

Simple Yet Efficient Improvements of SAT Based Bounded Model Checking

Emmanuel Zarpas

IBM Haifa Research Laboratory
zarpas@il.ibm.com

Abstract. In this paper, we show how proper benchmarking, which matches day-to-day use of formal methods, allows us to assess direct improvements for SAT use for formal methods. Proper uses of our benchmark allowed us to prove that previous results on tuning SAT solver for Bounded Model Checking (BMC) were overly optimistic and that a simpler algorithm was in fact more efficient.

1 Introduction

Over the past decade, verification via model checking has evolved from a theoretical concept to a production-level technique. It is being actively used in chip design projects across the industry, where formal verification engineers can now tackle the verification of large industrial hardware designs.

Assessing theoretical results requires more than simply getting experimental results on some benchmarks. Rather, results must be attained for a wide range of benchmarks, under conditions as close as possible to real-life formal verification. In order to illustrate the significance of proper benchmarking, we demonstrate how applying overly restrictive benchmarking can be misleading. We, then, show how proper benchmarking, which matches day-to-day use of formal methods, allows us to assess direct improvements. We present a method of splitting up the problem, which verification engineers usually meet in day-to-day real-life work. This allows us to distribute the problem in a simple and efficient way. This distribution technique is simple, but we, nevertheless, prove that it is very efficient by benchmarking. In a more general way, our goal is to show how very simple ideas can be proven to have great impact by experimentation.

This paper is organized as follows: Section 2 discusses our work in tuning decision-heuristics for a SAT solver and shows how proper benchmarking proved that the results were too optimistic. Section 3 explains how we assessed the performance of a straightforward distributed SAT Bounded Model Checking algorithm. Section 4 summarizes our conclusions. The benchmark we mainly used in this paper (the IBM Formal Verification Benchmark), is presented in Appendix A.

2 Decision Heuristics for Tuning SAT

This section presents some decision heuristics for tuning the zChaff VSIDS for bounded model checking as in [15]. The experimental results use a predecessor

of the IBM Formal Verification Benchmark. The next section shows how proper use of the IBM Formal Verification (Cf. appendix A). Benchmarking proves the results in this section are too optimistic.

2.1 Bounded Model Checking and SAT Basis

BMC Basis: BMC translates a safety formula from LTL [14] into a propositional formula under bounded semantics. The general structure of a $G(P)$ formula, as generated in BMC [5], is as follows:

$$\phi : I_0 \wedge \bigwedge_{i=0}^{k-1} \rho(i, i+1) \wedge \left(\bigvee_{i=0}^k \neg P_i \right)$$

where I_0 is the initial state, $\rho(i, i+1)$ is the transition between cycles i and $i+1$, and P_i is the property in cycle i .

If this propositional formula is proven to be satisfiable, the satisfying assignment provided by the SAT solver is a counterexample to the property $G(P)$. To convert the initial propositional formula into Conjunctive Normal Form (used as the input format by most SAT solvers), extra variables are introduced to avoid combinatory explosion. Usually, these extra variables represent more than 80% of the total number of variables in the CNF formula.

SAT Basis: SAT is the problem of determining the satisfiability of a Boolean formula. The problem was used by Cook to define NP-completeness [6]. Today, many implementations are available for solving the problem, such as Grasp[16] and zChaff [12]. Most of them are based on the complete DPLL algorithm [7], we now describe as in [12]:

```
while(true) {
  if (!decide()) // if no unassigned vars
    return(satisfiable) ;
  while (!bcp()) {
    if (!resolveConflict ())
      return (not satisfiable) ;
  }
}

bool resolveConflict() {
  d=most recent decision not 'tried both ways' ;
  if (d==null) // no such d was found
    return false ;

  flip the value of d;
  mark d as tried both ways ;
  undo any invalidated implications;
  return true;
}
```

decide() is a function that chooses the next variable according to which branching will occur. There are many heuristics for choosing this next variable, such as DLIS (Dynamic Largest Individual Sum) and VSIDS (Variable State Independent Decaying Sum). zChaff uses VSIDS as its decision heuristic. bcp() returns true when the boolean constraint propagation (bcp) [12] finishes without conflict. Our next subsection focuses on tuning this heuristic for BMC.

2.2 Tuning VSIDS

The original zChaff decision VSIDS strategy is as follows:

1. Each variable in each polarity has a counter, which is initialized to 0.
2. When a clause is added (by learning) to the database, the counter associated with each literal in the clause is incremented.
3. The (unassigned) variables and polarity with the highest counter are chosen at each decision.
4. Periodically, all counters are divided by a constant.

One of Strichman's [17] main ideas is as follows: in the Davis-Putnam decision procedure, the variable of the original propositional formula is used first and in a specific static order. This static order is determined by a breadth first search of the (k -unfolding of the) variable dependency graph; the search starts from the set $\bigcup_{0 \leq i \leq k} \neg P_i$. Roughly speaking the intuition behind this, is that the formula variables are the most critical. We chose to implement several decision heuristics on top of zChaff¹. We tuned the zChaff VSIDS decision heuristic in several different ways. First, we wanted to reflect the idea of static order (SO) [17]. Second, we wanted to implement a heuristic that gives priority to dominant variables (DV) over any other variables, but otherwise relies on zChaff decisions. This is because it is unclear from [17] whether the improvements were due to the static order or to the priority given to the dominant variables (*i.e.*, the variables from the initial propositional formula before the conversion to CNF, or domain variables). Because zChaff uses the VSIDS decision strategy, which is less time consuming than the decision heuristic used by Strichman with GRASP (*i.e.*, DLIS), it is not clear whether the benefits from a static order would still be realized. Therefore, we also implemented a static order heuristic and dominant variables priority heuristics to act as a tie breaker for the zChaff decision (respectively SB and DVB). The four decision strategies implemented are described in details in [15].

¹ We used the zChaff SAT solver because this solver has been stable for a few years and its source code is available for research and academic purposes. Even though zChaff did not win the SAT2003 contest, its results on the IBM benchmarks are comparable to the results achieved by the winning tools. In fact, our own experiment [23] with the IBM CNF benchmark show results slightly better for zChaff than for Berkmin561 [10]. We believe these heuristic results should be re-usable for other modern SAT solvers [10, 13, 8].

Table 1. The results are displayed in seconds, with two significant digits. Timeout was set to 10000 seconds

	VSIDS	DVB	SB	DV	SO	min(DVB,SB, DV,SO)
1_2001	18	12	31	14	170	(DVB) 12
2_2001	1400	620	820	300	1300	(DV) 300
3_2001	43	66	190	240	240	(DVB) 66
4_2001	4000	1000	800	210	120	(SO) 120
5_2001	2400	44	240	170	100	(DVB) 44
6_2001	1000	67	190	3800	250	(DVB) 67
7_2001	340	120	140	8	19	(DV) 8
8_2001	13	10	34	14	4800	(DVB) 10
9_2001	71	44	110	190	900	(DVB)44
10_2001	70	70	110	86	timeout	(DVB) 70
11_2001	4800	4200	6400	timeout	3100	(SO) 3100
12_2001	44	42	73	46	51	(DVB) 42
13_2001	78	6	58	6	52	(DVB) 6
14_2001	32	31	42	18	26	(DV) 18
15_2001	13	13	13	13	13	13
16_2001	timeout	timeout	9200	timeout	180	(SO) 180
17_2001	7600	4000	3100	2400	timeout	(DV) 2400
18_2001	11	85	95	6	3000	(DV) 6
19_2001	timeout	timeout	timeout	1600	8700	(DV) 1600

2.3 Experimental Results

For the experiments described in this paper we used zChaff 2001.2.17 (for a comparison between performances of zChaff 2001.2.17 and 2004.5.13, see for examples [23]).

In our first experiments, we used a predecessor of the IBM Formal Verification Benchmark. For each model we used only one CNF (see Table 1 contend for experimental results). In Table 2, we computed speedup, taking the VSIDS heuristic as a reference: therefore we did not take into account IBM_16 and IBM_19, for which VSIDS times out. We could also have decided to take the Static Order (SO) heuristic as a reference and not take IBM_10 and IBM_17 into account. However, SO is the only heuristic that times out for IBM_10 and IBM_17, while three out of five heuristics time out for IBM_16 and IBM_19. When we compared $\min(\text{DVB,SB,DV,SO})$ with VSIDS for each case, we were impressed: Running four concurrent instances of SAT, each with a different heuristic, should give a theoretical speedup greater than six. Indeed if we could build a tool which would run concurrently SAT instances with DVB, SB, DV and SO heuristics, this would solve SAT problems about six times faster than with VSIDS heuristics².

² Arguably we should have compared it to VSIDS run on six machines concurrently. Anyway we will see in the following that using these six heuristics concurrently gives, in real-life situations, unenthusiastic results.

Table 2. The results are computed without IBM_16 and IBM_19 for which VSIDS times out

	VSIDS	DVB	SB	DV	SO
Total time	21933	10430	12446	> 17521	> 34141
Global speedup (VSIDS total time/total time)		2.10	1.76	< 1.25	< 0.64
Average speedup (average VSIDS time/time)		6.11	2	< 6.26	< 5.15

In order to evaluate this claim, we made some additional experiments, with a different approach. First, we decided to use the IBM Formal Verification Benchmark, which is wider than its predecessors; the fact that it is available online (for academic organizations) allows to reproduce our results. Second, we decided not to run SAT as a stand-alone with one or two CNFs per model, but to run our global bounded model checking tool, as a regular user would, without any prior knowledge of the model. We began with a bound equal to 10, and in the case where no satisfiable assignment is found for the CNF generated by BMC translation, we incremented the bound by 5, and so on. In other words, in order to try to falsify a safety formula, say $G(P)$, we usually try to find a satisfiable assignment for

$$\phi_0 : I_0 \wedge \bigwedge_{i=0}^9 \rho(i, i+1) \wedge \left(\bigvee_{i=0}^{10} \neg P_i \right)$$

if it fails, we try

$$\phi_1 : I_0 \wedge \bigwedge_{i=0}^{14} \rho(i, i+1) \wedge \left(\bigvee_{i=11}^{15} \neg P_i \right)$$

and keep incrementing until a satisfiable assignment is found for some ϕ_j , some timeout is reached, or a user defined maximal bound is reached.

This is the most common approach for day-to-day use of BMC. Even when the completeness threshold d (as in [5]) is known, it is very often too big to allow the BMC to finish with $k = d$. Therefore, most of the time, users will try to falsify a formula with BMC, without knowing whether the formula holds or not, or which k would be needed to get a satisfiable ϕ_k .

We ran RuleBase with five concurrent SAT BMC instances (each of them using a different decision heuristic from VSIDS, DV, DVB, SB, SO). Each instance was independent and run on a single workstation with a 867841X Intel(R) Xeon(TM) CPU 2.40GHz with 512 KB first level cache, and 2.5 GB physical memory, running with Red Hat Linux release 7.3. Surprisingly, we got a speedup of only 1.14 (see the following section and Concur 5/5 heuristics results in Table 4). This can be explained by several factors:

- The search includes a SAT search together with pre-processing and BMC translation (translation from the model with a given bound to a CNF). SAT is only a part of the whole process, however the bottom line for formal verification is performance of this global process.

- For each model, several CNFs (from BMC translation with different bounds) are searched for satisfiability. All in all, SAT was run in many more CNFs than for the 2001 benchmark (several CNFs from the 2001 benchmarks are generated from the IBM Benchmark). Running SAT on a benchmark that was too small did not accurately reflect the conditions of day-to-day formal verification. This led us to anticipate overly optimistic conclusions.

3 A Simple but Efficient Distributed SAT BMC Algorithm

We conducted several experiments with the IBM Formal Verification Benchmark. In fact, it has become one of our main tools to assess new search engines and algorithms. The experiments we present here were conducted to offer a better assessment of different decision heuristics for zChaff and to assess a straightforward distributed algorithm for bounded model checking. The results provide a good perspective on the importance of the IBM Formal Verification Benchmark.

3.1 Experimental Settings

We ran RuleBase on the IBM benchmark with several configurations on 867841X Intel(R) Xeon(TM) CPU 2.40GHz with 512 KB first level cache, and 2.5 GB physical memory, blade workstations running Red Hat Linux release 7.3. When RuleBase was used with engines distributed on several workstations, they were interconnected with a 1 Gb Ethernet LAN. The different configurations are as follows:

- Sequential: RuleBase runs SAT bounded model checking on a single workstation. The bounds used sequentially include $k=0..10$, $k=11..15, \dots$, $k=46..50$. As soon as a satisfiable assignment is found, the search is over. If no satisfiable assignment is found for any of the bounds, the result is then unsat with bound 50.
- Concur 7: RuleBase distributes the tasks corresponding to the seven first bounds to seven workstations (i.e., BMC translation and SAT search for $k=0..10$, $k=11..15, \dots$, $k=36..40$). As soon as a satisfiable assignment is found, the whole search is over. If a task finishes without finding a satisfiable assignment, the next task is then assigned to the now idling workstation (until there are no tasks left, in which case the result is unsat 50).
- Concur 5: This uses the same principle as Concur 7, but distributes tasks in a five by five manner to five workstations.
- Concur 3: This uses the same principle as Concur 7, but distributes tasks in a three by three manner to three workstations.
- Concur 5, 5 heuristics: RuleBase runs five independent SAT BMC instances (similar to sequential). Each of the SAT BMC instances uses a different decision heuristic for zChaff, from VSIDS, SO, SB, DV, DVB.

- Concur 9 2/nodes: RuleBase distributes the nine SAT BMC tasks (corresponding to bounds $k=0\dots 10$, $k=11\dots 15,\dots$, $k=46\dots 50$) on five bi-processors workstations.

Table 3 presents the experimental results for the rules from the IBM Benchmark. We omitted the rules that ran in under two minutes with Sequential configuration and the rules that timed out (timeout was set at two and a half hours) for every configuration. For this experiment, we used bi-processor workstations with 867841X Intel(R) Xeon(TM) CPUs 2.40GHz with 512 KB first level cache, and 2.5 GB physical memory, running with Red Hat Linux release 7.3. Speedup NA means the search timed out for both configurations (i.e., Sequential and the given configuration).

Table 3. Experimental results with the IBM Formal Verification Benchmark. S stands for Sequential, C7 for Concur 7, C5 for Concur 5, C3 for Concur 3, H5 for Concur 5 with five heuristics, C 9/2 for Concur 9 with two processes per node

	result	S (hh:mm:ss)	C7 (speedup)	C5 (speedup)	H5 (speedup)	C9/2 (speedup)	C3 (speedup)
02_1 rule 1	unsat 50	00:06:00	2.0	1.9	1.5	2.0	1.6
02_1 rule 2	unsat 50	00:08:19	2.8	2.7	1.5	2.8	2.2
02_3 rule 2	unsat 50	00:09:02	2.7	2.6	1.4	2.7	2.5
02_3 rule 4	unsat 50	00:09:46	2.4	2.3	1.2	2.4	1.9
02_3 rule 6	unsat 50	00:09:25	2.3	2.2	1.7	2.4	1.8
02_3 rule 7	unsat 50	00:05:18	2.0	1.9	1.1	2.1	1.7
06	sat 31	00:02:25	1.8	1.7	1.1	1.8	1.6
10	unsat 50	00:29:47	2.7	2.6	1.9	2.8	2.0
11 rule 1	sat 31	00:25:26	8.9	8.2	1.3	8.8	7.4
14 rule 1	unsat 50	00:02:10	2.3	2.2	1.0	2.7	1.7
14 rule 2	unsat 50	00:25:43	2.9	2.7	1.0	1.9	2.2
17_1 rule 2	pass	00:05:13	2.5	2.4	1.0	2.1	1.9
18	sat 29	00:38:59	1.4	1.3	1.3	1.1	1.3
19	sat 29	00:02:23	1.4	1.4	1.1	1.2	1.2
20	sat 44	> 02:30:00	1.9	1.8	NA	1.8	1.3
22	unsat 50	> 02:30:00	NA	NA	1.6	NA	NA
23	sat 36	> 02:30:00	6.8	6.8	NA	6.3	5.9
26	unsat 50	00:15:25	3.4	3.1	1.0	3.8	2.3
29	sat 26	> 02:30:00	26.5	26.6	NA	21.8	9.7

3.2 Interpretation of Results

We noticed that Concur 9 2/node performs more poorly than Concur 5. This may seem quite surprising at first. However, we should keep in mind that BMC translation and SAT solving are very memory accesses consuming. Therefore, memory access can be a bottleneck when running two SAT instances on bi-processor workstations.

Heuristic tuning sometimes allows spectacular speedup for SAT solving (the only way we were able to achieve a result for 22). However, the overall improvement for SAT BMC, even when concurrently running several heuristics, is altogether marginal (though this approach could be mixed with the Concur approach).

Concur 7 and Concur 5 produce results that are similar, however, the Concur 3 results are significantly poorer. Therefore, the ideal configuration for our search (incremental bounded model checking with a maximal bound of 50) appears to be Concur 5.

Concur k (*i. e.* Concur 3, Concur 5, Concur 7) configurations may give more than linear³ speedup (e.g., 11 rule 1 and 29). The difficulty of a SAT BMC search does not necessarily grow with the bound. For some rules, it is far more difficult to prove there are no satisfiable assignment for $k+5$ than to find a satisfiable assignment for k , when searching concurrently. On the other hand, for some models, if k is the smallest bound for which a satisfiable assignment can be found, it will be easier to find a satisfiable agreement for $k+5$ than for k . Because k is the length of the shortest counter-examples to the model, it is likely that there will be more counter-examples of a longer length, eg $k+5$, and so more satisfiable assignments.

In summary, Concur k configurations have the biggest speedup potential for models that fail (in a number of cycles less than the maximal bound). Most such models from our benchmark fail in less than 30 cycles. This explains why Concur 7 displays little improvement when compared to Concur 5. In order to exhibit better performance for Concur k with k greater than or equal to 7, we would have to run the SAT BMC with a greater maximal bound and probably a broader benchmark (with models failing within a greater number of cycles). As can be observed in Table 4, the Concur 5 and Concur 3 results are quite good, especially considering that these two configurations distribute their tasks in a straightforward manner on five and three nodes, respectively.

Table 4. The results are computed without the “NA” cases

	Concur 7	Concur 5	Concur3	5 heuristics
Global speedup (Sequential total time/total time)	> 3.40	> 3.40	> 2.78	> 1.14
Average speedup (average Sequential time/time)	> 4.26	> 4.13	> 2.49	> 1.29

4 Conclusions

We explained how we used the IBM Formal Verification Benchmark to prove that previous results on SAT tuning were too optimistic and to assess the efficiency of a straightforward distribution for day-to-day SAT bounded model checking. We

³ With respect to the number of processors.

noted that a wider benchmark would allow even better and sounder assessments. As a result, we plan to make the IBM Formal Verification Benchmark library a living repository. We will add new design models in the future, to increase the benchmark diversity and keep it relevant in light of new technological advances.

Acknowledgment

The author wishes to thank Fabio Somenzi for his help on the IBM Formal Verification Benchmark translation to BLIF and, Cindy Eisner, Sharon Keidar, and Ofer Strichman for careful reviews and important comments of previous version of this paper.

A The IBM Formal Verification Benchmark

With the increased use of formal verification, benchmarking new verification algorithms and tools against real-life test-cases is now a must in order to assess performance gains. However, industrial designs are generally highly proprietary; therefore, models generated from these designs are usually not published. This makes difficult to assess the results reported in papers from the industry, as their benchmarks are usually not available and it is not possible to compare the published results with those achieved by other engines. Additionally, formal verification algorithms described in academic papers are often difficult to assess in terms of performance, since they are usually not applied to “real-life” benchmarks.

We want to stress how difficult it is to assess the real practical value of more sophisticated theoretical results without proper benchmarking. In the past, benchmarking enabled significant technology improvements, such as those for BDD packages in [18]. In the same way, many major improvements to boolean satisfiability solvers were proven useful by experimental results [19]. This may appear trivial, however there are many examples in the literature where elaborated and complex algorithms are not evaluated in a satisfactory manner. From the author’s experience, the claims of several papers could not be reproduced using the IBM Formal Verification Benchmark.

The IBM Formal Verification Benchmark library encompasses 37 declassified models, from 31 different hardware designs. The IBM Formal Verification benchmark library is available for academic users from the IBM Haifa verification projects web site [21].

The designs presented in the library are industrial designs that were verified by IBM teams. Each of the benchmark’s 37 files contains:

- A group of one or more temporal formulas, collectively called “rule”. To avoid language compatibility issues related to the specification language, the original PSL/Sugar [3, 1] formulas were translated into very simple $G(p)$ formulas (still written in PSL/Sugar), which most model checkers can readily address.

- A design model in PSL/Sugar environment description layer format. Some variables were renamed and some simple reductions were applied to hide the original design intent.

The models presented are of different sizes (Cf. Table 5) and varying degrees of complexity. However, as shown in Table 6 and Table 3, the same problem can sometimes be easily solved by one verification engine and at the same time with difficulty for another engine. For this reason, we tried to use a variety of problems. This benchmark is available in PSL/Sugar [1] and in Sugar1 format [3]. It was also translated to BLIF[20] format. The CNF output of BMC, applied to the benchmark for several bounds, is available from [22]. Some of these CNFs were used for the SAT2003 and SAT2004 contests[11, 26].

Table 5. IBM Formal Verification Benchmark Circuits Details

Name	Variables	Gates	Formulas	Name	Variables	Gates	Formulas
IBM 01	94	3266	1	IBM 17-1	1582	29190	2
IBM 02-1	139	1699	5	IBM 17-2	1581	28807	2
IBM 02-2	135	1671	1	IBM 18	78	4768	1
IBM 02-3	177	1983	7	IBM 19	120	5557	1
IBM 03	109	2656	1	IBM 20	78	4805	1
IBM 04	222	5067	1	IBM 21	78	4768	1
IBM 05	309	8410	1	IBM 22	103	6451	1
IBM 06	132	3375	1	IBM 23	102	6259	1
IBM 07	438	1341	1	IBM 24-1	49048	125896	3
IBM 08	395	84886	1	IBM 24-2	44807	115151	2
IBM 09	232	2000	1	IBM 25	120	4501	1
IBM 10	218	8702	6	IBM 26	1713	9640	1
IBM 11	222	8987	3	IBM 27	42	999	1
IBM 12	224	1055	1	IBM 28	95	3303	1
IBM 13	1506	17459	27	IBM 29	90	2562	1
IBM 14	156	3066	2	IBM 30	180	6654	1
IBM 15	231	4884	1	IBM 31-1	224	2488	3
IBM 16-1	1163	21750	1	IBM 31-2	224	2488	2
IBM 16-2	1162	21674	6				

We present some sample results⁴ achieved against this benchmark. To achieve these results, we used the Discovery engine, one of the RuleBase Classic [2] BDD engines. Each rule was run “from scratch” (i.e., without taking advantage of any pre-existing BDD orders). In real-life projects, the Discovery engine runs considerably faster when a good BDD order was found previously. This occurs because RuleBase can take advantage of rules previously run for this design, and get the best BDD order for a new rule. However, since the notion of “good order”

⁴ These sample results were achieved with the RuleBase version available through the RuleBase University Program [25].

Table 6. Experimental Results – time out: 3 hours

Name	Discovery	Name	Discovery
IBM 01	0:04:29	IBM 17-1	time out
IBM 02-1	0:02:04	IBM 17-2	time out
IBM 02-2	0:00:30	IBM 18	0:02:49
IBM 02-3	0:01:23	IBM 19	0:05:46
IBM 03	0:01:26	IBM 20	0:08:35
IBM 04	0:05:25	IBM 21	0:04:14
IBM 05	0:19:09	IBM 22	1:40:22
IBM 06	0:07:31	IBM 23	0:09:32
IBM 07	0:01:05	IBM 24-1	time out
IBM 08	0:23:38	IBM 24-2	time out
IBM 09	0:00:06	IBM 25	time out
IBM 10	2:48:16	IBM 26	time out
IBM 11	1:32:45	IBM 27	0:00:18
IBM 12	time out	IBM 28	1:34:52
IBM 13	0:03:44	IBM 29	0:15:18
IBM 14	time out	IBM 30	time out
IBM 15	3:08:01	IBM 31-1	time out
IBM 16-1	time out	IBM 31-2	time out
IBM 16-2	0:08:15		

is not very precise, and because an accurate description of such an order would comprise its entire listing, we only present results of runs without pre-existing orders.

In fact, RuleBase includes a set of engines of significantly higher performance than those referenced here.

We used an IBM Cascades PC with a Pentium III 700 MHz microprocessor running Red Hat Linux Advanced Server Release 2.1AS (Pensacola). The results are presented in Table 6 in an hh:mm:ss format.

References

1. Accelera. PSL/Sugar LRM. <http://www.eda.org/vfv/>
2. I. Beer *et al.* RuleBase: An Industry Oriented Formal Verification Tool. In *33rd Design Automation Conference, DAC 96. ACM/IEEE*, 1996.
3. I. Beer *et al.* The Temporal Logic Sugar. In *Computer Aided Verification, Proceedings of the 13th International Conference, CAV 2001. LNCS 2102*, July 2001.
4. A. Biere *et al.* Symbolic Model Checking Without BDDs. In *Proceedings of the workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS99)*, 1999.
5. A. Biere *et al.* Bounded Model Checking. In *Vol. 58 of Advances in Computers. Academic Press (pre-print)*, 2003.
6. S. Cook. The Complexity of Theorem Proving Procedures. In *Proceeding, Third Annual ACM Symp. on the Theory of Computing*, 1971.

7. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. In *Journal of the ACM*, 5(7), 1962.
8. N. Een, N. Sorensson. An Extensible SAT-solver, SAT 2003.
9. E. Emerson and C. Lei. Modalities for Model Checking: Branching Time Strikes Back. In *Science of Computer Programming*, 8:275-306, 1986.
10. E. Goldberg, Y. Novikov. BerkMin: a Fast and Robust SAT-solver In *Proceedings of the Design, Automation and Test in Europe. DATE'02.*, 2002.
11. D. Le Berre, L. Simon. The essentials of the SAT 2003 competition. In *Proceeding of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, LNCS 2919, pp 468-487, 2003.
12. M. Moskewicz *et al.* Chaff: Engineering an Efficient SAT Solver. In *38th Design Automation Conference*, page 530-535. *ACM/IEEE*, 2001.
13. A. Nadel. JeruSAT satisfiability solver. Available on-line. <http://www.geocities.com/alikn78/>
14. A. Pnueli. A Temporal Logic of Concurrent Programs. In *Theoretical Computer Science*, Vol 13, pp 45-60, 1981.
15. O. Shacham, E. Zarpas, Tuning the VSIDS Decision Heuristic for Bounded Model Checking. In *Proceeding of the 4th International Workshop on Microprocessor, Test and Verification*, IEEE Computer Society, Austin, Mai 2003.
16. J. Silva, K. Sakallah. Grasp - a New Search Algorithm for Satisfiability. In *Technical Report TR-CSE-292996*, University of Michigan, 1996.
17. O. Strichman. Tuning SAT Checkers for Bounded Model Checking. In *Computer-Aided Verification: 12th International Conference*, Lecture Notes in Computer Science, 1855. Springer-Verlag, 2000.
18. B. Yang *et al.* A Performance Study of BDD-Bases Model Checking. In *Formal Methods in Computer-Aided Design: 2nd International Conference*, Lecture Notes in Computer Science, 1522. Springer-Verlag, 1998.
19. L. Zhang and S. Malik The Quest for Efficient Boolean Satisfiability Solvers, in *Proceedings of 14th Conference on Computer Aided Verification (CAV2002)*, Copenhagen, Denmark, July 2002
20. Berkley Logic Interechange Format (BLIF). University of California, Berkley, 1992. <http://www-cad.eecs.berkeley.edu/Respep/Research/vis/usrDoc.html>
21. IBM Formal Verification Benchmark Library. http://www.haifa.il.ibm.com/projects/verification/RB_Homepage/fvbenchmarks.html
22. CNF Benchmarks from IBM Formal Verification Benchmarks Library. http://www.haifa.il.ibm.com/projects/verification/RB_Homepage/bmcbenchmarks.html
23. IBM CNF Benchmark Illustration. http://www.haifa.il.ibm.com/projects/verification/RB_Homepage/bmcbenchmarks_illustrations.html
24. Web version of the RuleBase User Manual. http://www.haifa.il.ibm.com/projects/verification/RB_Homepage/
25. RuleBase University Program. http://www.haifa.il.ibm.com/projects/verification/Formal_Methods-Home/university.html
26. SAT2004 contest. <http://satlive.org/SATCompetition/2004/>