig. 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 $\text{model-check}_{\text{forward}}^{\mu}$ 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 fixpoints in Sect. 4.1.2.

```

model-checkforwardμ( $I, T, B$ )
 $S_C = \emptyset; S_N = I;$ 
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”;
   $S_N = S_C \cup \text{Img}(S_C);$ 
done;
return “no bad state reachable”;

```

Fig. 3. Forward least fixpoint algorithm for safety properties

```

model-checkbackwardν( $I, T, G$ )
 $S_C = \text{“all states”}; S_P = G;$ 
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”;

```

Fig. 4. Backward greatest fixpoint algorithm for safety properties

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 counterexamples 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:

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

where I is the characteristic function for the set of initial states of M and T_i is the characteristic function of the transition relation of M for time step i . Thus, the formula $I \wedge \bigwedge_{i=0}^{k-1} T_i$ 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 Eq. (2) reduces to:

$$\varphi^k = I \wedge \bigwedge_{i=0}^{k-1} T_i \wedge \left(\bigvee_{i=0}^k \neg P_i \right) \quad (3)$$

where P_i is the expression p in time step i . Thus, this formula can be satisfied if and only if for some i ($i \leq k$) 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

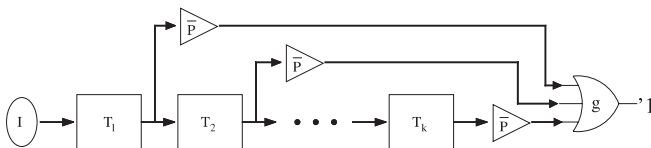


Fig. 5. Bounded model checking

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, in the hope that the resulting SAT problem will be 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 Fig. 5, the BCOI reduction would extract the transitive fanin cone of gate g and construct the BMC-CNF only from this subcircuit. 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 (Fig. 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 rewriting 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 (Fig. 5) to 0 and then rehash 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 heuristics such as the *DLIS* heuristic [70]. Strichman [95] proposed a static variable ordering scheme specifically targeted for BMC problems that 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 fur-

ther 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 a problem is repeatedly solved 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 relearn them. Specifically, any conflict clause derived *exclusively* from the subformula $\Phi_k = I \wedge \bigwedge_{i=0}^{k-1} T_i$ can be reused (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 satisfiability* techniques, with applications in BMC [92] and other general classes of SAT problems [61, 102]. Generally, this technique has been found to offer speedups of up to $2\times$ or more with negligible overhead.

A related technique called *constraints replication* [95] first identifies conflict clauses c , derived from the subformula $\bigwedge_{i=0}^{k-1} T_i$ 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., subcircuits) in the unrolled design for BMC. The authors 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

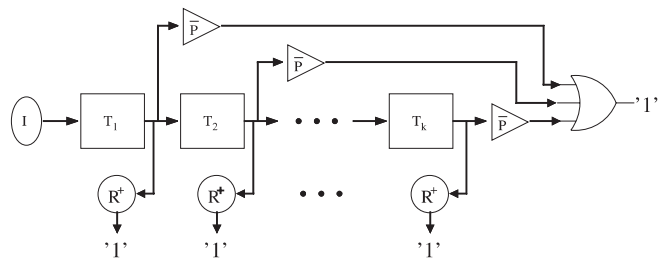


Fig. 6. Improving BMC using reachability overapproximation

BDD-based approximate reachability analysis to *quickly* compute a *succinct* and coarse overapproximation, 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 Fig. 6. The BDDs are converted to CNF constraints that are conjoined with the BMC formulation of Eq. (2). This technique does indeed have an overhead and is therefore useful primarily for larger, more difficult BMC problems. In such cases speedups of up to an order of magnitude have been observed.

3.4 Methodology improvements to BMC

Although BMC is by intent an incomplete, bug-finding method rather than a complete verification method, a given property can be certified to be true if no counterexamples are found through BMC, up to the *sequential depth* of the circuit [15]. The sequential depth of a circuit is the 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 underapproximation or an overapproximation 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 (QBFs) (see also Sect. 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 overapproximating 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 runtimes 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 five control units from the PowerPC 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 subsystem 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, the researchers 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 counterexamples. They also find that explicit-state BMC based on random simulation can give performance comparable 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 counterexamples 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 is 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 Sect. 6.

4.1 SAT-based state space search

Due to the success of SAT solvers in BMC, there has been growing interest in their use for *unbounded* model checking. Here, the crucial nontrivial 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 Sect. 2.3:

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

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/preimage operations (shown in Eq. (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. Abdulla et al. use *reduced Boolean circuits (RBCs)* to represent the Boolean formulas, which

can be exponentially more succinct than BDDs but are semicanonical. 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 the 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 that 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 . The authors 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 (Fig. 4, Sect. 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 rearrange the implication graph (described in Sect. 2.2) when a solution (i.e., a satisfying assignment) is found by the SAT solver. This rearrangement can be viewed as a *cube en-*

largement 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 overapproximation of the set of states reachable from the initial state in one step (or any fixed number of steps). This provides an approximate image operator, which can be used iteratively to compute an overapproximation of the set of reachable states, i.e., till a fixpoint is obtained. If at any point the overapproximate 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 overapproximation 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 overapproximate 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 cannot 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 with cubewise 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 with 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/preimage state sets, unlike the interpolant-based technique (described above), which computes approximate state sets. The technique has been used in SAT-based unbounded symbolic model checking to handle many difficult industry examples that 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 counterexamples 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.

$$\begin{aligned}\phi_{base} &= I \wedge \neg P_0 \\ \phi_{induc} &= P_k \wedge T(k, k+1) \wedge (\neg P_{k+1})\end{aligned}\quad (5)$$

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 Eq. (5) become:

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

Essentially, induction with depth corresponds to strengthening the induction hypothesis by imposing the original induction hypothesis (P_k in ϕ_{induc} , Eq. (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 Sect. 3.3. As in [23], BDD-based

techniques are used to *efficiently* compute a *succinct* overapproximation 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 overapproximation for the reachable states in the context of sequential equivalence checking [18, 94, 99].

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 that 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 tradeoff 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* [75,109], 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 [50,75]. 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 approaches – 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. The authors 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, were crucial for successful verification of large industry designs.

In practice, refinement-based approaches that 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 that 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 what follows) are designed to search for input sequences to the given circuit that can test for the presence of a certain fault. A subtask in this process involves performing a search on the space of input sequences of the sequential machine for an 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 tristate buses and high-impedance logic values. Further, unlike SAT-based BMC, 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 Fig. 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 counterexample for the specified property. A key contribution of this work was to recognize that most sequential designs have *synchronizing sequences* that 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 $\mathbf{AG\ EF}p$ can be reduced to verifying $\mathbf{EF}_{s_0}p$. 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 counterexample within n time steps of the initial state. The authors report impressive speedups 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-based 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 Sect. 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 that 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 formulas* (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 formulas (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

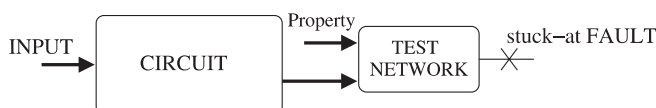


Fig. 7. Sequential ATPG for model checking

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 Fig. 3 in Sect. 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 does an alternation of quantifiers occur.

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] (Sect. 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 Sect. 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:

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

Let the model consist 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 Eq. (7) compared to standard image computation as in Eq. (1). Additionally, the approximations to the transitive closure of the transition relation cannot 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 subformulas. 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:

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

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 no longer propositional since it involves $2n$ alternations of quantifiers, but it is linear in the number of state bits n . It still remains 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 [94, 99], directly based on SAT [73, 74], or based on the 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 Sect. 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 Fig. 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 subproblems, assigning $x = 0$ and $x = 1$, return *satisfiable*.

Recently, several groups [43, 66, 108] independently applied techniques from the SAT domain to QBF, such as the conflict-driven learning and conflict-driven backtracking techniques discussed in Sect. 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 and 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 and 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 what follows 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 one technology significantly outperforms the other and vice versa. Further, SAT-based techniques have been found to be less sensitive to 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 (up to 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 that 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 Sect. 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 was pointed out by the original proponents of SAT-based BMC [26], the technique can be used to formally verify properties by performing the BMC check up to the sequential circuit depth (or some overapproximation thereof). Recently, there have been several attempts at the problem of *computing the sequential depth* of a given system. Tight overapproximations 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

¹ <http://www.qbflib.org>

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 at *implicit time frame unrolling* has been made in the SATORI solver [59] that 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 as for tool developers seeking to integrate this technology into the next generation of verification tools in industrial practice.

References

1. Abdulla PA, Bjesse P, Eén N (2000) Symbolic reachability analysis based on SAT-solvers. In: Graf S, Schwartzbach M (eds) Proceedings of the 6th international conference on tools and algorithms for the construction and analysis of systems (TACAS), March 2000. Lecture notes in computer science, vol 1785. Springer, Berlin Heidelberg New York, pp 411–425
2. Abraham JA, Vedula VM, Saab DG (2002) Verifying properties using sequential ATPG. In: Proceedings of the International Test Conference (ITC), October 2002, pp 194–202
3. Alur R (1999) Timed automata. In: Halbwachs N, Peled D (eds) Proceedings of the 11th international conference on computer-aided verification (CAV), July 1999. Lecture notes in computer science, vol 1633. Springer, Berlin Heidelberg New York, pp 8–22
4. Amla N, Kurshan R, McMillan K, Medel R (2003) Experimental analysis of different techniques for bounded model checking. In: Garavel H, Hatcliff J (eds) Proceedings of the 9th international conference on tools and algorithms for the construction and analysis of systems (TACAS), April 2003. Lecture notes in computer science, vol 2619. Springer, Berlin Heidelberg New York, pp 34–48
5. Andersen HR, Hulgaard H (2002) Boolean expression diagrams. *Inf Comput* 179(2):194–212
6. Ayari A, Basin D (2000) Bounded model construction for monadic second-order logics. In: Emerson EA, Sistla AP (eds) Proceedings of the 12th international conference on computer-aided verification (CAV), July 2000. Lecture notes in computer science, vol 1855. Springer, Berlin Heidelberg New York, pp 99–113
7. Ayari A, Basin D (2002) QUBOS: Deciding quantified Boolean logic using propositional satisfiability solvers. In: Aagam M, O’Leary JW (eds) Proceedings of the 4th international conference on formal methods in computer-aided design (FMCAD). Lecture notes in computer science, vol 2517. Springer, Berlin Heidelberg New York, pp 187–201
8. Ball T, Rajamani SK (2002) The SLAM project: debugging system soft-ware via static analysis. In: Proceedings of the 29th SIGPLAN-SIGACT symposium on principles of programming languages (POPL) January 2002. ACM Press, New York, pp 1–3
9. Barrett CW, Dill DL, Stump A (2002) Checking satisfiability of first-order formulas by incremental translation to SAT. In: Brinksma E, Larsen KG (eds) Proceedings of the 14th international conference on computer-aided verification (CAV), July 2002. Lecture notes in computer science, vol 2404. Springer, Berlin Heidelberg New York, pp 236–249
10. Baumgartner J, Kuehlmann A, Abraham JA (2002) Property Checking via Structural Analysis. In: Brinksma E, Larsen KG (eds) Proceedings of the 14th international conference on computer-aided verification (CAV), July 2002. Lecture notes in computer science, vol 2404. Springer, Berlin Heidelberg New York, pp 151–165
11. Bayardo RJ, Schrag RC (1997) Using CSP look-back techniques to solve real-world SAT instances. In: Proceedings of the national conference on artificial intelligence (AAAI), July 1997, pp 203–208
12. Le Berre D, Simon L, Tachella A (2004) Challenges in the QBF arena: the SAT’03 evaluation of QBF solvers. In: Giunchiglia E, Tacchella A (eds) Proceedings of the 6th international conference on theory and applications of satisfiability testing (SAT), May 2004. Lecture notes in computer science, vol 2919. Springer, Berlin Heidelberg New York, pp 468–485
13. Biere A (2004) Resolve and expand. In: Proceedings of the 7th international conference on theory and applications of satisfiability testing (SAT), May 2004
14. Biere A, Cimatti A, Clarke EM, Fujita M, Zhu Y (1999) Symbolic model checking using SAT procedures instead of BDDs. In: Proceedings of the 36th conference on design automation (DAC), June 1999, pp 317–320
15. Biere A, Cimatti A, Clarke EM, Zhu Y (1999) Symbolic model checking without BDDs. In: Cleaveland R (ed) Proceedings of the 5th international conference on tools and algorithms for the construction and analysis of systems (TACAS), March 1999. Lecture notes in computer science, vol 1579. Springer, Berlin Heidelberg New York, pp 193–207
16. Biere A, Clarke E, Raimi R, Zhu Y (1999) Verifying safety properties of a PowerPC microprocessor using symbolic model checking without BDDs. In: Halbwachs N, Peled D (eds) Proceedings of the 11th international conference on computer-aided verification (CAV), July 1999. Lecture notes in computer science, vol 1633. Springer, Berlin Heidelberg New York, pp 60–71
17. Biere A, Clarke EM, Zhu Y (1999) Multiple state and single state tableaux for combining local and global model checking. In: Olderog E-R, Steffen B (eds) Correct system design, recent insight and advances. Lecture notes in computer science, vol 1710. Springer, Berlin Heidelberg New York, pp 163–179
18. Bjesse P, Claessen K (2000) SAT-based verification without state space traversal. In: Hunt Jr WA, Johnson SD (eds) Proceedings of the 3rd international conference on formal methods in computer-aided design (FMCAD), November 2000. Lecture notes in computer science, vol 1954. Springer, Berlin Heidelberg New York, pp 372–389
19. Bjesse P, Leonard T, Mokkedem A (2001) Finding bugs in an alpha microprocessor using satisfiability solvers. In: Berry G, Comon H, Finkel A (eds) Proceedings of the 13th international conference on computer-aided verification (CAV), July 2001. Lecture notes in computer science, vol 2102. Springer, Berlin Heidelberg New York, pp 454–464
20. Boppana V, Rajan SP, Takayama K, Fujita M (1999) Model checking based on sequential ATPG. In: Halbwachs N, Peled D (eds) Proceedings of the 11th international conference on computer-aided verification (CAV), July 1999. Lecture notes in computer science, vol 1633. Springer, Berlin Heidelberg New York, pp 418–430
21. Bryant RE (1986) Graph based algorithms for Boolean function manipulation. *IEEE Trans Comput* C(35):677–691
22. Burch JR, Clarke EM, Long DE, McMillan KL, Dill DL (1994) Symbolic model checking for sequential circuit verification. *IEEE Trans Comput Aided Des Integ Circuits Syst* 13(4):401–424

23. Cabodi G, Nocco S, Quer S (2003) Improving SAT-based bounded model checking by means of BDD-based approximate traversals. In: Proceedings of Design Automation and Test in Europe (DATE), March 2003, pp 898–903
24. Cadoli M, Giovanardi A, Schaerf M (1998) An algorithm to evaluate quantified Boolean formulae. In: Proceedings of the 15th national conference on artificial intelligence (AAAI), July 1998, pp 262–267
25. Chauhan P, Clarke EM, Kukula J, Sapra S, Veith H, Wang D (2002) Automated abstraction refinement for model checking large state spaces using SAT based conflict analysis. In: Aagaard M, O’Leary JW (eds) Proceedings of the 4th international conference on formal methods in computer-aided design (FMCAD), November 2002. Lecture notes in computer science, vol 2517. Springer, Berlin Heidelberg New York, pp 33–51
26. Clarke E, Biere A, Raimi R, Zhu Y (2001) Bounded model checking using satisfiability solving. *Formal Methods Syst Des* 19(1):7–34
27. Clarke EM, Emerson EA (1982) Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Kozen D (ed) Proceedings of the workshop on logic of programs. Lecture notes in computer science, vol 131. Springer, Berlin Heidelberg New York, pp 52–71
28. Clarke EM, Grumberg O, Peled DA (2000) Model checking. MIT Press, Cambridge, MA
29. Clarke EM, Gupta A, Kukula J, Strichman O (2002) SAT-based abstraction refinement using ILP and machine learning techniques. In: Brinksma E, Larsen KG (eds) Proceedings of the 14th international conference on computer-aided verification (CAV), July 2002. Lecture notes in computer science, vol 2404. Springer, Berlin Heidelberg New York, pp 265–279
30. Clarke EM, Schlingloff B-H (2001) Model checking. In: Robinson JA, Voronkov A (eds) Handbook of automated reasoning, vol 2. Elsevier/MIT Press, Amsterdam/Cambridge, MA, pp 1635–1790
31. Copti F, Fix L, Fraer R, Giunchiglia E, Kamhi G, Tacchella A, Vardi MY (2001) Benefits of bounded model checking in an industrial setting. In: Berry G, Comon H, Finkel A (eds) Proceedings of the 13th international conference on computer-aided verification (CAV), July 2001. Lecture notes in computer science, vol 2102. Springer, Berlin Heidelberg New York, pp 436–453
32. Davis M, Logemann G, Loveland D (1962) A machine program for theorem-proving. *Commun ACM* 5(7):394–397
33. Davis M, Putnam H (1960) A computing procedure for quantification theory. *J ACM* 7(3):201–215
34. Donini FM, Liberatore P, Massacci F, Schaerf M (2002) Solving QBF with SMV. In: Proceedings of the 8th international conference on principles of knowledge representation and reasoning (KR), pp 578–589
35. Eén N, Sörensson N (2003) Temporal induction by incremental SAT solving. In: Strichman O, Biere A (eds) Proceedings of the 1st international workshop on bounded model checking (BMC), July 2003. Electronic notes in theoretical computer science, vol 89. Elsevier, Amsterdam
36. Emerson EA (1990) Temporal and modal logic, vol B. MIT Press, Cambridge, MA, pp 995–1072
37. Fallah F (2002) Binary time-frame expansion. In: Proceedings of the international conference on computer-aided design (ICCAD), November 2002, pp 458–464
38. Fujiwara H, Shimono T (1983) On the acceleration of test generation algorithms. *IEEE Trans Comput C-32*:1137–1144
39. Ganai MK, Aziz A (2002) Improved SAT-based bounded reachability analysis. In: Proceedings of the 15th international conference on VLSI design (VLSID), January 2002, pp 729–734
40. Ganai MK, Gupta A, Ashar P (2004) Efficient SAT-based unbounded symbolic model checking using circuit cofactoring. In: Proceedings of the international conference on computer-aided design (ICCAD), November 2004
41. Ganai MK, Zhang L, Ashar P, Gupta A (2002) Combining strengths of circuit-based and CNF-based algorithms for a high performance SAT solver. In: Proceedings of the 39th conference on design automation (DAC), June 2002, pp 747–750
42. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco
43. Giunchiglia E, Narizzano M, Tacchella A (2002) Learning for quantified Boolean logic satisfiability. In: Proceedings of the 18th national conference on artificial intelligence (AAAI), July 2002, pp 649–654
44. Goel P (1981) An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Trans Comput C-30*:215–222
45. Goldberg E, Novikov Y (2002) BerkMin: a fast and robust SAT-solver. In: Proceedings of Design Automation and Test in Europe (DATE), March 2002, pp 142–149
46. Goldberg E, Novikov Y (2003) Verification of proofs of unsatisfiability for CNF formulas. In: Proceedings of Design Automation and Test in Europe (DATE), March 2003, pp 886–891
47. Goldberg E, Prasad MR, Brayton RK (2001) Using SAT for combinational equivalence checking. In: Proceedings of Design Automation and Test in Europe (DATE), March 2001, pp 114–121
48. Gupta A, Ganai M, Wang C, Yang Z, Ashar P (2003) Abstraction and BDDs complement SAT-based BMC in *DiVer*. In: Hunt Jr WA, Somenzi F (eds) Proceedings of the 15th international conference on computer-aided verification (CAV), July 2003. Lecture notes in computer science, vol 2725. Springer, Berlin Heidelberg New York, pp 206–209
49. Gupta A, Ganai M, Wang C, Yang Z, Ashar P (2003) Learning from BDDs in SAT-based bounded model checking. In: Proceedings of the 40th conference on design automation (DAC), June 2003, pp 824–829
50. Gupta A, Ganai M, Yang Z, Ashar P (2003) Iterative abstraction using SAT-based BMC with proof analysis. In: Proceedings of the international conference on computer-aided design (ICCAD), November 2003, pp 416–423
51. Gupta A, Gupta A, Yang Z, Ashar P (2001) Dynamic detection and removal of inactive clauses in SAT with application in image computation. In: Proceedings of the 38th conference on design automation, June 2001, pp 536–541
52. Gupta A, Yang Z, Ashar P, Gupta A (2000) SAT based state reachability analysis and model checking. In: Hunt WA, Johnson SD (eds) Proceedings of the 3rd international conference on formal methods in computer-aided design (FMCAD), November 2000. Lecture notes in computer science, vol 1954. Springer, Berlin Heidelberg New York, pp 354–371
53. Gupta A, Yang Z, Ashar P, Zhang L, Malik S (2001) Partition-based decision heuristics for image computation using SAT and BDDs. In: Proceedings of the international conference on computer-aided design (ICCAD), November 2001, pp 286–292
54. Henzinger TA, Kupferman O, Qadeer S (1998) From pre-historic to post-modern symbolic model checking. In: Hu AJ, Vardi MY (eds) Proceedings of the 10th international conference on computer-aided verification (CAV), July 1998. Lecture notes in computer science, vol 1427. Springer, Berlin Heidelberg New York, pp 195–206
55. Holzmann GJ (1991) Design and validation of computer protocols. Prentice Hall, Upper Saddle River, NJ
56. Huan C-Y, Cheng K-T (2001) Using word-level ATPG and modular arithmetic constraint-solving techniques for assertion property checking. *IEEE Trans Comput Aided Des* 20(3): 381–391
57. Iwashita H, Nakata T (1997) Forward model checking techniques oriented to buggy designs. In: Proceedings of the international conference on computer-aided design (ICCAD), November 1997, pp 400–404
58. Iwashita H, Nakata T, Hirose F (1996) CTL model checking based on forward state traversal. In: Proceedings of the international conference on computer-aided design (ICCAD), November 1996, pp 82–87
59. Iyer MK, Parthasarathy G, Cheng K-T (2003) SATORI – A fast sequential SAT engine for circuits. In: Proceedings of the international conference on computer-aided design (ICCAD), November 2003, pp 320–325
60. Jackson D, Vaziri M (2000) Finding bugs with a constraint solver. In: Proceedings of the international symposium on software testing and analysis (ISSTA), August 2000, pp 14–25

61. Kim J, Whittimore J, Sakallah K (2000) On solving stack-based incremental satisfiability problems. In: Proceedings of the international conference on computer design (ICCD), October 2000, pp 379–382
62. Kleine Büning H, Karpinski M, Flögel A (1995) Resolution for quantified boolean formulas. *Inf Comput* 117(1):12–18
63. Kleine Büning H, Lettmann T (1999) Propositional logic: deduction and algorithms, Cambridge tracts in theoretical computer science, vol 48. Cambridge University Press, Cambridge, UK. ISBN-0-521-63017-7
64. Kuehlmann A, Paruthi V, Krohm F, Ganai MK (2002) Robust Boolean reasoning for equivalence checking and functional property verification. *IEEE Trans Comput Aided Des Integ Circuits Syst* 21(12):1377–1394
65. Larrabee T (1992) Test pattern generation using Boolean satisfiability. *IEEE Trans Comput Aided Des Integ Circuits Syst* 11(1):4–15
66. Letz R (2002) Lemma and model caching in decision procedures for quantified Boolean formulas. In: Egly U, Fermüller CG (eds) Proceedings of the international conference on automated reasoning with analytic tableaux and related methods (TABLEAUX), July 2002. Lecture notes in computer science, vol 2381. Springer, Berlin Heidelberg New York
67. Li B, Wang C, Somenzi F (2003) A satisfiability-based approach to abstraction refinement in model checking. In: Proceedings of the 1st international workshop on bounded model checking (BMC), July 2003. Electronic notes in theoretical computer science, vol 89. Elsevier, Amsterdam
68. Lu F, Wang L-C, Cheng K-T, Moondanos J, Hanna Z (2003) A signal correlation guided ATPG solver and its applications for solving difficult industrial cases. In: Proceedings of the 40th conference on design automation (DAC), June 2003, pp 436–441
69. Lu F, Wang L-C, Cheng K-T, Huang RC-Y (2003) A circuit SAT solver with signal correlation guided learning. In: Proceedings of Design Automation and Test in Europe (DATE), March 2003, pp 892–897
70. Marques-Silva JP (1999) The impact of branching heuristics in propositional satisfiability algorithms. In: Proceedings of the 9th Portuguese conference on artificial intelligence (EPIA), September 1999
71. Marques-Silva JP, Sakallah KA (1999) GRASP: A search algorithm for propositional satisfiability. *IEEE Trans Comput* 48(5):506–521
72. McMillan KL (1993) Symbolic model checking: an approach to the state explosion problem. Kluwer, Dordrecht
73. McMillan KL (2002) Applying SAT methods in unbounded symbolic model checking. In: Brinksma E, Larsen KG (eds) Proceedings of the 14th international conference on computer-aided verification, July 2002. Lecture notes in computer science, vol 2404. Springer, Berlin Heidelberg New York, pp 250–264
74. McMillan KL (2003) Interpolation and SAT-based model checking. In: Hunt Jr WA, Somenzi F (eds) Proceedings of the 15th conference on computer-aided verification (CAV), July 2003. Lecture notes in computer science, vol 2725. Springer, Berlin Heidelberg New York, pp 1–13
75. McMillan KL, Amla N (2003) Automatic abstraction without counterexamples. In: Garavel H, Hatcliff J (eds) Proceedings of the international conference on tools and algorithms for the construction and analysis of systems (TACAS), April 2003. Lecture notes in computer science, vol 2619. Springer, Berlin Heidelberg New York, pp 2–17
76. Mneimneh M, Sakallah K (2002) SAT-based sequential depth computation. In: Proceedings of the 1st international workshop on constraints in formal verification, September 2002
77. Moskewicz MH, Madigan CF, Zhao Y, Zhang L, Malik S (2001) Chaff: engineering an efficient SAT solver. In: Proceedings of the 38th conference on design automation (DAC), June 2001, pp 530–535
78. Parthasarthy G, Huang C-Y, Cheng K-T (2001) An analysis of ATPG and SAT algorithms for formal verification. In: Proceedings of the 6th international workshop on high-level design validation and test (HLDVT), November 2001, pp 177–182
79. Kurshan RP (1995) Computer-aided verification of coordinating processes: the automata-theoretic approach. Princeton University Press, Princeton, NJ
80. Plaisted D, Biere A, Zhu Y (2003) A satisfiability procedure for quantified Boolean formulae. *Discrete Appl Math* 130(2):291–328
81. Plaisted D, Greenbaum S (1986) A structure-preserving clause form translation. *J Symbol Comput* 2(3):293–304
82. Rintanen J (2001) Partial implicit unfolding in the Davis-Putnam procedure for quantified boolean formulae. In: International conference on logic for programming, artificial intelligence and reasoning (LPAR)
83. Savitch WJ (1970) Relational between nondeterministic and deterministic tape complexity. *J Comput Syst Sci* 4:177–192
84. Schuppan V, Biere A (2004) Efficient reduction of finite state model checking to reachability analysis. *Int J Softw Tools Technol Transfer* 5(1–2):185–204
85. Selman B, Kautz HA, Cohen B (1994) Noise strategies for improving local search. In: Proceedings of the 12th national conference on artificial intelligence (AAAI), July 1994, pp 337–343
86. Selman B, Levesque HJ, Mitchell D (1992) A new method for solving hard satisfiability problems. In: Proceedings of the 10th national conference on artificial intelligence (AAAI), July 1992, pp 440–446
87. Seshia SA, Lahiri SK, Bryant RE (2003) A hybrid SAT-based decision procedure for separation logic with uninterpreted functions. In: Proceedings of the 40th conference on design automation (DAC), June 2003, pp 425–430
88. Shacham O, Zarpas E (2003) Tuning the VSIDS decision heuristic for bounded model checking. In: Proceedings of the 4th international workshop on microprocessor test and verification (MTV), May 2003, pp 75–79
89. Sheeran M, Singh S, Stålmarck G (2000) Checking safety properties using induction, a SAT-solver. In: Hunt Jr WA, Johnson SD (eds) Proceedings of the 3rd international conference on formal methods in computer-aided design (FMCAD), November 2000. Lecture notes in computer science, vol 1954. Springer, Berlin Heidelberg New York, pp 108–125
90. Sheeran M, Stålmarck G (2000) A tutorial on Stålmarck’s proof procedure for propositional logic. *Formal Methods Syst Des* 16(1):23–58
91. Sheng S, Takayama K, Hsiao MS (2002) Effective static property checking using simulation-based ATPG. In: Proceedings of the 39th conference on design automation (DAC), June 2002, pp 813–818
92. Shtrichman O (2000) Sharing information between instances of propositional satisfiability (SAT) problems, January 2000. US patent (Disclosure no.: IL8-2000-0070)
93. Stockmeyer LJ, Meyer AR (1973) Word problems requiring exponential time. In: Proceedings of the 5th annual ACM symposium on the theory of computing (STOC), pp 1–9
94. Stoffel D, Kunz W (1997) Record and play: a structural fixed point iteration for sequential circuit verification. In: Proceedings of the international conference on computer-aided design (ICCAD), November 1997, pp 394–399
95. Strichman O (2000) Tuning SAT checkers for bounded model checking. In: Emerson EA, Sistla AP (eds) Proceedings of the 12th international conference on computer-aided verification (CAV), July 2000. Lecture notes in computer science, vol 1855. Springer, Berlin Heidelberg New York, pp 480–494
96. Strichman O (2001) Pruning techniques for the SAT-based bounded model checking problem. In: Margaria T, Melham TF (eds) Proceedings of the 11th advanced research working conference on correct hardware design and verification methods (CHARME), September 2001. Lecture notes in computer science, vol 2144. Springer, Berlin Heidelberg New York, pp 58–70
97. Strichman O (2002) On solving Presburger and linear arithmetic with SAT. In: Aagaard M, O’Leary JW (eds) Proceedings of the 4th international conference on formal methods in computer-aided design (FMCAD), November 2002. Lecture notes in computer science, vol 2517. Springer, Berlin Heidelberg New York, pp 160–170

98. Tseitin GS (1968) On the complexity of derivation in propositional calculus. In: Slisenko AO (ed) *Studies in constructive mathematics and mathematical logic. Seminars in mathematics*, vol 8. Steklov Mathematical Institute, Leningrad, Russia, pp 234–259 (English Translation: Consultants Bureau, New York, 1970, pp 115–125)
99. van Eijk CAJ (1998) Sequential equivalence checking without state space traversal. In: *Proceedings of Design Automation and Test in Europe (DATE)*, February 1998, pp 618–623
100. Velev MN, Bryant RE (2003) Effective use of Boolean satisfiability procedures in the formal verification of superscalar and VLIW microprocessors. *J Symbol Comput* 35(2):73–106
101. Wang C, Li B, Jin HS, Hachtel GD, Somenzi F (2003) Improving Ariadne’s bundle by following multiple threads in abstraction refinement. In: *Proceedings of the international conference on computer-aided design (ICCAD)*, November 2003, pp 408–415
102. Whittemore JP, Kim J, Sakallah KA (2001) SATIRE: A new incremental satisfiability engine. In: *Proceedings of the 38th conference on design automation (DAC)*, June 2001, pp 542–545
103. Williams PF, Biere A, Clarke EM, Gupta A (2000) Combining decision diagrams and SAT procedures for efficient symbolic model checking. In: Emerson EA, Sistla AP (eds) *Proceedings of the 12th international conference on computer-aided verification (CAV)*, July 2000. *Lecture notes in computer science*, vol 1855. Springer, Berlin Heidelberg New York, pp 124–138
104. Yen C-C, Chen K-C, Jou J-Y (2002) A practical approach to cycle bound estimation for property checking. In: *Proceedings of 11th international workshop on logic and synthesis (IWLS)*, June 2002, pp 149–154
105. Zhang H (1997) SATO: An efficient propositional prover. In: McCune W (ed) *Proceedings of the 14th international conference on automated deduction (CADE)*, July 1997. *Lecture notes in computer science*, vol 1249. Springer, Berlin Heidelberg New York, pp 272–275
106. Zhang L, Madigan CF, Moskewicz MH, Malik S (2001) Efficient conflict driven learning in a Boolean satisfiability solver. In: *Proceedings of the international conference on computer-aided design (ICCAD)*, November 2001, pp 279–285
107. Zhang L, Malik S (2002) The quest for efficient Boolean satisfiability solvers. In: Brinksma E, Larsen KG (eds) *Proceedings of the 14th international conference on computer-aided verification (CAV)*, July 2001. *Lecture notes in computer science*, vol 2404. Springer, Berlin Heidelberg New York, pp 17–36
108. Zhang L, Malik S (2002) Towards symmetric treatment of conflicts and satisfaction in quantified Boolean satisfiability solvers. In: Van Hentenryck P (ed) *Proceedings of the 8th international conference on principles and practice of constraint programming (CP)*. *Lecture notes in computer science*, vol 2470. Springer, Berlin Heidelberg New York, pp 200–215
109. Zhang L, Malik S (2003) Validating SAT solvers using an independent resolution-based checker: practical implementations and other applications. In: *Proceedings of Design Automation and Test in Europe (DATE)*, March 2003, pp 880–885