Artif Intell Rev (2019) 52:2575–2601
https://doi.org/10.1007/s10462-018-9628-0

A comprehensive study and analysis on SAT-solvers: advances, usages and achievements

Sahel Alouneh[1] · **Sa'ed Abed**[2] · **Mohammad H. Al Shayeji**[2] · **Raed Mesleh**[1]

Published online: 28 March 2018

Abstract Boolean satisfiability (SAT) has been studied for the last twenty years. Advances have been made allowing SAT solvers to be used in many applications including formal verification of digital designs. However, performance and capacity of SAT solvers are still limited. From the practical side, many of the existing applications based on SAT solvers use them as blackboxes in which the problem is translated into a monolithic conjunctive normal form instance and then throw it to the SAT solver with no interaction between the application and the SAT solver. This paper presents a comprehensive study and analysis of the latest developments in SAT-solver and new approaches that used in branching heuristics, Boolean constraint propagation and conflict analysis techniques during the last two decade. In addition, the paper provides the most effective techniques in using SAT solvers as verification techniques, mainly model checkers, to enhance the SAT solver performance, efficiency and productivity. Moreover, the paper presents the remarkable accomplishments and the main challenges facing SAT-solver techniques and contrasts between different techniques according to their efficiency, algorithms, usage and feasibility.

Keywords SAT-solvers · CNF · EUF · Verification techniques · BMC · UMC

1 Introduction

These days, satisfiability checking (SAT) based tools have attained lots of attention since they have the advantage of being less sensitive to the size and the state explosion problem over the binary decision diagram (BDD) (Bryant 1992) based model checking. Therefore, representing the transition relation in conjunctive normal form (CNF) based SAT solvers

✉ Sahel Alouneh
sahel.alouneh@gju.edu.jo

1 School of Electrical Engineering and Information Technology, German Jordanian University, Amman, Jordan

2 Computer Engineering Department, Kuwait University, Kuwait City, Kuwait

is considered an attractive and effective approach for decision diagrams than BDDs. Thus, numerous researchers have concentrated on developing functions for execution bounded model checking (BMC) (Bjesse and Claessen 2000; Ganai and Aziz 2002; Abdulla et al. 2000) based on SAT technique. Enhancing the performance of the SAT solver is the main goal. The conception is to convert the given circuit into a SAT format by creating suitable propositional Boolean formulae and then exploiting other non-canonical representations of state sets. On the other hand, all approaches utilize SAT solvers capability to discover a satisfiable solution, if exists. Currently, the technology of SAT solvers have significant enhancement due to the availability of many classy and refined packages. Examples of well-known and famous state of the art SAT solvers are CHAFF (Moskewicz et al. 2001), GRASP (Marques-Silva and Sakallah 1999), MiniSat (Sorensson and Een 2005), MAX-SAT (Argelich and Manyà 2006), SATO (Zhang 1997), etc. The maximum satisfiability problem (MAX-SAT) is the problem of determining the maximum number of clauses that can be made true. It is a generalization of the Boolean satisfiability problem, which asks whether there exists a truth assignment that makes all clauses true. Argelich and Manyà (2006) designed a Max-SAT-like solvers that deal with blocks of clauses instead of individual clauses, and exploit the new structure of the encodings. Because model checker approaches manipulate state sets and since they can be represented as Boolean formulae, SAT solvers have massively improved their performance.

The International SAT competition, organized since 2002, is considered as a competitive affair for evaluating the progress in state-of-the-art for solvers of the Boolean satisfiability (SAT) problem to highlight the development in SAT tools. The competition compares different SAT tools in many hardware/software verification problems and evaluates their performance. The main goal of the competition is to help and motivate implementers to present their work to a broader audience and to compare it with that of others (Järvisalo et al. 2012).

Most of SAT solvers are based on the developments carried out on the original Davis–Putnam algorithm (Davis and Putnam 1960) where satisfiable instances of SAT were solved using local search algorithms. These algorithms dividing the search area among processors using variables partial assignments. Then, each processor has its own assigned area and can communicate with other processors if processor completes searching its corresponding search area. The drawback of these algorithms is that they are not scalable in terms of memory space due to data redundancy where each processor retains all clauses and variables. This paper discusses different new approaches that used in branching heuristics, Boolean constraint propagation (BCP) and conflict analysis to enhance the performance of the SAT solver.

In addition, the paper studies the up-to-date advances in the SAT-based verification approaches, mainly model checking approaches, and the methods with some kind of comparison between these techniques, how they are used in practice, including many references and others during the last two decades. Also, the paper targets the usage of SAT solvers as verification techniques to enhance the efficiency of the SAT solver. In addition, the paper discusses the automatic target generation process (ATGP) based SAT techniques to enhance their speedup while retaining robustness. Finally, the paper proposes the achievements, improvements and the most important challenges in SAT-solver techniques and contrasts between these techniques according to their efficiency, algorithms, usage and feasibility.

The paper is organized as follows: Section 2 reviews some basics on SAT solvers including algorithms and advanced feature of recent SAT solver. Section 3 discusses the work done on pseudo Boolean (PB) constraints. In Sect. 4, we review the Equality with Uninterpreted Functions (EUF) and how SAT deal with this type of theories and also discuss the CNF-based SAT solver. Section 5 reviews the decision procedure approach while Sect. 6 details the SAT-solvers based model checking. Section 7 discusses the ATPG-based SAT techniques.

Section 8 discusses the parallel SAT-solvers. Section 9 concludes the paper and gives some future research trends and directions.

2 Preliminaries on SAT solver

Two kinds of algorithms for solving SAT instances exist: the first one is the stochastic local search (SLS) algorithms (not complete or do not prove unsatisfiability) which can get solution quickly such as WalkSAT (Selman et al. 1994), the second one is called complete (they are guarantee to find satisfying assignment or prove unsatisfiability) like recent variations of DPLL algorithm such as Chaff (Moskewicz et al. 2001) and GRASP (Marques-Silva and Sakallah 1999) and will be our focus on this work.

2.1 Stochastic local search (SLS) algorithm

In the past two decades, a large number of algorithms used to solve SAT problems have been proposed and investigated. Stochastic local search (SLS) algorithms and genetic algorithms are also used to solve SAT problems in case of limited information of the problem structure instances. Some definite kinds of random satisfiable SAT instances are solved using survey propagation (SP) (Moskewicz et al. 2001) especially in applications of hardware and verification designs. The main problem with SLS algorithms is the cycling problem where a candidate solution can be revisited again (Cai and Su 2011). Random walk and restarting strategies were used to handle this problem. Another approach called a configuration check (CC) (Cai et al. 2011) was proposed to deal with the cycling problem in SLS. Other strategies using CC idea were developed later as in Luo et al. (2014). Cai and Su (2013) proposed a CC strategy for SAT called Swcc suitable for random 3-SAT instances. They enhanced the CC strategy by using an aspiration technique to get a new variable selection heuristic called configuration checking with aspiration (CCA) which improved the algorithm called Swcca. Later, Luo et al. (2014) proposed a new heuristic called DCCA, which combines two CC strategies with different definitions of configuration to design an efficient heuristic SLS solver for SAT dubbed DCCASat. For more details, many local search SAT algorithms have been presented and evaluated (under benchmarks-specific 'noise levels') in Hoos and Stützle (2000).

Xu et al. (2008) proposed and evaluated SATzilla07 solver based on constructing the portfolio by merging local search solvers, predicting performance score, and using hierarchical hardness models. Lindauer et al. (2015) addressed and applied the Algorithm selection (AS) procedure using automated algorithm configuration. They used dubbed AUTOFOLIO to model several AS methods and measured their performances.

Audemard and Simon (2009) suggested using the literal block distance, which is the number of decision literals that caused the conflict, to predict how useful a learned clause will be for other nodes. Ehlers et al. (2014) suggested limiting the literal block distance to be at most four. Glucose, the solver used by Ehlers et al. (2014), uses the literal block distance and clause length to priorities clauses and uses the activity factor to break ties. To increase the threads diversification, the thread ID is used as a seed to give random values to literals activity factors.

pprobSAT solver is a competitive parallel solver based on probSAT that runs n instances of probSAT together with the last two instances using restarts after 10^7 and 10^8 flips, respectively (The international SAT Competitions, http://satcompetition.org/edacc/sc14/experiment/29/solver-configurations/1561).

Table 1 Different variations of few SLS solvers

Solver	Heuristic	Completeness
GSAT (Selman et al. 1992)	Flip variable with max score If more than one has max score, flip one of them at random	Essentially incomplete
GWSAT (Hoos and Stützle 2000)	Flip variable with max score with probability or, flip a variable randomly from a random clause with probability	Satisfies the probabilistic approximate completeness property
WalkSAT (Hoos 2002)	Choose a clause at random If it has a variable with score equals to zero, flip it Otherwise flip variable with minimum score with probability, or flip a variable at random with probability If there are more than one variable with the same minimum score, one of them will be chosen at random	Unknown
Novelty$^+$ (Hoos 1999)	Choose a clause at random Flip a variable at random with probability else Flip variable with max score, if it is not the most recently flipped variable If it is the most recently flipped, it will be flipped with probability, or the variable with the next max score is flipped	Satisfies the probabilistic approximate completeness property
Novelty^{++} (Li and Huang 2005)	Choose a clause at random. Flip the least recently flipped variable with probability else Flip variable with max score, if it is not the most recently flipped variable If it is the most recently flipped, it will be flipped with probability, or the variable with the next max score is flipped	Satisfies the probabilistic approximate completeness property
G^{2}W SAT (Li et al. 2007)	Flip a promising decreasing variable, if none exists Choose a clause at random Flip the least recently flipped variable with probability else Flip variable with max score, if it is not the most recently flipped variable If it is the most recently flipped, it will be flipped with probability, or the variable with the next max score is flipped	

Different Variations of SLS Solvers are shown in Table 1 such as GSAT (Selman et al. 1992), GWSAT (Hoos and Stützle 2000), WalkSAT (Hoos 2002), Novelty+ (Hoos 1999), Novelty++ (Li and Huang 2005) and G^2W SAT (Li et al. 2007). Table 1 presents a summary of the different SLS solvers used in incomplete SAT algorithms.

2.2 DPLL algorithms

The second class of algorithms used in solving the satisfiability problem is called the complete algorithm where the SAT solver based on the modern improvements or variations of a Davis–Putnam–Logemann–Loveland (DPLL) algorithm. The DPLL SAT solver is basically based on backtracking search method to find the satisfying assignments for the variable in the search space. The search technique was presented in previous papers in the 60s and is currently denoted as the "DPLL" or "DLL" algorithm. The DPLL algorithm is sound and complete (Marić and Janičić 2010), this means that it will find the solution if and only if the formulae is satisfiable.

Two flavors of most current SAT solvers: *conflict-driven* (Formisano and Vella 2014) and *look-ahead* (Giunchiglia et al. 2003) solvers. *Conflict-driven* solvers expanded DPLL search routine by adding effective features for handling SAT instances in electronic design automation (EDA) tools such as conflict analysis, clause learning, non-chronological backtracking (back jumping), "two-watched-literals" unit propagation, adaptive branching and random restarts. *Look-ahead* solvers supported reductions and the heuristics, they have the advantage over *conflict-driven* solvers on difficult and tough instances while *conflict-driven* solvers are better on big instances. Pseudo code adapted from the basic Davis–Putnam search pro-

Algorithm 1 DAVIS_PUTNAM()

```
1: while true do
2:    if DecideNextBranch()==false then
3:       return(SATISFIABLE);
4:    end if
5:    while Deduce()==CONFLICT do
6:       if ResolveConflict()==false then
7:          RETURN(UNSATISFIABLE);
8:       end if
9:    end while
10: end while
```

cess is shown in Algorithm (1). The DecideNextBranch() is decision function used to choose a non-assigned variable and assign it a value. The operation Deduce() executes the BCP to propagate variable assignments based on current decision. Basically, if a clause contains only literals with value 0 and has one unassigned literal, then the unassigned literal must take a 1 value. This case is considered as a unit clause rule. In the pseudo-code, Deduce() keeps executing the BCP until no more implications exist and returns SATISFIABLE or a conflict is occurred (assigning the same variable both 1 and 0 values) and returns UNSATISFIABLE.

Each DPLL or SAT solver algorithm contains three main parts: Branching Heuristics, Deduction Mechanism and BCP and Conflict Analysis and Learning. In the following subsections, we present the well-known methods used to implement the operation for each part or function of the DPLL.

2.2.1 Branching heuristics

Branching heuristics select a sequence of decision variable to recognize a faster satisfying assignment. Researchers proposed many intelligent decision heuristics; static (does not consider search history) or dynamic. Here, we summarize the most common heuristic decisions.

Table 2 Comparing usage of VSIDS in Chaff and MINISAT

	VSIDS in CHAFF (Moskewicz et al. 2001)	VSIDS in MINISAT (Sorensson and Een 2005)
Initialization	All literal counters are initialized to 0	All variable counters are initialized to 0
Assignment	An unassigned literal with the maximum counter value is chosen arbitrarily at each decision point. After that, all counters are scaled down by a constant, occasionally	Instead of a decay factor, variable counters are "bumped" with larger and larger values in a floating point representation. When very large values are encountered, all counters are scaled down
Procedure	Once a clause is added to the clause database, each counter related to literal is incremented. The counter monitors the literal usage	2% of the time, a random decision is made instead. This factor is set at run-time

The first DPLL, RAND, uses simple decision heuristics to choose randomly from the unassigned variable the next decision variable. *Maximum occurrences on minimum sized clauses* MOM (Freeman 1995) heuristics employs greedy routines. Based on the maximum number of clauses or implications, the routine chooses the decision variable. Dynamic decision heuristics are of interest these days and thus, in the following, we review the most known techniques and compare between them.

Dynamic largest combined sum (DLCS) is introduced in GRASP SAT solver (Marques-Silva and Sakallah 1999). It selects the *variable* with largest number of occurrences in unresolved clauses. The differentiation between learned and original clauses is required. GRASP proposed the *dynamic largest individual sum* (DLIS) decision heuristic. The *literal* with largest number of unresolved clauses is selected.

Variable State Independent Decaying Sum (VSIDS) is introduced first in chaff (Moskewicz et al. 2001) and later used in many other SAT solver such as MINISAT (Sorensson and Een 2005). Basically, the VSIDS works as follow: Each literal, l, has a score s(l) and an occurrence count r(l). When a decision is necessary, a free literal with the highest score is set true. Initially, for every literal, l, s(l) = r(l) = 0. Before the search begins, s(l) is incremented for each occurrence of a literal, l, in the input formula. When a clause c is learned during search, r(l) is incremented for each literal $l \in c$. Every 255 decisions, the scores are updated: for each literal, l, s(l) becomes r(l) + s(l)/2, and r(l) becomes zero. This approach is considered cheap and effective on a variety of problems.

Table 2 shows the VSIDS branching heuristics technique used in chaff (Moskewicz et al. 2001) compared to the one used in MINISAT (Sorensson and Een 2005).

The *variable move-to-front* (VMTF) is an another decision method based on the VISDS (basically it is proposed to solve the problem that VSIDS facing), where Ryan (2004) shows that if the solver uses VMTF instead of VSIDS, far fewer decisions are needed to solve benchmarks from various interesting domains: planing, bounded model checking, circuit equivalence, and so on.

Finally, the method used in BerkMin (Goldberg and Novikov 2007) is similar to chaff method but in case of a conflict, the literals scores have increased. More details about the differences between the method used in BerkMin and VMTF method is included in Ryan (2004). Table 3 compares different techniques used in branching heuristics and shows their usage.

Table 3 Techniques used in branching heuristics and their usage

Technique	Used in	Usage
DLCS	GRASP (Marques-Silva and Sakallah 1999)	Picks the variable with largest number of occurrences in unresolved clauses
DLIS	GRASP (Marques-Silva and Sakallah 1999)	Selects the literal that appears in the largest number of unresolved clauses
VSIDS	Chaff (Moskewicz et al. 2001)	See Table 2
	Minisat (Sorensson and Een 2005)	
BerkMin	BrekMin (Goldberg and Novikov 2007)	Increment the literal scores responsible for the conflict or contradiction

With many available heuristics techniques, it is not easy to decide which one is the best. Each heuristic technique behaves different based on the problem kinds. For example, VSIDS is very inexpensive to compute compared to the most effective formulae simplification heuristics. Also, VMTF is cheaper to compute than VSIDS. The VSIDS implementation in zChaff accounts for about ten percent of the runtime. Most of that time is spent on sorting literals by score but the VMTF implementation accounts for less than one percent of solver's runtime according to the results presented in Ryan (2004).

2.2.2 Deduction mechanism and Boolean constraint propagation

Deduction mechanism is considered the most important mechanism because it takes a lot of runtime (almost 90 percent of runtime is spent here). All SAT solvers use the unit clause rule.

Example 1 $(Y_1 + Y_2) \cdot (Y_1 + Y_3) \cdot (Y_1' + Y_3 + Y_4)$

For the variable Y_1 or Y_2, the implemented circuit should be set to "1" if Y_3 is set to "1" or if Y_3 is set to "0" and Y_4 is set to "1". The clauses are called unit clauses and the unassigned literal is called a unit literal, the process of assigning 1 to all such literals is called BCP. Thus, effective BCP method is essential to SAT solver. BCP implemented using Deduce() function in Algorithm(1) which returns SAT if the assignment causes a satisfying instance and CONFLICT if it causes a conflict, else returns UNKNOWN and updates the program state.

Figure 1 shows the most BCP approach used in SAT. However, there are three methods for Deduction Mechanism summarized as follows:

1. *Counter method* is used in GRASP (Marques-Silva and Sakallah 1999): a simple method used for keeping track of which clauses are satisfied or conflicting where each clause has 3 variables; 1-value literal numbers, 0-value literal numbers and the total number of literals.
2. *Head–tail list* (HTL) algorithm introduced in SATO (Zhang 1997): each clause maintains two pointers the head and tail pointer. Also, each variable maintains 4 linked lists: pos-head, neg-head, pos-tail and neg-tail.
3. *Two-literal watch* (TLW) algorithm was first used in Chaff (Moskewicz et al. 2001): a method presented the watched literals usage to watch and monitor any two unassigned literals in each clause. Each clause and each variable has two lists: pos-watched and neg-watched. The clause can not be unsatisfied given that the two watched literals are

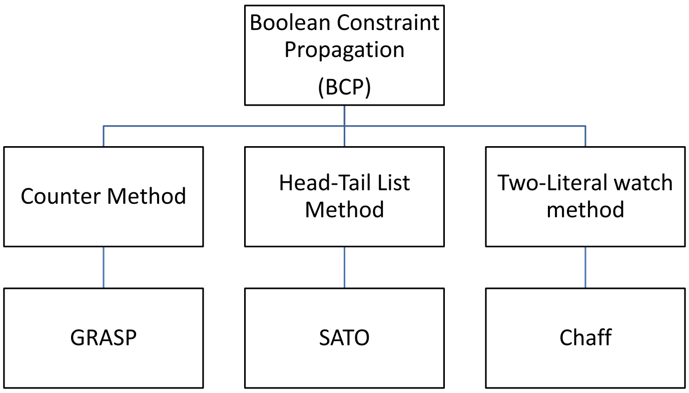

Fig. 1 Boolean constraint propagation

unassigned. Therefore, the clause can only be visited when any of the two watched literals is given to zero. Results indicate that the watched literals implementation lead to good enhancements compared to conventional BCP implementations, particularly for circuits with huge numbers of clauses (Aloul 2006).

2.2.3 Conflict analysis

Conflict Analysis discovers the reason of conflicting and solve it. It is considered a significant mechanism, which influences the search space portion that must be explored. DPLL algorithm used Chronological Backtracking in case of a conflict occurs. At the decision point, a SAT solver unassigns all variable assignments and flips the value of the decision variable, in which the last assignment that has not been flipped is flipped.

In order to deal with a conflict, we can flip the value of the decision assignment, undo all the implications and allow BCP to proceed normally in the manner described in Algorithm 2.

Many existing work emphasizing on how to improve the conflict analysis (Jin and Somenzi

Algorithm 2 RESOLVECONFLICT()

```
1: dec =latest decision not tried both ways;
2: if dec == NULL then
3:     return(false);
4: end if
5: complement the dec value;
6: mark dec as tried both ways;
7: undo any invalidated implications;
8: return(true);
```

2006; Fu and Malik 2004; Nadel 2002). Jin and Somenzi (2006) compared their technique with all these techniques. They suggested adding more than one conflict clause to reduce the literals number in conflict clauses and the implication graphs depth.

GRASP (Marques-Silva and Sakallah 1999) proposed a technique using conflict diagnosis in case of conflict. The method is based on identifying the variable assignments, which makes the clause unsatisfiable. These assignments are then used to build a conflict-induced clause database to avoid producing any similar conflict (pruning the search space).

Modern SAT solvers such as GRASP (Marques-Silva and Sakallah 1999) has independently contributed techniques for conflict analysis and conflict-driven learning. An implication graph, during conflict analysis, archives assignments to variables due to decisions as well as implications. Whenever a conflict occurs, a conflict clause is resulted by traversing the implication graph backward from a conflicting point up to some chosen cut-set.

Grasp also used conflict clauses to monitor the search space. A clause having a Unique Implication Point (UIP) is asserted after backtracking, and hence advances in testing all possible solutions. Some methods extract a clause having UIPs from the implications chains which cause a conflict (implication graph of the conflict) studied in Zhang et al. (2001). They stated that selecting the first UIP (the closest to the conflict) performed well.

In non-chronological backtracking, SAT solver backtracks to the decision variable (to earlier levels) that cause the conflict. This step helps in pruning huge parts of the search space.

One of the most essential components of conflict analysis is an implication graph (Tseitin 1968). Several clauses can be obtained from the same implication graph and several implication graphs can be derived from the same sequence of decisions based on the propagation of the implications. Fu and Malik (2004) and Nadel (2002) suggested enhancements in the conflict clauses feature derived by amending the implication graph generated by the SAT solver. Fu and Malik (2004) offered a scheme during propagating implications, which modifies the antecedents of a variable in case a smaller clause is detected. Thus, a smaller conflict clause can be found by the conflict analysis. In Nadel (2002), Jerusat solver used technique called shrinking which eliminates some literals from a conflict clause produced by conflict analysis. For the same conflict, it creates new and reduced implication graph.

Example 2 (Implication graph) : *Consider the CNF formulae*

$$\begin{aligned}\Psi &= w_1 \wedge w_2 \wedge w_3 \wedge w_4 \wedge w_5 \wedge w_6\\ &= (x_1 \vee x_7 \vee \neg x_1) \wedge (x_1 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee x_4)\\ &\quad \wedge(\neg x_4 \vee \neg x_5) \wedge (x_8 \vee \neg x_4 \vee \neg x_6) \wedge (x_5 \vee x_6) \qquad (1)\end{aligned}$$

Let's make the decision assignments $x_8@2$ and $x_7@3$. Also, make the present decision assignment $x_1 = 0@5$. Figure 2 shows the corresponding implication graph which produces a conflict since clause $(x_5 \vee x_6)$ is unsatisfiable. To implement non-chronological backtracking (Marques-Silva and Sakallah 1999), the learnt clauses are then used. Most SAT-solvers as SAT11 (Balint et al. 2013) are based on backtracking approach.

Chauhan et al. (2002) focused on automating the abstraction process for handling large designs. So, they presented an automatic abstraction refinement framework based SAT based conflict analysis. They described two methods to generate abstract systems by eliminating invisible variables. The first is abstraction by making invisible variables as input variables, and the second is abstraction by pre-quantifying invisible variables.

Improved conflict analysis algorithm proposed in Jing et al. (2009). The authors evaluated the conflict characters in model checking and offered their new method. The process adds some learnt clauses to the database while the implication graph depth reaches a specific level in the conflict analysis process. They have implemented the algorithm based on the model checking software TIP and compared their method with the algorithm that only adds a first unique implication point (FUIP) clause in case of conflict. They discovered that their algorithm decreases the number of BCP and increases the speed of SAT solver (Sorensson and Een 2005).

In Table 4, we compare the strengths and weaknesses between the mentioned conflict analysis methods.

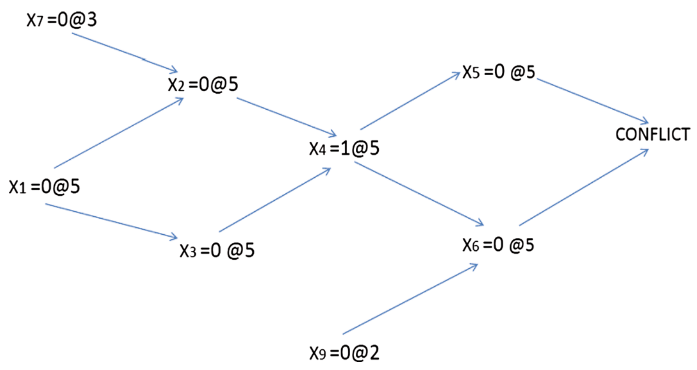

Fig. 2 Implication graph for Example 2

Table 4 Techniques used for conflict analysis

	Implication graph method	More than one conflict clause	Compact conflict clause	Shrinking
Advantages	Different implication graphs are generated from the same sequence of decisions	Literals number in conflict clauses and the implication graphs depth are reduced	Have limited literals is useful since it leads to fast BCP and identifies conflicts earlier	Costly due to the needed amount of Backtracking operation. In addition, it does not guarantee a reduction in literals number
Disadvantages	Deep implication Graphs on same clause makes the search well in hot spots, may slow down the BCP	May substantially slow down BCP	May slow down BCP	Produces a new smaller implication graph for the same conflict

2.3 Other technique algorithm

There are also other random methods but are fundamentally different. A relatively new method is called Survey Propagation (SP) which was discovered in 2002 by *Marc, Mzard, Giorgio Parisi* and *Zecchina*. The solvers based on SP are like belief propagation methods that are used for decoding error-correcting codes. In each iteration, about 10% of the literals are set. When the formulae density gets too low, SP switches to a random local search algorithm.

Presently, the basic DPLL still the main algorithm for SAT solving and most of the enhancements carried out later on BCP, decision heuristic, conflict analysis, etc., where to improve the performance of these algorithms by modifying the manner the algorithm works.

3 Pseudo Boolean constraints

Boolean satisfiability problems can contain Pseudo Boolean (PB) terms as given below:

Table 5 Comparison between PB SAT-solvers

SAT-Solver	Modification on the SAT	Modification on the PB
PBS (En and Srensson 2006)	Yes	No
PUEBLO (Mcmillan et al. 2009)	Yes	No
GALENA (Chai and Kuehlmann 2003)	Yes	No
MINISAT (Olivier Bailleux and Roussel 2006)	No	Yes

$$l_1 \times_1 + l_2 \times_2 + \cdots + l_n \times_n \leq b \tag{2}$$

such that l_i and x_i are Boolean variables literals. Any arbitrary PB constraint can be transformed to the normalized form which enables more effective SAT processes.

Table 5 shows how to deal with pseudo-Boolean constraints (PB-constraints). There are two choices: modifying the SAT procedure itself to provide PB-constraints with instances (Aloul et al. 2002; Sheini and Sakallah 2006; Bozzano et al. 2005). Or taking the opposite approach, which is based on the conversion of PB-constraints to SAT without modifying the SAT solver (En and Srensson 2006).

In En and Srensson (2006), the primary technique used in MINISAT is to translate linear PB constraints into circuits and then to clauses by a different *Tseitin* transformation (Tseitin 1968; Mcmillan et al. 2009). Three different techniques are used to convert the constraint into a BDD, a network of adders and a network of sorters. The techniques are evaluated using random benchmarks. The results show that the conversion via adders were not efficient. It appears better to use the sorter-translation.

In Chai and Kuehlmann (2003), the authors indicated that the capability of BCP using the watch-literal approach could be expanded for PB constraints and accepted to determine the proof for the cutting plane circuits. In Olivier Bailleux and Roussel (2006), the authors proposed a method to encode PB constraints into CNF formulae. This method allows unit propagation to apply global arc-consistency. The offered encoding method was combined with a zChaff SAT solver and submitted to the PB estimation.

Three basic methods are used to solve the PB constraint: integer linear programming methods, Pure SAT-based methods, and hybrid methods which is proposed in Sheini and Sakallah (2006). In Sheini and Sakallah (2006), the authors investigate learning methods based on cutting planes for solving PB constraints. On the learning front, they proposed hybrid algorithm which efficiently combines the CNF and PB learning methods in order to produce both a CNF clause and a PB constraint in case of a conflict. In this combined procedure, the CNF learning guarantees the correctness and completeness of the process by always producing a unit CNF clause, and terminates the learning process as soon as the first UIP is reached.

General method like two watched literals was extended for PB as carried out by Chai and Kuehlmann (2003) and Dixon and Ginsberg (2002). They proposed a fast PB constraint solver built on generalization of multiple conceptions learned from recent SAT solvers.

Linear pseudo-Boolean (LPB) constraints represent inequalities among the total of weighted Boolean functions and afford a modeling power expansion of propositional constraints. LPB is used to define different automation linear designs with constraints, weighted Boolean variables and effective search policies for demonstrating if a satisfying solution occurs.

Several approaches have been explored to develop new LPB constraints. Table 6 compares between SAT solvers (minisat, Chaff, SATO, Rsat (Pipatsrisawat and Darwiche 2007) and

Table 6 Comparison between different SAT-solvers

SAT-solver	BCP	Branching heuristics	Conflict analysis
Minisat (Olivier Bailleux and Roussel 2006)	Two-watch literal	VSIDS	First-UIP
SATO (Zhang 1997)	Head–tail list method	DLCS	Non-chronological backtracking
Rsat (Pipatsrisawat and Darwiche 2007)	Two-watch literal	VSIDS	First-UIP
Grasp (Marques-Silva and Sakallah 1999)	Counter method	DLCS	Non-chronological backtracking
Chaff (Moskewicz et al. 2001)	Two-watch literal	VSIDS	First-UIP

GRSAP) and the techniques used for the BCP , Branching Heuristics and the conflict analysis for each of them. As indicated from the table, some SAT solvers share the same techniques but the differ in how it is implemented such as: Minisat and the Chaff. Both use VSIDS for the Branching Heuristics while some differences exist between them as explained in Table 2.

MINISAT is considered as one of the common high-performance satisfiability solver used to implement various heuristics in a brief style. It won many awards in the SAT 2005 competition due to its open-source. In addition, it is used in many applications and verification tools (software and hardware) and detective provers as in Satisfiability Modulo Theories (SMT) solver MATHSAT (Bozzano et al. 2005). OPENSMT (Bruttomesso and Sharygina 2009) is written in C++, and based on MiniSat 2.0 (Sorensson and Een 2005). In first-order theorem proving, the author's presented a strong instantiation-based system IPROVER (Korovin 2008) based on MINISAT.

In order to increase the efficiency, MiniSAT 2.0 (Sorensson and Een 2005) is modified in Chu et al. (2009) to utilize new systems that enhance its caching system and hence its propagation speed. Furthermore, the authors proved their enhancements using a simple parallelization of MiniSat 2.0 (PMiniSat).

Other new promising solvers are Lingeling (Biere 2010), Glucose (Audemard 2013) and Riss (Manthey 2011) which won on the SAT competitions. The idea of lingeling (Biere 2010) is to use much less space than other solvers by using implicit clauses to have differing sizes of elements in a watchlist. Glucose solver (Audemard 2013) improves an intensive assumption-based incremental SAT solving task: Minimal Unsatisfiable Set. Riss solver (Manthey 2011) is component based and able to enable most of the recently developed techniques in SAT solving and preprocessing a formula.

4 EUF and CNF

In this section, we focus on various methods to convert the Equality with Uninterpreted Functions (EUF) formulae to propositional logic. Then, we describe different procedures for the translation of propositional formulae to CNF.

4.1 EUF

A EUF formula is defined as a Boolean formula of atoms with equalities among terms. The formula has truth (T) value whereas term has a value from other domain. The logic of EUF presented by Burch and Dill (1994). The formula: $f(x) \neq f(y) \wedge x = z \wedge z = y$ is unsatisfiable.

In order to solve the problem of deciding the validity of an EUF formulae, Ackermann (1954) and Bryant et al. (1999) showed that this problem can be minimized to checking the equality formulae satisfiability. There are many current approaches (Bryant and Velev 2002; Goel et al. 2003) use a transformation of an EUF formulae into the equality logic formulae. Then, the equality logic formulae can be transformed into a propositional formulae where a standard satisfiability checker can be applied.

In Bryant and Velev (2002) and Goel et al. (2003), the authors reduced an equality formulae to a propositional one by adding transitivity constraints and then analyzing which transitivity properties may be relevant. In Tveretina and Wesselink (2009), the authors presented the algorithm to solve satisfiability problem for EUF logic which is based on the generalized DPLL (GDPLL) procedure (Mcmillan et al. 2009). Based on this procedure, EufDpll tool is used to check the satisfiability of EUF formulas. They proved the correctness and completeness of the GDPLL procedure for the EUF-logic, and explained the reduction rule for EUF-CNFs for some basic function such as: REDUCE, Eligible, SatCriterion, and Filter. GDPLL continues until the SatCriterion function terminated with satisfiable for one branch, at least, or empty clause for all branches (unsatisfiable).

In Kozawa et al. (2007), the authors propose a satisfiability checking method with equivalence constraint (a formula representing the functions and predicates properties). The algorithm begins first by converting the EUF formula to CNF formula and then it's satisfiability is checked. If it is unsatisfiable, then the FQ (CNF formulae under equivalence constraints) is unsatisfiable too. In case F is satisfiable, then the solution is checked for conflict with Q (equivalence constraints). If we get a NO answer, then it is satisfiable else (there is conflict) and we need to resolve it.

In Velev (2004), the authors combined an automatic generated case-splitting expressions and effective conversion to CNF by generating If-Then-Else (ITE), and combining ITE-trees with two levels of their leaves to validate an out of order superscalar processor using Hardware Description Language AbsHDL based on EUFM logic. This was validated based on EVC decision procedure integrated with a SAT-solver.

4.2 CNF

The normal CNF translation is achieved by the distributive properties of $\wedge$ and $\vee$; which results in an exponential growth in the formula size. There are two methods for CNF formulae simplification: using preprocessors namely *Hyper* (Bacchus and Winter 2003) and *NIVER* (Subbarayan and Pradhan 2004), and using the generalized implication-based reasoning (Arora and Hsiao 2004).

Many existing methods for CNF generation are presented in Fig. 3. First CNF generation based on Tseitin's linear-time algorithm (Tseitin 1968). The *AIGER* tool includes a very efficient implementation of *Tseitin's* algorithm (Biere 2015).

Velev presented in Velev (2004) another CNF generation algorithm, where he defined certain patterns arising in formulas from pipelined machine verification problems and provided CNF generation methods for these patterns. He also proposed an approach for converting Boolean formula to CNF by finding gates of one fan-out count and combining them with their fan-out gate to produce an equivalent CNF clause. Thus, eliminating CNF variables and clauses, and thus speeding up the BCP as well as the SAT-solving. The translation to CNF by merging gates has different strategies depend on which group of gates to be merged.

Boy de la Tour (1992) showed how to compute the set of internal nodes by introducing a variable for them to minimize the number of clauses generated. The algorithm proposed in Jackson and Sheridan (2005) is considered as a generalization to *Velev's* method. In other

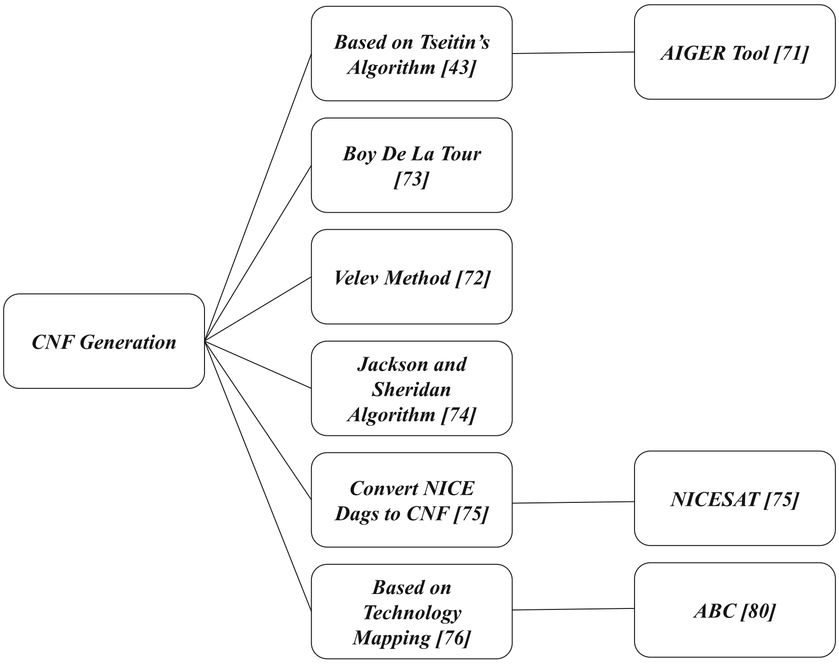
CNF Generation
Based on Tseitin's Algorithm [43]
AIGER Tool [71]
Boy De La Tour [73]
Velev Method [72]
Jackson and Sheridan Algorithm [74]
Convert NICE Dags to CNF [75]
NICESAT [75]
Based on Technology Mapping [76]
ABC [80]

Fig. 3 CNF generation methods

word, this algorithm introduces a new variable for an argument to a disjunction only if that leads to fewer clauses.

New linear-time CNF generation algorithm is proposed in Chambers et al. (2009) to produce smaller CNF and then faster SAT solver. In this regard, the authors presented NICE dags, a structure used to represent various circuits, and then introduced a linear time procedure that translates NICE dags to CNF. They developed NICESAT, a C++ implementation that operates on NICE dags and Boolean expressions.

In Een et al. (2007), the authors provided an approach to CNF generation based on technology mapping. They explored preprocessing of problem based SAT circuits by two latest developments in logic synthesis. The first one is a Directed Acyclic Graph DAG-aware logic minimization by making modifications for the AIG nodes to cuts (rewrites) of the graph. The second one is a new structural technology mapping to minimize the CNF size obtained from the circuit.

In Bjesse and Boralv (2004), DAG-aware circuit compression was presented and later modified in Manolios and Vroon (2007) and Mishchenko et al. (2006). Een et al. (2007) show that the circuit can be reduced using several conversions based on logic sharing. Reducing nodes number of a circuit leads to minimize the derived CNFs size fed to the SAT engine. The method is comparable to CNF preprocessing since a reduced representation is reached via several rewrites.

A new CNF translation algorithm was proposed in Jackson and Sheridan (2004) where a new clause form translation called compact translation is used. The authors described the CNF conversion over the RBC tree rather than general graph due to the sharing technique usage. Experimental work showed that the compact translation produces less clauses and the solving time is enhanced in most suitcases.

A ASIG, an all-solution solver for satisfiability problems, was introduced in Zhao and Wu (2009). ASIG operates on conjunctive normal forms (CNFs) instead of circuits or hybrid representations as most other all-solution solvers did. ASIG prevents the same solution from being found again by creating a blocking clause from a solution and adding it into the clause database.

In Arora and Hsiao (2004), the authors presented a suite of lemmas and theorems based on non-trivial Boolean reasoning. These lemmas and theorems analyze a set of clauses (in the original CNF formulae) affected by a given state of assignments, and infer new clauses based on the information relating to Satisfiability of these affected clauses. This reasoning helps to identify highly non-trivial clauses which aid in the identification of unit literals, equivalent/complement literals, and other implication relationships. These learned relationships reduce the CNF instance complexity immensely.

Hamadi et al. (2009) presented a dynamic subsumption technique for Boolean CNF formula. It exploited enough and necessary conditions to discover the clauses that can be minimized by subsumption based on conflict analysis. During the learnt clause derivation, it tests for backward subsumption among the current resolving and the original clauses. The resulting approach permits the dynamic elimination of literals from the original clauses. The integration of their dynamic subsumption approach with the Minisat and Rsat solvers profits to manufactured circuits.

In order to provide different generation models of SAT instances, Ansótegui et al. (2009) extended the uniform and regular 3-CNF models. They proposed a generalization of the uniform and the regular k-CNF (Calabro et al. 2010) random generation models by generalizing the probability distribution used on the selection of variables to a geometric and a power law distribution. An important result is that all their models guarantee the existence of the phase transition phenomena. They generated instances at the phase transition point of any given number of variables and computational hardness by adjusting the parameters of the distributions. In order to be able to generate bigger instances with variable clause length, they provided a fifth model, double power law, where they assigned a different probability of being chosen to each variable and to each clause. This generates formulas were some variables occur very often and some clauses are very long.

In Andraus and Sakallah (2004), the authors described *Vapor*; an automatic tool that abstracts RTL behavioral in *Verilog* to the CLU language used by UCLID system. *Vapor* does a sound abstraction to minimize false negatives. *Vapor* was built in *C++* for *Linux*, and combined with UCLID and *Verilog Icarus* compiler.

5 Decision procedure

In Bryant et al. (2007), a decision procedure for quantifier-free bit-vector arithmetic that uses automatic abstraction-refinement proposed. This procedure is implemented in the verification system UCLID. It alternates between generating *under-* and *over-approximations* of the original bit-vector formula. From an input bit-vector formula, UCLID first builds an *under-approximation* formula by restricting the number of Boolean variables used to encode each bit-vector variable. If an under approximation formula is satisfiable, so is the original formula, and the algorithm terminates.

If it is found to be unsatisfiable, the SAT solver is able to output a resolution proof of this fact. Then by using unsatisfiable core, an over-approximation is built. This over-approximation uses the full set of bits of the original vectors, but only a subset of the constraints (determined by examining the unsatisfiable core of an under-approximation). If over-approximation is unsatisfiable, so is the original formula and UCLID terminates. Otherwise, the algorithm refines the under-approximation by increasing, for at least one bit-vector variable, the number of Boolean variables encoding it. This process is repeated until the original formula is shown to be either satisfiable or unsatisfiable.

In Kroening et al. (2010), they present a decision procedure for *Quantifier-Free Presburger* (QFP) that is based on alternately under and over-approximating a formula, where Boolean interpolants are used to compute the *over-approximation*. Given a QFP formula, first the QFP translate to a Boolean formula and check the satisfiablility using SAT. If it unsatisfiable, then they construct an *under-approximation* formula and check it's satisfiability, if it is unsatisfiable then they construct the formula of an over-approximation (by using Boolean interpolant formula and the theory formula). The satisfiability of an over-approximation is checked using a conflict clause generator CCG() for QFP. If CCG (over-approximation) returns *UNSATISFIABLE*, the algorithm returns UNSATISFIABLE. Otherwise, CCG (*over-approximation*) returns a set of (conflict) clauses, representing tautologies in the theory of QFP. The bound of each variable is increased.

We can change that algorithm by replace the CCG with a sound and complete lazy SAT-based decision procedure for QFP (Kroening et al. 2010).

6 SAT based model checking

Huge efforts are invested on improving SAT tools to carry out various kinds of model checking due to the fact that they are less sensitive to the size and the state explosion problems of BDD. EDA researchers presented many techniques from SAT solvers world which can handle millions of variables and constraints and used them to develop model checking tools to investigate the correctness of hardware designs. Modern verification techniques such as bounded model checking (BMC), theorem proving and unbounded model checking (UMC) are all based on SAT solvers and their extensions (Vizel et al. 2015). 1999 was the year where Biere offered BMC based on SAT method (Biere et al. 1999). This integration provides MC a big push based on BMC (Reimer et al. 2014; Tsuchiya 2012). UMC combined with the techniques of theorem proving (Mcmillan 2003). In the following, we explain both bounded and unbounded model checking and it's combination with the SAT solver.

6.1 SAT-based bounded model checking

Consider M is a transition system, F is a temporal logic formula F and K is a user-supplied time. Then, the propositional formula $\Omega(k)$ is said to be satisfiable if and only if it is valid along a path of length K such that:

$$I(s_{s_0} \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1})$$

Most effort spent on SAT-based model checking adopts safety properties, for checking the safety properties the reachable states in k steps are captured by:

$$I(S_0) \wedge R(S_0, S_1) \wedge ... \wedge R(S_{k-1}, S_k)$$

The property p fails in one of the k steps:

$$\neg P(S_0) \vee \neg P(S_1) \vee ... \vee \neg P(S_k)$$

The safety property p is valid up to step k if and only if $\Omega(k)$ is unsatisfiable such that:

$$\Omega(k) = I(S_0) \wedge \bigwedge_{i=0}^{k-1} R(S_i, S_{i+1}) \wedge \bigvee_{i=0}^{k} \neg P(S_i)$$

Example: F = **EF** P and k = 2, then, $\Omega(2) = I(s_{s_0} \wedge R(s_0, s_1) \wedge R(s_1, s_2) \wedge (P_0 \vee P_1 \vee P_2)$

A *hydlogic* is a BMC tool used for hybrid systems as proposed in Ishii et al. (2011). The tools is based on converting a reachability analysis of a non-linear hybrid system into a predicate logic formula, which cannot be handled, by most of the existing tools. Then, it tests the formula satisfiability based on a SMT technique. The authors combined an incremental SAT solver to compute the constraints sets and an interval-based solver for *Hybrid Constraint Systems* (HCSs) to solve the constraints.

Wieringa et al. (2009) investigate approaches to parallelizing BMC for shared memory environments and clusters of workstations. They propose a general platform for parallelized BMC named *Tarmo* which permits using encoding of BMC instances into incremental SAT.

A work presented the relationship between SAT solvers proof system and general resolution is carried out by *Knot and Adnan* (Pipatsrisawat and Darwiche 2009). The result demonstrations that SAT solvers can simulate any resolution proof based empowerment and provability. These two concepts allow them to formalize the clause-learning SAT solver with restarts (CLR) and use it to simulate the resolution. The proof needs the solver to restart in case of conflict and proposes a normal restart policy to be a key to the SAT solvers efficiency. Results on different benchmarks show that their method significantly reduce the SAT runtime.

IBM’s sixth sense model for system verification is an example of this improvement as stated by Mony et al. (2004). Bentley (2005) points to the SAT usage to link dynamic simulation-based tools and formal verification tools and reflects this on microprocessor verification. The reader can expand their knowledge on the use of SAT solvers, BMC and induction tools by referring to Barrett et al. (2009).

Even SAT based BMC has doubly exponential complexity due to the SAT instance size defined as $O(k(|M| + |\Psi|))$ but it detects shallow errors effectively and handles hardware components where in many cases the longest path between any two reachable states is not exponential.

The complexity of BMC is determined based on the size of SAT instance given as $O(k(|M| + |\Psi|))$ where k can become as large as the diameter of the system, which is exponential in the number of state variables in the worst case. SAT is an exponential time complexity. Therefore, SAT based BMC has doubly exponential complexity. However, LTL model checking is singly exponential!

There are many reasons to answer the question *Why use SAT based BMC?* The answer because it can be infeasible to represent P explicitly, it can identify shallow errors efficiently, in many cases rd(P) and d(P) are not exponential and can be rather small (e.g. hardware components without counters), where rd is the recurrence diameter defined as the longest loop-free path between any two reachable states and d is the diameter defined as the longest shortest path between any two reachable states. Finally, because modern SAT solvers are very successful in practice.

6.2 SAT-based unbounded model checking

We briefly survey two approaches:

Sheeran et al. (2000) offered the first *Unbounded model checking* (UMC) SAT-based approach and extended later (Bjesse and Claessen 2000). It came as a result when the BMC can not prove that counter-example does not exist for a given safety property (Wu et al. 2013). Sheeran's work uses the standard BMC loop, with additional conditions for establishing completeness through induction. The main idea is that, if at a step i, no more loop-free paths of size i can exist or no more loop-free paths states can satisfy Fk, then no reachable state has a path to a state where Fk is satisfied.

McMillan's work uses the standard BMC loop, but also utilizes interpolates with the goal of obtaining approximations of the reachable states. In contrast to Sheeran et al. (2000) work, this approach ensures that maximum unfolding is bounded by diameter of the transition relation.

In Vizel and Grumberg (2009), the authors presented a new SAT-based methodology for verification. The method integrates BMC with interpolation-sequence to emulate BDD-based symbolic model checking. McMillan in Mcmillan (2003) offered a SAT-based model-checking routine suitable for full verification. His approach is based on merging BMC and Craig's interpolation. Vizel and Grumberg (2009) included a thorough comparison between their and McMillan method based on the algorithmic level and running experiments. For instance, unlike the interpolation-based model checking approach offered in Mcmillan (2003), their approach does not need consecutive BMC runs to calculate the reachable states.

The most successful approach for the unbounded model checking is called *induction* and *interpolants* following the definition for the both techniques: First the *interpolants* technique for a given two subsets of clauses X and Y, assume $X \wedge Y$ is unsatisfiable. Then, there exists a *interpolant* X' for the pair (X, Y) with the following properties:

1. X' denotes only to X and Y variables
2. X implies X'
3. $X' \wedge Y$ is unsatisfiable.

The second technique is the induction-based method (Sheeran et al. 2000) which uses a SAT solver as a decision procedure for a specific type of induction named k-induction. In order to check if the property holds in the current state we need to assume that it holds in the earlier k successive states. More detail about the *interpolant* and induction techniques is presented in Amla et al. (2005). A variety of methods to exploit SAT and BMC for unbounded model checking:

1. Completeness threshold
2. k-induction
3. Abstraction (refutation proofs useful here)
4. Exact and over-approximate image computations (refutation proofs useful here)
5. Use of Craig interpolation.

7 ATPG-based SAT

Automatic Target Generation Process (ATGP) is used to generate sequences of test vectors to identify the good and faulty circuit's behavior based on specific input patterns. The ATGP can be then solved by converting the circuit and the fault into SAT instances and use the SAT solver to evaluate the test (Becker et al. 2014).

Searching for vector sequences in the space of all possible input sequences for a given circuit to detect a fault for a certain signal in the circuit to logic 1 or 0 is known as *Sequential* (ATPG).

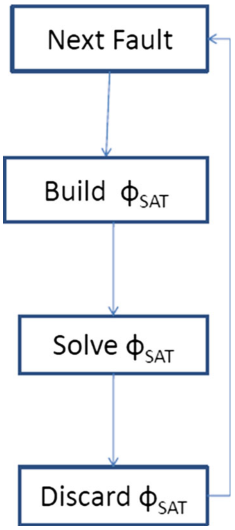

Fig. 4 Flow of classical ATPG based SAT technique

Many ATGP algorithms have been developed to handle combinatorial designs (D-Algorithm, PODEM and FAN) and sequential designs (State Table Verification Approach). Most classical ATPG algorithms are based on a circuit structure while ATPG algorithms based SAT handle Boolean formula in CNF format (Marques-Silva and Sakallah 1999; En and Srensson 2004).

ATPG algorithms based SAT have been proved to be an efficient complement to classical ATPG algorithms as stated in both Drechsler et al. (2008, 2009). They considered robust even they suffer from the overhead due to CNF transformation time while classical structural ATPG algorithms are fast. Therefore, many enhancements were carried out to speed up the ATPG algorithms based SAT techniques (Eggersgluss et al. 2009).

Figure 4 shows the flow of classical ATPG based SAT technique. The CNF for a given a circuit consisting of set of gates and signal lines is derived by assigning a Boolean variable to each signal line according to its logic value in the circuit. The gate is then converted to a set of clauses using a specific characteristic function. The CNF of the circuit is determined as the conjunction of the CNF of each gate. The final SAT instance ϕ_{SAT} is then obtained by a conjunction of CNF of the circuit with the fault specific constraints, e.g. the fault.

Boppana et al. (1999) started working on ATPG-based Model Checking to use sequential ATPG solvers to verify the safety property of the form AGEF*p* on a network circuit. More details on ATPG-based Model Checking can be found in Prasad et al. (2005).

8 Parallel Sat solver

Since the last two decades, there have been many implementations of SAT-solvers to be executed in clusters and grids, using distinct technologies. Some of them are GrADSAT (Chrabakh and Wolski 2003), NAGSAT (Forman and Segre 2002), PSATO (Zhang et al. 1996) and SATz (Jurkowiak et al. 2005). For the most part, all these solvers have a master/slave (or Task Farm) architecture where a master task sends out work and collects the results, while the clients run a sequential SAT-solver.

GrADSAT is a solver that is based on Chaff and runs over a grid of widely distributed computers that are dynamically acquired and released. Each client in the GrADSAT environment searches in some specific subspace, but learnt clauses are shared between clients (immediately after being generated), a characteristic that can improve performance as the sharing of clauses can potentially improve search space prunning, albeit at the cost of additional communication. PSATO, based on solver SATO, runs in clusters and it is implemented in C and uses the parallel language P4 (Butler and Lusk 1992) to manage concurrency and communication. PSATO can save checkpoints of the search allowing a temporary suspension of the program. Clients in PSATO also search in non-overlapping subspaces of the original problem.

SATz is based on SATz, also run in clusters, is implemented in C, communicates through RPC (Remote Procedure Call) and makes load balance using the work stealing technique, which amounts to repartitioning of subspaces that have been found to be too large for a single client to search in.

PMSat (Gil et al. 2008), that is based on MiniSAT and uses MPI technology, to be executed in clusters or in a grid of computers. It contains several features including a high degree of portability, different search modes, sharing of learnt clauses and pruning of the search space. Therefore, PMSat shares some common characteristics with other parallel versions of SAT solvers, such as GrADSAT, PSATO and //SATz. For once, it is based on partitioning of the domain or search space, an assignment undertaken by the master, which controls the scheduling of the clients and distributes the various tasks between them. The individual clients then perform the actual search corresponding to the task given. Unlike other solvers, however, more than one partitioning heuristic is available for the user as we will see in the following section. Like GrADSAT and PaSAT sharing of learnt clauses is allowed, albeit using a slightly different mechanism and heuristics. Conflict learning is also used to prune the outstanding tasks and potentially to stop running clients whose search space has been proven irrelevant.

They also split the search space, using different methods, and analyze each subspace in parallel, in separate clients. Their execution is time constrained and all of them are, to some extent, fault tolerant (Heule and Maaren 2008) observe that the probability of hitting a solution of propositional Boolean formulae is increased by assigning multi-bit values instead of Boolean values. They implemented the UnitWalk algorithm as a multi-bit local search solver using Unit-Propagation Queue. The resulting solver, called UnitMarch, can be used for any number of bits.

Conventional parallel Sat solving (Blochinger et al. 2003; Bhm and Speckenmeyer 1996; Zhang et al. 1996) differs from the proposed method in Heule and Maaren (2008). The former gains performance by dividing the workload over multiple processors and by some minor changes to the solving algorithm, while the latter uses a single processor and requires significant modifications to the algorithm. Both Heule and Maaren (2008) and Srensson (2008) are related to each other where Srensson (2008) also parallelizes a Sat solver (GSAT), on a single processor. However, they use a vector processor (used in most supercomputers), instead of scalar processor (used in most desktop computers).

Recently, parallel version of Z3 proposed in Wintersteiger et al. (2009), they parallelize the sequential solver by running multiple solvers, each configured to use different heuristics. To further improve the performance of their solver, they share derived lemmas between solvers (Sharing lemmas may prevent a solver from entering a search space that was previously investigated by another solver, thus improving the performance of the first solver).

9 Conclusions and future trends

9.1 Conclusion

The success of SAT-based verification techniques is due to its development as a technology to BDD-based model checking techniques since it is less sensitive to the size problem and state explosion problem. Therefore, SAT-based approaches can verify huge designs than those done by BDDs and hence improving the efficiency.

In this paper, we present a comprehensive study on the state of the art achievements in SAT-based verification during the last two decades. In addition, we discuss different new approaches that used in Branching Heuristics, BCP and conflict analysis to enhance the SAT solver productivity and efficiency. Moreover, we present the distinguished attainments and the main problems encounter SAT-based verification techniques and contrasts between different techniques according to their efficiency, algorithms, usage and feasibility.

9.2 Future trends and directions

An increased number of applications are being tackled by SAT-solvers due to its efficiency as in many EDA tools (Biere et al. 1999; Ivan 2013) to verify microprocessor (Bryant et al. 1999), automate test generation (Larrabee 1989) and other usages (Marques-Silva and Sakallah 2000). Current state of the art solvers being amended to satisfy some explicit features of these domains further amplify this achievement. The scheme represents a search problem instance as a propositional formula, and then runs a SAT solver to look for an assignment if exists is considered as an active approach for solving a number of problems. Perhaps, SAT-based bounded model checking (Shtrichman 2001) is extensively used verification method. In addition to that, these approaches are extended to un-bounded model checking (McMillan 2002). SAT solvers are used as an engine for several model checking tools (Abed et al. 2007; Hoque et al. 2012), for tools using more expressive logics (Lierler 2010; Lin and Zhao 2004), quantified Boolean formulae and modal logics (Giunchiglia et al. 2002), and even first order theorem proving (Naumowicz 2014; Kaivola et al. 2013).

For future trends, developing a verification approach to combine SAT and BDD approaches to benefit from their advantages and strengths is considered the main challenge. Another future trend is the implementation of an SMT-solver (Liu et al. 2013; Ogawa and Khanh 2013) which requires a theory solver to be able to work incrementally, compute minimal infeasible subsets, and backtrack on demand to previous computation steps. To our knowledge, currently none of these functions are supported by the available theory solvers for real algebra. If we take a look to the most recent SMT-solvers such as Z3 (De Moura and Bjørner 2008; Rodrguez Vega 2014), HySAT (Frnzle et al. 2007) and ABsolver (Bauer et al. 2007) which are able to handle arithmetic constraints. The algorithm implemented in HySAT uses interval arithmetic to check real constraints. However, Z3 does not support full non-linear arithmetic. The authors of ABsolver do not address the issues of incrementally and backtracking. Though ABsolver computes minimal infeasible subsets.

Another promising future work is the use of effective parallelization of SAT solvers (Audemard and Simon 2014; Yoo et al. 2014; Holldobler et al. 2011; Marques 2013; Arbelaez and Codognet 2013). Several SAT-solvers are implemented in clusters and grids such as GrADSAT (Chrabakh and Wolski 2003), NAGSAT (Forman and Segre 2002), PSATO (Zhang et al. 1996) and SATz (Jurkowiak et al. 2005). The common part among these solvers is a master/slave architecture. The master transmits the work and assembles the results, while the client executes the SAT-solver. The following are some keys for current and future research results:

1. High-level learning rather than learning at the clause level.
2. Effective partitioning of the original SAT problem such that learning from different partitions can be utilized in other partitions.
3. Implementing various solver and learning instances such that the complexity of the original problem is reduced in combination.
4. Utilizing SAT solvers with theorem proving and reasoning engine.

Based on this work, there are still many other research subjects to be considered to better development for effective implementations.

References

Abdulla PA, Bjesse P, Eén N (2000) Symbolic reachability analysis based on sat-solvers. In: Proceedings of the 6th international conference on tools and algorithms for construction and analysis of systems: held as part of the European joint conferences on the theory and practice of software, ETAPS 2000. Springer, London, pp 411–425

Abed S, Mohamed OA, Yang Z, Sammane GA (2007) Integrating SAT with multiway decision graphs for efficient model checking. In: Proceedings of IEEE ICM'07. IEEE Press, Egypt, pp 129–132

Ackermann W (1954) Solvable cases of the decision problem, 1st edn. North-Holland Publishing, North-Holland

Aloul FA, Ramani A, Markov IL, Sakallah KA (2002) Pbs: a backtrack search pseudo Boolean solver. In: Symposium on the theory and applications of satisfiability testing (SAT), pp 346–353

Aloul FA (2006) Search techniques for sat-based Boolean optimization, modeling, simulation and applied optimization. J Franklin Inst 343(4–5):436–447

Amla N, Du X, Kuehlmann A, Kurshan RP, Mcmillan KL (2005) An analysis of sat-based model checking techniques in an industrial environment. In: CHARME, pp 254–268

Andraus ZS, Sakallah KA (2004) Automatic abstraction and verification of verilog models. In: Proceedings of the 41st annual design automation conference. ACM, New York, pp 218–223

Ansótegui C, Bonet ML, Levy J (2009) Towards industrial-like random sat instances. In: Proceedings of the 21st international joint conference on artifical intelligence. Morgan Kaufmann Publishers Inc., San Francisco, pp 387–392

Arbelaez A, Codognet P (2013) A survey of parallel local search for sat. In: Theory, implementation, and applications of SAT technology. Workshop at JSAI 2013, pp 1–4

Argelich J, Manyà F (2006) Exact MAX-SAT solvers for over-constrained problems. J Heuristics 12(4–5):375–392

Arora R, Hsiao MS (2004) CNF formula simplification using implication reasoning. In: Proceedings of the high-level design validation and test workshop, ninth IEEE international. IEEE Computer Society, Washington, pp 129–134

Audemard G, Lagniez J-M, Simon L (2013) Improving glucose for incremental sat solving with assumptions: application to MUS extraction. In: International conference on theory and applications of satisfiability testing. Springer, pp 309–317

Audemard G, Simon L (2009) Predicting learnt clauses quality in modern sat solvers. In: Proceedings of the 21st international jont conference on artifical intelligence. Morgan Kaufmann Publishers Inc, San Francisco, pp 399–404

Audemard G, Simon L (2014) Lazy clause exchange policy for parallel SAT solvers. In: Proceedings of international conference on theory and applications of satisfiability testing—SAT 2014—17th, held as part of the Vienna summer of logic, VSL 2014, Vienna, Austria, pp 197–205

Bacchus F, Winter J (2003) Effective preprocessing with hyper-resolution and equality reduction. In: SAT, pp 341–355

Balint A, Belov A, Heule MJ, Järvisalo M (2013) Solver and benchmark descriptions. In: Proceedings of SAT competition 2013, vol B-2013-1. Department of Computer Science Series of Publications, University of Helsinki, Helsinki

Barrett C, Sebastiani R, Seshia S, Tinelli C (2009) Satisfiability modulo theories, frontiers in artificial intelligence and applications, vol 185, ch. 26, IOS Press, pp 825–885

Bauer A, Pister M, Tautschnig M (2007) Tool-support for the analysis of hybrid systems and models. In: Proceedings of the conference on design, automation and test in Europe. EDA Consortium, San Jose, pp 924–929

Becker B, Drechsler R, Eggersglü S, Sauer M (2014) Recent advances in sat-based ATPG: non-standard fault models, multi constraints and optimization. In: Proceedings of the 9th international conference on design & technology of integrated systems in nanoscale era, DTIS 2014, Santorini, Greece, pp 1–10

Bentley B (2005) Validating a modern microprocessor. In: Proceedings of the 17th international conference on computer aided verification. Springer, Berlin, pp 2–4

Bhm M, Speckenmeyer E (1996) A fast parallel sat-solver-efficient workload balancing'. In: Annals of mathematics and artificial intelligence, p 40

Biere A (2010) Lingeling, plingeling, picoSAT and precoSAT at SAT race 2010. Technical report 10/1, Institute for formal models and verification, Johannes Kepler University

Biere A (2015) AIGER format and toolbox. http://fmv.jku.at/aiger/

Biere A, Cimatti A, Clarke EM, Fujita M, Zhu Y (1999) Symbolic model checking using sat procedures instead of BDDS, pp 317–320

Bjesse P, Boralv A (2004) Dag-aware circuit compression for formal verification. In: Proceedings of the 2004 IEEE/ACM international conference on computer-aided design. IEEE Computer Society, Washington, pp 42–49

Bjesse P, Claessen K (2000) Sat-based verification without state space traversal. In: In formal methods in computer-aided design. Springer, pp 372–389

Bjesse P, Claessen K (2000) Sat-based verification without state space traversal. In: Proceedings of the third international conference on formal methods in computer-aided design. Springer, London, pp 372–389

Blochinger W, Sinz C, Kchlin W (2003) Parallel propositional satisfiability checking with distributed dynamic learning. Parallel Comput 29:969–994

Boppana V, Rajan SP, Takayama K, Fujita M (1999) Model checking based on sequential ATPG. In: Proceedings of the 11th international conference on computer aided verification. Springer, London, pp 418–430

Boy de la Tour T (1992) An optimality result for clause form translation. J Symb Comput 14(4):283–301

Bozzano M, Bruttomesso R, Cimatti R, Junttila T, Rossum PV, Schulz S, Sebastiani R (2005) The mathsat 3 system. In: Automated deduction: proceedings of the 20th international conference, volume 3632 of Lecture notes in computer science. Springer, pp 315–321

Bruttomesso R, Sharygina N (2009) OpenSMT 0.1

Bryant RE, Kroening D, Ouaknine J, Seshia SA, Strichman O, Brady B (2007) Deciding bit-vector arithmetic with abstraction. In : Proceedings of the 13th international conference on tools and algorithms for the construction and analysis of systems. Springer, Berlin, pp 358–372

Bryant RE (1992) Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Comput Surv 24(3):293–318

Bryant RE, Velev MN (2002) Boolean satisfiability with transitivity constraints. ACM Trans Comput Logic 3(4):604–627

Bryant RE, German S, Velev MN (1999) Exploiting positive equality in a logic of equality with uninterpreted functions. Springer, London, pp 470–482

Burch JR, Dill DL (1994) Automatic verification of pipelined microprocessor control. In: Proceedings of the 6th international conference on computer aided verification. Springer, London, pp 68–80

Butler R, Lusk E (1992) User's guide to the p4 parallel programming system

Cai S, Su K (2013) Local search for boolean satisfiability with configuration checking and subscore. Artif Intell 204:75–98. https://doi.org/10.1016/j.artint.2013.09.001

Cai S, Su K, Sattar A (2011) Local search with edge weighting and configuration checking heuristics for minimum vertex cover. Artif Intell 175(9–10):1672–1696

Cai S, Su K (2011) Local search with configuration checking for sat. In: 23rd IEEE international conference on tools with artificial intelligence (ICTAI), pp 59–66

Calabro C, Impagliazzo R, Paturi R (2010) On the exact complexity of evaluating quantified-CNF. In: IPEC, pp 50–59

Chai D, Kuehlmann A (2003) A fast pseudo-boolean constraint solver. In: Proceedings of the 40th annual design automation conference. ACM, New York, pp 830–835

Chambers B, Manolios P, Vroon D (2009) Faster sat solving with better CNF generation. In: Proceedings of the conference on design, automation and test in Europe, 3001 Leuven, Belgium. European Design and Automation Association, Belgium, pp 1590–1595

Chauhan P, Clarke EM, Kukula JH, Sapra S, Veith H, Wang D (2002) Automated abstraction refinement for model checking large state spaces using sat based conflict analysis. In: Proceedings of the 4th international conference on formal methods in computer-aided design. Springer, London, pp 33–51

Chrabakh W, Wolski R (2003) GrADSAT: a parallel SAT solver for the grid

Chu G, Harwood A, Stuckey PJ (2009) Cache conscious data structures for Boolean satisfiability solvers. J Satisf Boolean Model Comput 6:99–120

Davis M, Putnam H (1960) A computing procedure for quantification theory. J ACM 7(3):201–215

De Moura L, Bjørner N (2008) Z3: an efficient smt solver. In: Proceedings of the theory and practice of software, 14th international conference on tools and algorithms for the construction and analysis of systems. Springer, Berlin, pp 337–340

Dixon HE, Ginsberg ML (2002) Inference methods for a pseudo-Boolean satisfiability solver. In: Eighteenth national conference on artificial intelligence. American Association for Artificial Intelligence, Menlo Park, pp 635–640

Drechsler R, Eggersgluss S, Fey G, Glowatz A, Hapke F, Schloeffel J, Tille D (2008) On acceleration of sat-based ATPG for industrial designs. Trans Comput Aided Des Integ Circuit Syst 27(7):1329–1333

Drechsler R, Eggersgl S, Fey G, Tille D (2009) Test pattern generation using Boolean proof engines, 1st edn. Springer, London

Een N, Mishchenko A, Sörensson N (2007) Applying logic synthesis for speeding up sat. In: Proceedings of the 10th international conference on Theory and applications of satisfiability testing. Springer, Berlin, pp 272–286

Eggersgluss S, Tille D, Drechsler R (2009) Speeding up sat-based ATPG using dynamic clause activation. In: Asian test symposium. ATS ’09, pp 177–182

Ehlers T, Nowotka D, Sieweck P (2014) Communication in massively-parallel sat solving. In: 2014 IEEE 26th international conference on tools with artificial intelligence, pp 709–716

En N, Srensson N (2006) Translating pseudo-Boolean constraints into sat. J Satisf Boolean Model Comput 2:1–26

En N, Srensson N (2004) An extensible sat-solver. In: Giunchiglia E, Tacchella A (eds) Theory and applications of satisfiability testing. Lecture Notes in Computer Science, vol 2919. Springer, Berlin, pp 502–518

Forman SL, Segre A (2002) Nagsat: a randomized, complete, parallel solver for 3-sat. In: Fifth international symposium on the theory and applications of satisfiability testing

Formisano A, Vella F (2014) On multiple learning schemata in conflict driven solvers. In: Proceedings of the 15th Italian conference on theoretical computer science, Perugia, Italy, pp 133–146

Freeman JW (1995) Improvements to propositional satisfiability search algorithms, Philadelphia, uMI Order No. GAX95-32175

Frnzle M, Herde C, Teige T, Ratschan S, Schubert T (2007) Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure. J Satisf Boolean Model Comput 1:209–236

Fu YMZ, Malik S (2004) New features of the SAT’04 versions of zChaff

Ganai MK, Aziz A (2002) Improved sat-based bounded reachability analysis. In: Proceedings of the 2002 Asia and South Pacific design automation conference. IEEE Computer Society, Washington, pp 729–735

Gil L, Flores P, Silveira LM (2008) PMSat: a parallel version of MiniSAT. J Satisf Boolean Model Comput 6:71–98

Giunchiglia E, Tacchella A, Giunchiglia F (2002) Sat-based decision procedures for classical modal logics. J Autom Reason 28:143–171

Giunchiglia E, Maratea M, Tacchella O (2003) Look-ahead versus look-back techniques in a modern sat solver. In: SAT03—Sixth international conference on theory and applications of satisfiability testing, Portofino

Goel A, Sajid K, Zhou H, Aziz A, Singhal V (2003) Bdd based procedures for a theory of equality with uninterpreted functions. Form Methods Syst Des 22(3):205–224

Goldberg E, Novikov Y (2007) Berkmin: a fast and robust sat-solver. Discrete Appl Math 155(12):1549–1561

Hamadi Y, Jabbour S, Saïs L (2009) Learning for dynamic subsumption. In: Proceedings of the 2009 21st IEEE international conference on tools with artificial intelligence. IEEE Computer Society, Washington, pp 328–335

Heule MJ, Van Maaren H (2008) Parallel sat solving using bit-level operations. J Satisf Boolean Model Comput 99–116

Holldobler S, Manthey N, Nguyen VH, Stecklina J, Steinke P (2011) A short overview on modern parallel sat-solvers. In: International conference on advanced computer science and information system (ICACSIS), pp 201–206

Hoos H (1999) On the run-time behaviour of stochastic local search algorithms for sat. In: Sixteenth national conference on artificial intelligence and the eleventh innovative applications of artificial intelligence conference innovative applications of artificial intelligence. American Association for Artificial Intelligence, Menlo Park, pp 661–666

Hoos HH (2002) An adaptive noise mechanism for walksat. In: Eighteenth national conference on artificial intelligence. American Association for Artificial Intelligence, Menlo Park, pp 655–660

Hoos HH, Stützle T (2000) Local search algorithms for sat: an empirical evaluation. J Autom Reason 24(4):421–481. https://doi.org/10.1023/A:1006350622830

Hoque KA, Mohamed OA, Abed S, Boukadoum M (2012) Mdg-sat: an automated methodology for efficient safety checking. Int J Crit Comput Based Syst 3(1/2):4–25

Ishii D, Ueda K, Hosobe H (2011) An interval-based sat modulo ode solver for model checking nonlinear hybrid systems. Int J Softw Tools Technol Transf 13(5):449–461

Ivan T (2013) An efficient hardware implementation of a sat problem solver on FPGA. In: Proceedings—16th Euromicro conference on digital system design, DSD 2013. Universit de Montral, Montral, Department d'informatique et recherche oprationnelle, Montral, pp 209–216

Jackson P, Sheridan D (2004) The optimality of a fast CNF conversion and its use with sat. APES Research Group. Technical report APES-82-2004

Jackson P, Sheridan D (2005) Clause form conversions for Boolean circuits. In: Proceedings of the 7th international conference on theory and applications of satisfiability testing. Springer, Berlin, pp 183–198

Järvisalo M, Le Berre D, Roussel O, Simon L (2012) The international sat solver competitions. AI Mag 33(1):89–92

Jing M, Yin W, Chen G, Zhou D (2009) Enhance sat conflict analysis for model checking. In: IEEE 8th international conference on ASIC, 2009, ASICON '09, pp 686–689

Jin H, Somenzi F (2006) Strong conflict analysis for propositional satisfiability. In Proceedings of the conference on design, automation and test in Europe, 3001 Leuven, Belgium. European Design and Automation Association, Belgium, pp 818–823

Jurkowiak B, Li CM, Utard G (2005) A parallelization scheme based on work stealing for a class of sat solvers. J Autom Reason 34(1):73–101

Kaivola R, O'Leary J, Melham T (2013) Relational STE and theorem proving for formal verification of industrial circuit designs. In: Proceedings of the 2013 international conference on formal methods in computer-aided design, formal methods in computer-aided design (FMCAD). Springer, London, pp 97–104

Korovin K (2008) iProver—an instantiation-based theorem prover for first-order logic (system description). In: Proceedings of the 4th international joint conference on automated reasoning. Springer, Berlin, pp 292–298

Kozawa H, Hamaguchi K, Kashiwabara T (2007) Satisfiability checking for logic with equality and uninterpreted functions under equivalence constraints. IEICE Trans Fundam Electron Commun Comput Sci 90(12):2778–2789

Kroening D, Leroux J, Rmmer P (2010) Interpolating quantifier-free presburger arithmetic. In: Fermller C, Voronkov A (eds) Logic for programming, artificial intelligence, and reasoning. Lecture Notes in Computer Science, vol 6397. Springer, Berlin, pp 489–503

Larrabee T (1989) Efficient generation of test patterns using boolean difference. In: Test conference, proceedings. Meeting the tests of time, international, pp 795–801

Li CM, Huang WQ (2005) Diversification and determinism in local search for satisfiability. In: Proceedings of the 8th international conference on theory and applications of satisfiability testing, SAT'05, pp 158–172

Li CM, Wei W, Zhang H (2007) Combining adaptive noise and look-ahead in local search for sat. In: Proceedings of the 10th international conference on theory and applications of satisfiability testing. Springer, Berlin, pp 121–133

Lierler Y (2010) Sat-based answer set programming. Ph.D. dissertation, Department of Computer Sciences, The University of Texas at Austin, Austin

Lindauer M, Hoos HH, Hutter F, Schaub T (2015) Autofolio: an automatically configured algorithm selector. J Artif Int Res 53(1):745–778

Lin F, Zhao Y (2004) Assat: computing answer sets of a logic program by sat solvers. In: Artificial intelligence, nonmonotonic reasoning, vol. 157(1–2), pp 115–137. http://www.sciencedirect.com/science/article/pii/S0004370204000578

Liu L, Kong W, Ando T, Yatsu H, Fukuda A (2013) A survey of acceleration techniques for SMT-based bounded model checking In: International conference on computer sciences and applications (CSA), pp 554–559

Luo C, Cai S, Su K, Wu W (2014) Clause states based configuration checking in local search for satisfiability. IEEE Trans Cybern 99:1–1

Luo C, Cai S, Wu W, Su K (2014) Double configuration checking in stochastic local search for satisfiability. In: Proceedings of the twenty-eighth AAAI conference on artificial intelligence, pp 2703–2709 (in press)

Manolios P, Vroon D (2007) Efficient circuit to cnf conversion. In: Proceedings of the 10th international conference on Theory and applications of satisfiability testing. Springer, Berlin, pp 4–9

Manthey N (2011) Solver submission of RISS 1.0 to the sat competition 2011. SAT Competition

Marić F, Janičić P (2010) Formal correctness proof for dpll procedure. Informatica 21:57–78

Marques R (2013) Parallel sat solver. Universidade Tcnica de Lisboa

Marques-Silva JP, Sakallah KA (2000) Boolean satisfiability in electronic design automation. In: Proceedings of the 37th annual design automation conference. ACM, New York, pp 675–680

Marques-Silva J, Sakallah K (1999) GRASP: a search algorithm for propositional satisfiability. IEEE Trans Comput 5(48):506–521

Mcmillan KL, Kuehlmann A, Sagiv M (2009) Generalizing dpll to richer logics. In: Proceedings of the 21st international conference on computer aided verification. Springer, Berlin, pp 462–476

McMillan KL (2002) Applying sat methods in unbounded symbolic model checking. In: Proceedings of the 14th international conference on computer aided verification. Springer, London, pp 250–264

Mcmillan KL (2003) Interpolation and sat-based model checking. In: Hunt J, Warren A, Somenzi F (eds) Computer aided verification. Lecture Notes in Computer Science, vol 2725. Springer, Berlin, pp 1–13

Mishchenko A, Chatterjee S, Brayton R (2006) Dag-aware aig rewriting a fresh look at combinational logic synthesis. In: Proceedings of the 43rd annual design automation conference. ACM, New York, pp 532–535

Mony H, Baumgartner J, Paruthi V, Kanzelman R, Kuehlmann A (2004) Scalable automated verification via expert-system guided transformations. In: Hu A, Martin A (eds) FMCAD, Lecture Notes in Computer Science, vol 3312. Springer, Berlin, pp 159–173

Moskewicz MW, Madigan CF, Zhao Y, Zhang L, Malik S (2001) Chaff: engineering an efficient sat solver. In: Annual ACM IEEE design automation conference. ACM, pp 530–535

Nadel A (2002) The jerusat sat solver. Master's thesis, Hebrew University of Jerusalem

Naumowicz A (2014) SAT-enhanced mizar proof checking. Springer, Berlin, pp 449–452

Ogawa M, Khanh T (2013) Sat and SMT: their algorithm designs and applications. In: Software engineering conference (APSEC), 20th Asia-Pacific, vol 2, pp 83–84

Olivier Bailleux YB, Roussel O (2006) A translation of pseudo Boolean constraints to sat. J Satisf Boolean Model Comput 2:191–200

Pipatsrisawat K, Darwiche A (2007) A lightweight component caching scheme for satisfiability solvers. In: Proceedings of 10th international conference on theory and applications of satisfiability testing (SAT), pp 294–299

Pipatsrisawat K, Darwiche A (2009) On the power of clause-learning sat solvers with restarts. In: Proceedings of the 15th international conference on principles and practice of constraint programming. Springer, Berlin, pp 654–668

Prasad MR, Biere A, Gupta A (2005) A survey of recent advances in sat-based formal verification. Int J Softw Tools Technol Transf 7(2):156–173

Reimer S, Sauer M, Schubert T, Becker B (2014) Using MAXBMC for pareto-optimal circuit initialization. In: Design, automation and test in Europe conference and exhibition (DATE), pp 1–6

Rodrguez Vega M (2014) Analyzing toys models of arabidopsis and drosphila using z3 SMT-lib, vol 9118, pp 13–15

Ryan L (2004) Efficient algorithms for clause-learning sat solvers. Simon Fraser University, Burnaby

Selman B, Kautz HA, Cohen B (1994) Noise strategies for improving local search. In: Proceedings of the eleventh national conference on artificial intelligence (AAAI-94), pp 337–343

Selman B, Levesque H, Mitchell D (1992) A new method for solving hard satisfiability problems. In: Proceedings of the tenth national conference on artificial intelligence, pp 440–446 (in press)

Sheeran M, Singh S, Stålmarck G (2000) Checking safety properties using induction and a sat-solver. In: Proceedings of the third international conference on formal methods in computer-aided design. Springer, London, pp 108–125

Sheini HM, Sakallah KA (2006) Pueblo: a hybrid pseudo-Boolean sat solver. J Satisf Boolean Model Comput 2:165–189

Shtrichman O (2001) Pruning techniques for the sat-based bounded model checking problem. In: Proceedings of the 11th IFIP WG 10.5 advanced research working conference on correct hardware design and verification methods. Springer, London, pp 58–70

Sorensson N, Een N (2005) Minisat v1.13—a SAT solver with conflict-clause minimization. In: Eighth international conference on theory and applications of satisfiability testing (SAT 2005), vol 3569. Springer, St. Andrews

Srensson N (2008) Effective sat solving. Ph.D. dissertation, Department of Computer Science and Engineering Chalmers University of Technology and University of Gothenburg SE-412 96, Gteborg, Sweden

Subbarayan S, Pradhan DK (2004) Niver: non increasing variable elimination resolution for preprocessing sat instances. In: Proceedings of the 7th international conference on theory and applications of satisfiability testing (SAT). Springer, pp 276–291

The International SAT Competitions: SAT competition 2014, experiments: parallel, random SAT track: solver configurations: pprobSAT details. http://satcompetition.org/edacc/sc14/experiment/29/solver-configurations/1561

Tseitin GS (1968) On the complexity of derivations in the propositional calculus. Stud Math Math Logic Part II:115–125

Tsuchiya T (2012) Model checking that uses satisfiability solving. Comput Softw 29(1):19–29

Tveretina O, Wesselink W (2009) Eufdpll—a tool to check satisfiability of equality logic formulas. Electron Not Theor Comput Sci 225:405–420

Velev MN (2004) Efficient translation of Boolean formulas to CNF in formal verification of microprocessors. In: Proceedings of the 2004 Asia and South Pacific design automation conference. IEEE Press, Piscataway, pp 310–315

Velev MN (2004) Using automatic case splits and efficient CNF translation to guide a sat solver when formally verifying out-of-order processors. In: Artificial intelligence and mathematics (AIMATH '04), pp 242–254

Vizel Y, Grumberg O (2009) Interpolation-sequence based model checking. In: FMCAD, pp 1–8

Vizel Y, Weissenbacher G, Malik S (2015) Boolean satisfiability solvers and their applications in model checking. In: Proceedings of the IEEE, vol 99, pp 1–15

Wieringa S, Niemenmaa M, Heljanko K (2009) Tarmo: a framework for parallelized bounded model checking. In: Brim L, van der Pol J (eds) Proceedings of the 8th international workshop on parallel and distributed methods in verification (PDMC'09), electronic proceedings in theoretical computer science (EPTCS), vol 14 pp 62–76

Wintersteiger CM, Hamadi Y, Moura L (2009) A concurrent Portfolio approach to SMT solving. In: Proceedings of the 21st international conference on computer aided verification. Springer, Berlin, pp 715–720

Wu CY, Wu CA, Lai CY, Huang CY (2013) A counterexample-guided interpolant generation algorithm for sat-based model checking. In: Design automation conference (DAC), 2013 50th ACM/EDAC/IEEE, pp 1–6

Xu L, Hutter F, Hoos HH, Leyton-Brown K (2008) Satzilla: Portfolio-based algorithm selection for sat. J Artif Int Res 32(1):565–606

Yoo T, Kim S, Yeom Y, Kang J (2014) A study of the parallelization of hybrid sat solver using cuda. Adv Sci Technol Lett 48(1/2):19–24

Zhang H (1997) Sato: an efficient propositional prover. In: Proceedings of the 14th international conference on automated deduction. Springer, London, pp 272–275

Zhang H, Bonacina MP, Paola M, Bonacina HJ (1996) Psato: a distributed propositional prover and its application to quasigroup problems. J Symb Comput 21:543–560

Zhang L, Madigan CF, Moskewicz MH, Malik S (2001) Efficient conflict driven learning in a Boolean satisfiability solver. In: Proceedings of the 2001 IEEE/ACM international conference on computer-aided design. IEEE Press, Piscataway, pp 279–285

Zhao W, Wu W (2009) Asig: an all-solution sat solver for CNF formulas. In: 11th IEEE international conference on computer-aided design and computer graphics, CAD/Graphics '09, pp 508–513