Experimental Analysis of Different Techniques for Bounded Model Checking

Nina Amla[1], Robert Kurshan[1], Kenneth L. McMillan[1], and Ricardo Medel[2]

[1] Cadence Design Systems
[2] Stevens Institute of Technology

Abstract. *Bounded model checking* (BMC) is a procedure that searches for counterexamples to a given property through bounded executions of a non-terminating system. This paper compares the performance of SAT-based, BDD-based and explicit state based BMC on benchmarks drawn from commercial designs. Our experimental framework provides a uniform and comprehensive basis to evaluate each of these approaches. The experimental results in this paper suggest that for designs with *deep* counterexamples, BDD-based BMC is much faster. For designs with *shallow* counterexamples, we observe that indeed SAT-based BMC is more effective than BDD-based BMC, but we also observe that explicit state based BMC is comparably effective, a new observation.

1 Introduction

Model checking [CE81,QS82] is a formal technique for automatically verifying that a finite state model satisfies a temporal property. The states in the system may be represented explicitly as in [CE81]. Alternatively, Binary Decision Diagrams (BDDs) [Bry86] may be used to encode the transition relation. This approach is known as *symbolic model checking* [BCM$^+$90,McM93] and has been successfully applied in practice. However, it is computationally infeasible to apply this technique automatically to all systems since the problem is PSPACE-complete. *Bounded Model Checking* (BMC) [BCRZ99,BCCZ99] is a restricted form of model checking, where one searches for counterexamples in executions bounded by some length k. Recent advances [BCRZ99,BCCZ99] have encoded the bounded model checking problem as a propositional satisfiability problem that can then be solved by a SAT-solver. Initial results appear to be promising and show that the new generation of SAT-solvers (cf. [MSS99,MMZ$^+$01,GN02]) can handle large designs quite efficiently. SAT-based BMC can not provide any guarantees on the correctness of a property but it can be useful in finding counterexamples. It is possible to prove that a property holds with SAT-based BMC by computing the *completeness threshold* [KS02] and showing the absence of any errors at this bound. As observed in [KS02], since this threshold may be very large, it may not be possible to perform BMC at this bound.

This paper explores the performance of BMC with three reachability algorithms: SAT, BDDs and explicit state. We used the commercial model checking

H. Garavel and J. Hatcliff (Eds.): TACAS 2003, LNCS 2619, pp. 34–48, 2003.

tool COSPAN/FormalCheck, in which all three algorithms have been implemented. We implemented SAT-based BMC into COSPAN by using, as a post-processor, the Cadence SMV tool in conjunction with two SAT-solvers, BerkMin [GN02] and zChaff [MMZ+01]. This setup guarantees that the three types of BMC are applied uniformly to the same statically reduced model. We present experimental results on 62 benchmarks that were carefully chosen from a set of Cadence customer benchmarks based on a number of different criteria. We included benchmarks where BDDs performed well and others where BDDs performed poorly. Some of the benchmarks were beyond the scope of BDD-based model checking. Most of the benchmarks were customer hardware designs but we did include three software designs. The benchmarks are categorized according to the result and depth of the counterexample, and we include many examples with depth greater than 40. The properties were safety and liveness properties.

Several recent papers have compared BDD-based (bounded and unbounded) model checking to SAT-based bounded model checking. A recent comprehensive analysis, with respect to both the performance and capacity of BMC is presented in [CFF+01]. They compare a BDD-based tool (called Forecast) with a SAT-based tool (called Thunder) on 17 of Intel's internal benchmarks. Their results show an interesting tie between a tuned BDD-based Forecast and a default SAT-based Thunder, which suggest that, although the running times were similar, the time taken by experts to tune the BDD tool could be saved when the SAT-solver is used. However, since these were fundamentally distinct tools, there was no way to account for differences in front-end static reductions.

Our study differs from theirs in a number of key ways. First, we extended the analysis to include the explicit state representation. We found that a random search with the explicit state engine does about as well as SAT-based BMC in finding shallow counterexamples, that is, counterexamples at a depth of at most 50. Second, we focussed our study on larger depth limits. We observed that BDD-based BMC outperforms SAT-based BMC at larger depths. Last, our experimental framework uses the same static optimizations with all three engines. We believe this yields a more accurate comparison and diminishes the role that tuning the various tools played in the above work. For each of the three algorithms default settings were used, and there was no fine tuning for respective models. Thus, our results correspond better to what a commercial user might see.

Another interesting study [CCQ02] compares an optimized BDD-based BMC tool, called FBV, to SAT-based BMC with the NuSMV tool. Their results are similar to ours, in that, they find that the BDD approach scales better with increasing bounds. The key differences are that their analysis did not include the explicit state approach and they only considered safety properties. Moreover, we conducted our experiments with a larger and more varied set of benchmarks.

Over the last several years there has been considerable intent to compare the performance of unbounded BDD-based model checking versus SAT-based BMC. In [BCRZ99] they report that BMC with SAT-solvers, SATO [Zha97] and GRASP [MSS99], significantly outperformed the BDD-based CMU SMV

on 5 control circuits from a PowerPC microprocessor. Similar results were observed in [BCCZ99], where they found that SAT-based BMC with SATO and PROVE [Bor97] outperformed two versions of CMU SMV on benchmarks that were known to perform poorly with BDDs. SAT-based BMC, using SAT-solvers GRASP and CAPTAIN PROVE [SS98], was found to be better in [BLM01] than unbounded model checking with CMU SMV in the verification of the Alpha chip. The results reported in [Str00] showed that a tuned GRASP was able to outperform IBM's BDD-based model checker RuleBase in 10 out of 13 benchmarks. A new SAT-based method proposed in [BC00] was compared with unbounded BDD-based model checking with VIS [BHSV+96] on two benchmarks: an industrial telecommunications benchmark and an arbiter. They found that VIS did better on the arbiter while the SAT-based approach did better on the other benchmark. In [VB01], twenty-eight different SAT-solvers and one BDD-based tool were compared on a number of faulty versions of two microprocessor designs. The results show that zChaff outperforms the BDD-based tool and all the others SAT-solvers.

Our work differs from those mentioned above in a number of important ways. In addition to the differences already mentioned with regard to [CFF+01] above, a key difference is these authors' comparison of SAT-based BMC with unbounded BDD-based model checking, which we believe is not a good basis for comparing the two representations. In this paper, we show that BDD-based BMC has several advantages over SAT-based BMC. Our implementation of BDD-based BMC, unlike its SAT counterpart, can produce a positive answer if all the reachable states have been encountered at the depth checked. In addition, our experiments indicate that BDD-based BMC appears to be more successful at deeper depths. Our study includes both safety and liveness properties. The previous work either did not consider liveness properties or did not distinguish between safety and liveness properties.

The goal of this work was to provide a uniform and comprehensive basis for comparing BMC with three different representation: SAT, BDDs and explicit state. The trends observed in our study can be summarized as follows:

- SAT-based BMC is better than BDD-based BMC for finding shallow counterexamples.
- Random explicit state is as effective as SAT-based BMC in finding short counterexamples for safety properties but SAT-based BMC is better at finding the liveness counterexamples.
- Neither technique is better than BDDs in finding deep counterexamples.
- SAT-based BMC seems to be a consistent performer and completed in most cases.
- All three approaches seem fairly effective in proving the absence of counterexamples of length k. However, the BDD-based approach has two clear advantages. First, determining that a property holds is possible with BDDs, but not with SAT or random explicit state. Next, the BDD-based approach seems to scale better at larger depths than the two approaches. Both the explicit state and SAT engines perform rather well at the smaller depths but do not fare as well as the depth increases.

– The SAT-solver BerkMin appears to be better suited for BMC and outperforms zChaff quite significantly.

The paper is organized as follows. Section 2 describes our experimental framework, Section 3 presents our results and Section 4 summarizes our findings.

2 Experimental Framework

For our experiments we used the commercial model checking tool FormalCheck [HK90]. COSPAN, the verification engine of FormalCheck, was used for BMC with BDDs and with the explicit state approach. For SAT-based BMC, COSPAN was used to perform static reductions on the model and Cadence SMV [McM99], used as a post-processor, was used to do the SAT BMC. Cadence SMV has an interface to both BerkMin and zChaff. In this way the static reductions were applied in an uniform manner, for all three engines. These reductions include localization reduction [Kur94], constant propagation, equivalent code elimination, predicate simplification, resizing and macroization of combinational variables.

2.1 BDD-Based BMC

COSPAN's BDD-based BMC, for safety properties, is done by doing Reachability analysis for k steps. In the case of liveness properties, the bounded reachability analysis is done first and then a bad cycle detection check is done on the reachable states. Therefore, BDD-based BMC with a depth k terminates when one of the following conditions holds:

– all paths of length k have been explored,
– an error state (i.e. a counterexample) is reached, or
– all the reachable states have been explored (i.e. a fix-point has been reached).

The implementation uses a sifting-based dynamic re-ordering scheme. This BDD-based BMC implementation has two advantages over SAT-based BMC. First, it is possible to verify that a property holds if the fix-point is reached within the specified depth. Second, the length of the counterexample produced is independent of the depth checked and is guaranteed to be the shortest one. In SAT-based BMC, the counterexample produced will be of the same length as the depth checked.

2.2 Explicit State BMC

We used COSPAN for BMC with the explicit state engine. BMC was implemented through a command that "kills" all transitions after the depth limit k is reached. COSPAN allows a user to set the value of k through the command line. In order to deal with the large number of input values per state that are possible in commercial models, we used a random search of the state space for counterexamples, as supported by COSPAN's explicit state search engine. Thus, we

attempt to find counterexamples in some randomly chosen executions of length k. However, this process may miss executions and thus counterexamples.

We feel justified in comparing this random explicit state approach to the other two approaches since BMC is generally used to find counterexamples in contrast to proving that a property holds. The ability to use a random search is an advantage of the explicit state engine that we have exploited rather successfully.

2.3 SAT-Based BMC

For SAT-based BMC, we used Cadence SMV, as a post-processor to COSPAN, in conjunction with two state-of-the-art SAT-solvers: zChaff [MMZ$^+$01] and Berk-Min [GN02]. Cadence SMV implements many of the standard optimizations like bounded cone of influence and "BDD sweeping" [KK97] for tuning the performance of SAT-based BMC. We used a translator to convert the statically optimized programs (and properties) written in the input language of COSPAN (S/R) into the input format of SMV. The translator also translates counterexamples found by SMV back into the COSPAN format, and hence into the original HDL format via the FormalCheck interface. We found that the time to do the translations was not significant (usually a few seconds).

2.4 Benchmarks

The designs used are FormalCheck customer benchmarks that had some/all of the following characteristics.

- We chose benchmarks with deep counterexamples. The length of the counterexamples in the benchmarks varied from 3 up to 1152.
- We chose examples where BDDs performed poorly and others where BDDs performed well.
- The number of state variables in the designs varied from 11 up to 1273.
- We included software designs.
- We used both safety and liveness properties.

Informally, a *safety* property specifies that something "bad" never happens during an execution. The safety properties were of the following form: *Always x*, *Never x*, *After x Never y*, *After x Always y*, and *After x Always y Unless z*. *Liveness* properties state that something "good" eventually happens; the liveness properties we checked were of the following form: *Eventually x*, *After x Eventually y* and *After x Eventually y Unless z*. Most of the benchmarks contained only a single property to check but some were conjunctions of multiple properties.

We organized the 62 benchmarks into three groups. The first group (Group1) consisted of benchmarks where the property failed in at most 30 steps. The second group (Group2) had benchmarks that failed in more than 30 steps. The final group (Group3) had benchmarks where the property passed. The length of the counterexamples in the first two groups were already known (in most of the cases they were found using unbounded BDD-based model checking). All the experiments were run on Sun SPARC machines with 2 Gigabytes of memory.

3 Experimental Results

We ran each benchmark with FormalCheck default options with the BDD, SAT and random explicit state engines. Furthermore, we ran both the SAT-solvers, BerkMin and zChaff, on all the benchmarks.

In this Section, we analyze the results obtained. In the random explicit state approach, we ran a maximum of three random runs, increasing the number of inputs checked on each run from 50/state to 100/state and finally 500/state. If a counterexample was found we reported the time taken up to that point; otherwise we reported the cumulative time taken for all three runs.

Table 1. Results for benchmarks with counterexamples of length at most 30.

Benchmark				BDD	SAT		Exp. State	
name	type	depth	stvars	time	zChaff	Berkmin	time	result
A1	S	3	152	122.4	128.9	17.4	**0**	F
A2*	S	4	111	29.6	13.5	**13.3**	-	
A3	S	4	172	-	155.9	152.8	**4.8**	F
A4	L	5	92	61.8	**3.6**	**3.6**	-	
A5	S	7	1109	-	54.5	**46.4**	695.4	F
A6	L	7	171	-	**311.4**	736.1	-	
A7	S	7	62	4.5	4.1	1.6	**0.1**	F
A8	S	13	83	-	17.5	15.5	**4**	F
A9*	S	15	78	187.2	13909.7	807.4	**114.4**	F
A10	L	15	80	25.5	1.7	**1.4**	-	
A11	S	16	125	41.8	34.3	21.7	**2**	F
A12	S	16	58	1856.5	492.5	**259.7**	0.3	NC
A13	S	16	455	35.3	20.2	12.3	**1.2**	F
A14	S	20	132	16.1	29.4	15.1	**0.6**	F
A15	S	20	92	99.3	106.9	**7**	0.1	NC
A16*	S	21	23	79.7	1.5	**0.8**	-	
A17	S	21	115	-	3.5	**3**	97.7	F
A18	S	22	73	3477.5	3.5	**2.4**	-	
A19	L	23	93	197.4	**2.7**	3.5	-	
A20	S	23	102	34.5	34.4	**22.4**	1492.9	F

3.1 Benchmarks with Shallow Counterexamples

Group1 contained benchmarks that had properties that failed in at most 30 steps. Table 1 summarizes our results. The first column specifies the name of the benchmark, the second column is the type of property, the third is length of the counterexample and the fourth column is the number of state variables in the design. The next three columns give the time taken in seconds with BDDs and the SAT-solvers zChaff and BerkMin respectively. The last two columns give the

time taken for explicit state and the final result returned by the explicit state engine, "F" indicates a counterexample was found and "NC" indicates that a counterexample was not found in any of the three runs. We used a timeout of 30,000 seconds and is depicted as a "-" in the tables. Memory exhaustion is shown as "m/o". The symbol "*" next to the benchmark name indicates that it is a software design.

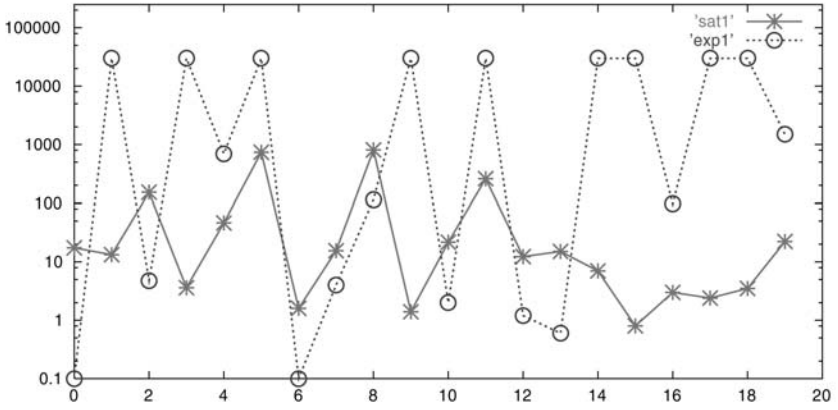

Fig. 1. Time taken for SAT BMC versus Random Explicit State on Group1 benchmarks. X-axis: benchmarks ordered by increasing depth, Y-axis: run time in seconds.

The most interesting observation in Group1 was that random explicit state, when it did finish, did extremely well, outperforming both the BDD and SAT engines on 8 out of the 16 safety benchmarks. The explicit state engine, however, did not find any counterexamples for the liveness properties. The SAT engine did better on 12 of the 20 benchmarks and did better than the BDD engine on all but one (`A9`). It also did better than the other two engines on all 4 liveness properties. The plot in Figure 1 shows that, while explicit state and SAT BMC are comparable, SAT-based BMC is more consistent.

3.2 Benchmarks with Deep Counterexamples

In Group2, where the length of the counterexamples varied from 34 up to 1152, we found the results to be quite different. Table 2 summarizes our results. The explicit state engine only completed successfully on 3 of the 17 benchmarks. The SAT engine did quite well up to a depth of 60 and outperformed the other engines on all 7 benchmarks. The BDD engine significantly outperformed the other two engines on the deeper counterexamples. This can be seen rather clearly in Figure 2 which shows that BDD-based BMC does better on all of the benchmarks that had counterexamples of length greater than 60, namely those numbered 8 and

Table 2. Results for benchmarks with counterexamples of length greater than 30.

Benchmark				BDD	SAT		Exp. State	
name	type	depth	stvars	time	zChaff	Berkmin	time	result
B1	S	34	184	1843.2	6.7	**3.6**	-	
B2	S	41	457	-	2418	**1760.4**	-	
B3*	S	41	43	-	13.3	**3.2**	-	
B4	L	52	1273	122.3	10.9	**8.2**	43.1	NC
B5	S	54	366	3699.4	422.5	**89.3**	-	
B6	S	54	195	44.1	4206.7	**37.7**	-	
B7	L	60	44	3026	58.5	**14.7**	-	
B8	S	66	11	0.1	0.1	0.1	**0.04**	F
B9	S	72	53	**12.7**	1973.4	45.5	-	
B10	S	82	46	**2.3**	1036.7	81.5	-	
B11	L	89	124	**34**	362.6	371.6	-	
B12	S	92	429	**337.9**	2988.9	27889.6	12473.8	F
B13	L	113	51	**84.1**	5946.6	1049.5	-	
B14	L	127	45	**1.4**	36	34.4	0.1	NC
B15	S	316	74	**15.6**	14159.2	229.9	150.7	F
B16	L	801	132	**75.4**	m/o	m/o	-	
B17	S	1152	153	**48.5**	2541.6	1035.9	-	

above. Overall, the BDD-based approach did better on 9 of the 17 benchmarks, the SAT approach did better on 7 and the explicit state approach did better on only 1 benchmark. Unlike BDD and explicit state BMC, SAT-based BMC did not complete on only one of the benchmarks (`B16`) in these two groups. We also found that BerkMin outperformed zChaff on most of the benchmarks and by a significant margin on the models with counterexamples of larger depths.

3.3 Benchmarks with Properties That Passed

Group3 contained benchmarks that had properties that passed. We ran each benchmark with a limit depth of 10, 25, 50 and 100 with all three engines. The time reported for the random explicit state engine is the cumulative time taken for three runs, systematically increasing the number of inputs considered in each successive run. Table 4 in the Appendix reports the results for BDDs, explicit state and SAT-based BMC with BerkMin. A comparison of the two SAT-solvers on these benchmarks that demonstrates that BerkMin does better, can be found in Table 3 in the Appendix. In Table 4, the first three columns correspond to the name of the benchmark, type of property and the number of state variables. The next three columns correspond to the time taken for BMC at a depth of 10 with explicit state, BDDs and SAT (using BerkMin) respectively. Similarly, the remaining columns report the results of BMC at depths 25, 50 and 100. As mentioned earlier, BDD-based BMC can assert that a property holds when it

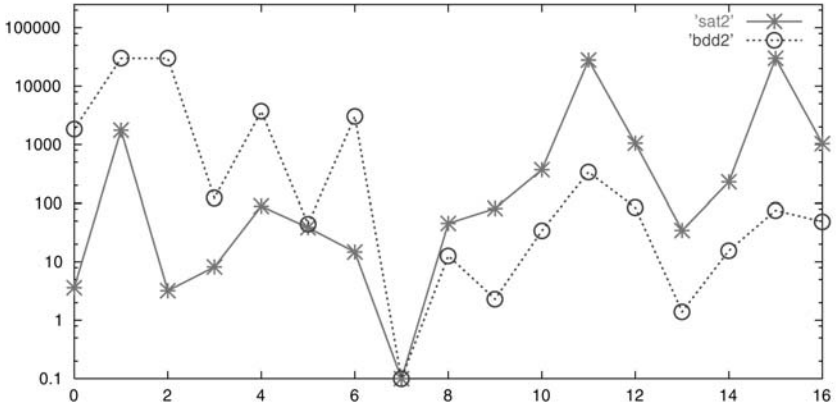

Fig. 2. Time taken for SAT BMC versus BDD BMC on Group2 benchmarks X-axis: benchmarks ordered by increasing depth. Y-axis: run time in seconds.

reaches a fix-point. This is depicted in the table by reporting the time taken at the depth where the fix-point was reached along with the suffix "P". We used a timeout of 36,000 seconds and this is shown in the table as "-".

We found that random explicit state BMC did fairly well at depth 10 and did better than the other two engines on 11 of the 25 benchmarks. However, the performance went down as the depth was increased. At a depth of 100, the explicit state engine timed out on all but 6 of the benchmarks but did better than the others on 5 of them. The BDD engine started out by doing better on only 3 of the 25 benchmarks at depth 10 but improved to do better on 13 benchmarks at final depth checked. Five of benchmarks (`C13`, `C17`, `C18`, `C22` and `C24`) in this group were known to do poorly with BDDs. In these cases, we found that SAT-based BMC did very well on three of them (`C18`, `C13` and `C17`) but did not do as well on the other two at larger depths. The explicit state engine did extremely well on one of them (`C17`). We reached a fix-point in 9 cases but BDDs outperformed the other approaches on only 4 of them. SAT-based BMC outperformed the other two engines on 11 of the 25 benchmarks at depth 10 but as the depth was increased it did worse, and at depth 100 it did better on only 6 of them. Again, the SAT-engine was the most consistent and only timed out at a depth of 50 or greater. Figure 3 plots the number of benchmarks that each technique did better on versus the four depths checked, namely 10, 25, 50 and 100. For example, the plot shows that at depth 10, the BDD-based approach did better on 3 benchmarks while the other two engines did better on 11 benchmarks. The BDD-based method, as shown in the plot in Figure 3, appears to scale better on benchmarks where it does not fail to build the global transition structure (those that fail are indicated in Table 4 by a "-"). An interesting point to note is that at least one of the three algorithms finished on all but one of the benchmarks.

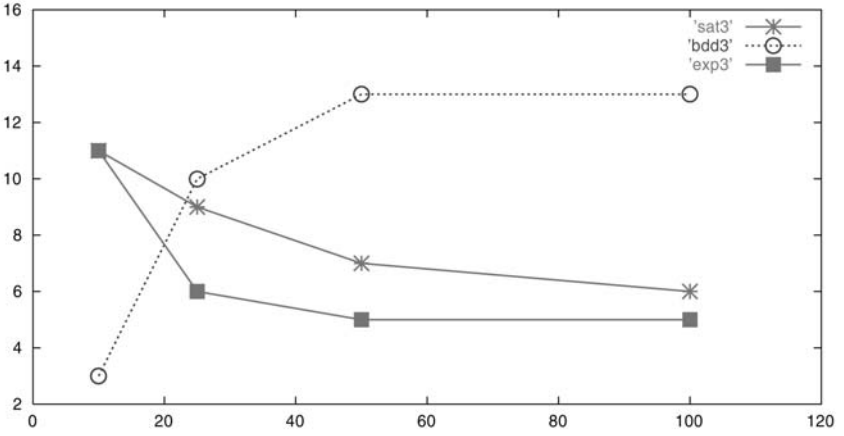

Fig. 3. Plot of the number of Group3 benchmarks that each engine did better on versus depth of BMC. X-axis: BMC depth at 10, 25, 50 and 100, Y-axis: Number of benchmarks (maximum is 25).

3.4 Choosing the Bound

In the results presented so far, we assumed that we knew the depth of the counterexample. This gave us a good measure of the performance of each engine. However, in general, the depth of the counterexample is unknown. There are two possible ways to choose the bound for BMC. One can choose a maximum bound and try to find the error within this bound. A disadvantage of this approach for SAT-based BMC is that the counterexample found will be of the specified length and this could make the counterexample harder to analyze. However, this approach seems to be the right one for both BDD-based and explicit state BMC since they stop as soon as they encounter the error. In order to investigate how this approach works with SAT-based BMC, and based on our results, we chose a maximum depth of 60 and ran the benchmarks in Groups 1 and 2 that had counterexamples of length at most 60. For each of these benchmarks, Figure 4 plots the time taken for BMC at the known depth of the counterexample versus the time taken at depth 60. We can see rather clearly that this approach is expensive and could take orders of magnitude more time. In fact three of the benchmarks that finished within 1000 seconds timed out at bound 60.

Alternatively we could employ an iterative approach, that starts at a minimum depth and systematically increases the depth until a counterexample is found. For our study, we used the following depths: 10, 25, 50 and 100. We applied this iterative method to benchmarks in Groups 1 and 2 that had counterexamples of length less than 100. Figure 5 plots the cumulative time taken versus the time taken at the depth of the counterexample. The X-axis represents the 29 benchmarks in increasing order based on the length of the counterexample and the Y-axis represents the cumulative time taken in seconds. This method appears to be more efficient and in most cases took approximately the same time

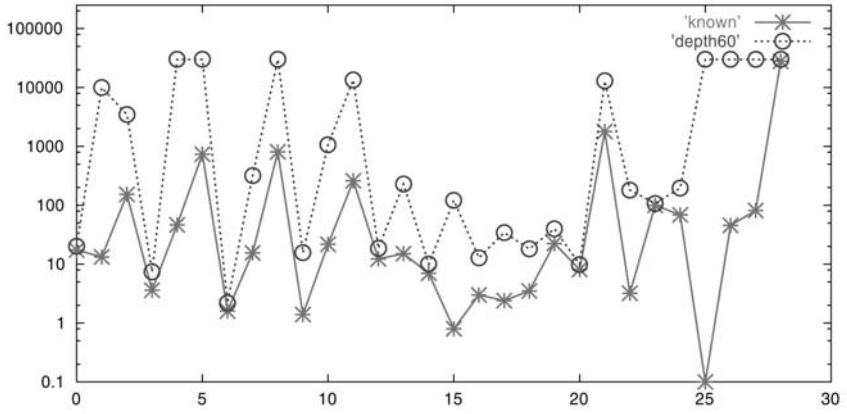

Fig. 4. Time taken for SAT BMC at the depth of counterexample versus depth=60. X-axis: benchmark, Y-axis: run time in seconds.

as BMC with the depth already known. The only three benchmarks where the difference was fairly significant had counterexamples of length 52 (`B3`) and 54 (`B5` and `B6` in Table 2) and therefore represent the worst case scenario. Only 2 of the 29 benchmarks with the iterative method changed the SAT BMC result in the comparison with BDDs and explicit state, and the difference in time in both cases was less than 30 seconds.

4 Conclusions

This paper presents a systematic performance analysis for BMC with three engines: SAT, BDDs and random explicit state. We used 62 industrial benchmarks that were partitioned into three groups based on the length of the counterexample and whether the property was falsified.

Our results demonstrate that for models with deep counterexamples, BDDs were the clear winner, while for models with shallow counterexamples, SAT and random explicit state performed comparably. The results were the same for both safety and liveness properties with the exception being that the random explicit state algorithm did much worse on the 13 liveness properties in our benchmark suite. The SAT-based approach was very consistent and completed within the timeout on all but 4 of the 62 benchmarks and, more importantly, all 4 timeouts were observed at a depth of 50 or greater. The SAT engine also seems to be less sensitive to the size of the design (number of state variables) and did well on the larger benchmarks. In cases when not much information is available about the design, running the engines in parallel until one of them completes is a sound idea.

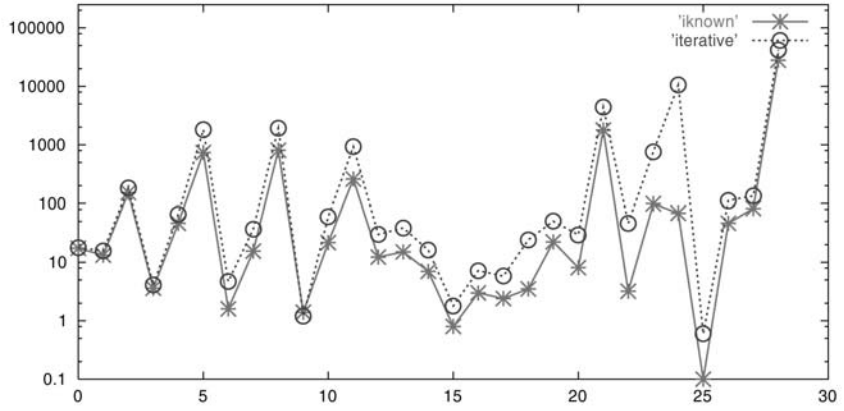

Fig. 5. Time taken for SAT BMC at the depth of counterexample versus cumulative time taken. X-axis: benchmark, Y-axis: run time in seconds.

References

[BC00] P. Bjesse and K. Claessen. SAT-based verification without state space traversal. In *FMCAD*, 2000.

[BCCZ99] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *LNCS*, 1999.

[BCM+90] J. R. Burch, E. M. Clarke, K. L. McMillan, D.L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *LICS*, 1990.

[BCRZ99] A. Biere, E. Clarke, R. Raimi, and Y. Zhu. Verifying safety properties of a PowerPC microprocessor using symbolic model checking without BDDs. In *CAV*, 1999.

[BHSV+96] R. Brayton, G. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. Ranjan, S. Sarwary, T . Shiple, G. Swamy, and T. Villa. VIS. In *FMCAD*, 1996.

[BLM01] P. Bjesse, T. Leonard, and A. Mokkedem. Finding bugs in an Alpha microprocessor using satisfiability solvers. In *CAV*, volume 2102 of *LNCS*, 2001.

[Bor97] A. Boralv. The industrial success of verification tools based on stalmarck's method. In *CAV*, 1997.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulations. *IEEE Transactions on Computers*, 1986.

[CCQ02] G. Cabodi, P. Camurati, and S. Quer. Can bdds compete with sat solvers on bounded model checking. In *DAC*, 2002.

[CE81] E.M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logics of Programs*, volume 131 of *LNCS*, 1981.

[CFF+01] Fady C., L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Y. Vardi. Benefits of bounded model checking at an industrial setting. In *CAV*, volume 2102 of *LNCS*, 2001.

[GN02] E. Goldberg and Y. Novikov. Berkmin: A fast and robust sat-solver. In *DATE*, 2002.

[HK90] Z. Har'El and R. P. Kurshan. Software for analytical development of communications protocols. *AT&T Technical Journal*, 69(1), 1990.

[KK97] A. Kuehlmann and F. Krohm. Equivalence checking using cuts and heaps. In *DAC*, 1997.

[KS02] D. Kroening and O. Strichman. Efficient computation of recurrence diameters. In *VMCAI*, 2002.

[Kur94] R. Kurshan. *Computer-aided Verification of Coordinating Processes: The Automata-Theoretic Approach*. Princeton University Press, 1994.

[McM93] K. McMillan. *Symbolic model checking: An approach to the state explosion problem*. Kluwer Academic Publishers, 1993.

[McM99] K. McMillan. Getting started with smv, 1999. URL: `http://www-cad.eecs.berkeley.edu/~kenmcmil`.

[MMZ+01] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.

[MSS99] Marques-Silva and Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEETC: IEEE Transactions on Computers*, 48, 1999.

[QS82] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. of the 5th International Symposium on Programming*, volume 137 of *LNCS*, 1982.

[SS98] M. Sheeran and G. Stålmarck. A tutorial on Stålmarck's proof procedure for propositional logic. In *CAV*, volume 1522 of *LNCS*, 1998.

[Str00] O. Strichman. Tuning SAT checkers for bounded model checking. In *CAV*, 2000.

[VB01] M. N. Velev and R. E. Bryant. Effective use of boolean satisfiability procedures in the formal verification of superscalar and VLIW. In *Proceedings of the 38th Conference on Design Automation Conference 2001*, 2001.

[Zha97] H. Zhang. SATO: An efficient propositional prover. In *Proceedings of the 14th International Conference on Automated deduction*, volume 1249 of *LNAI*, 1997.

5 Appendix

Table 3. Results for zChaff and BerkMin on benchmarks that passed.

Benchmark		depth 10		depth 25		depth 50		depth 100	
name	stvars	BerkMin	zChaff	BerkMin	zChaff	BerkMin	zChaff	BerkMin	zChaff
C1	51	**2.7**	**2.7**	**4.7**	5	**15.6**	50.1	**1002.1**	10529.8
C2	52	**3.3**	3.5	**3.8**	4.4	**5**	5.9	**7.4**	9.5
C3	53	3.3	**3.1**	**5.1**	15.1	**34.6**	219.2	**910.6**	7665.8
C4	58	**6.8**	46.5	**11.5**	145.7	**40**	1217.2	**414**	12284.2
C5	69	**1.6**	1.7	**2.6**	3.1	**5.8**	6.3	141.4	**131.9**
C6	70	**4.2**	4.4	**5.4**	11.6	**15.8**	67.7	**59.8**	495.2
C7	70	**6.8**	7.3	**7.1**	9.1	**9.7**	12.3	**14.8**	19.7
C8	77	13.3	**10**	17.7	**13.4**	25.3	**19.5**	40.4	**30.6**
C9	91	**10.3**	10.8	**14.4**	18.7	**22.6**	34.9	**96.8**	162.3
C10	95	10.7	**9.3**	**14.6**	14.8	**22.5**	24.1	57	**43.4**
C11	100	8.1	**5.3**	8.7	**7.6**	**15.2**	23.4	**1106.1**	6084.3
C12*	111	**35.4**	60.7	**4278.7**	23641.8	-	-	-	-
C13	114	**4.1**	5	**5.6**	7.2	**8.3**	11.5	**7.7**	20.3
C14	127	**70**	1723.4	**698**	-	**6960.9**	-	-	-
C15	131	**3.9**	**3.9**	6.3	**4**	**4**	6.2	6.5	**3.9**
C16	268	10	**8.8**	16.4	**16.3**	**34.3**	144.7	**460.9**	5026.3
C17	423	**15.6**	15.7	**23.5**	30.8	**38.6**	52.5	**71.4**	97.7
C18	423	14.6	**13.1**	21.2	**12.5**	**39**	2262.3	**98.9**	35340.3
C19	428	**54.7**	69.3	**78.2**	85.5	368.4	**133.3**	2356	**226.3**
C20	446	**22.1**	25.3	**27.9**	32.6	**37.7**	44.8	**60.4**	72.3
C21	455	22	**14.4**	33.8	**11**	52.2	**11**	100.5	**14.4**
C22	625	46.2	**42.7**	106.1	**58.3**	**3895.3**	12837.9	-	-
C23	600	**29**	29.8	**40.1**	45.9	**59.9**	72.4	**100.2**	125.7
C24	624	**43.9**	53.6	**61.7**	224.6	**69.6**	7977.9	**31753.6**	-
C25	644	**42.4**	46.8	**64.5**	80.8	152.6	**135**	**195**	255.4

Table 4. Results for benchmarks that passed.

Benchmark			depth 10			depth 25			depth 50			depth 100		
name	type	stvars	ES	BDD	BerkMin	ES	BDD	BerkMin	ES	BDD	BerkMin	ES	BDD	BerkMin
C1	S	51	118.9	**1.2**	2.7	2087.9	**0.9**	4.7	-	**1.5**	15.6	-	**51.7**	1002.1
C2	S	52	**0.09**	2.5	3.3	**0.08**	2.6	3.8	**0.13**	2.5	5	**0.12**	2.6(P)	7.4
C3	S	53	109.6	**1.3**	3.3	1588.2	**1.3**	5.1	-	**1.9**	34.6	-	**41.8**	910.6
C4	S	58	**2.6**	10.1	6.8	-	**10.1**	11.5	-	**10.1**	40	-	**10.3**	414
C5	S	69	**0.09**	1.1	1.6	**0.1**	1.4	2.6	**0.09**	2.5	5.8	**0.1**	2.7	141.4
C6	S	70	-	10.6	**4.2**	-	2164	**5.4**	-	9128.3	**15.8**	-	9586(P)	**59.8**
C7	S	70	-	7.7	**6.8**	-	111.2(P)	**7.1**	-	111.2(P)	**9.7**	-	111.2(P)	**14.8**
C8	S	77	**1.6**	15.5	13.3	**4.3**	7.8	17.7	8.9	**7.8**	25.3	17.9	**8.8**	40.4
C9	L	91	**0.14**	5.8	10.3	**0.15**	6.2	14.4	**0.14**	48.4	22.6	**0.14**	133.8(P)	96.8
C10	S	95	**2.3**	7	10.7	799.7	**7.1**	14.6	-	**9.5**	22.5	-	**10(P)**	57
C11	S	100	18.2	**3.2**	8.1	9326.3	**3.3**	8.7	-	**5.3**	15.2	-	**14.2**	1106.1
C12*	S	111	-	50	**35.4**	-	**370.4**	4278.7	-	**1872.4**	-	-	**4881.5**	-
C13	S	114	98.9	-	**4.1**	121.3	-	**5.6**	1568.5	-	**8.3**	-	-	**7.7**
C14	S	127	-	673.9	**70**	-	**664.4(P)**	698	-	**664.4(P)**	6960.9	-	**664.4(P)**	-
C15	S	131	-	49.1	**3.9**	-	7450.1	**6.3**	-	7869.5	**4**	-	8488.1	**6.5**
C16	S	268	**2.8**	6.2	10	182.5	**6.2**	16.4	9300.4	**6.4**	34.3	-	**12.6**	460.9
C17	S	423	**0.4**	12314(P)	15.6	**0.37**	12314(P)	23.5	**0.4**	12314(P)	38.6	**0.41**	12314(P)	71.4
C18	S	423	-	43.1	**14.6**	-	-	**21.2**	-	-	**39**	-	-	**98.9**
C19	L	428	**2.3**	34	54.7	57.8	**39.1**	78.2	3282.7	**96.9**	368.4	-	**297**	2356
C20	S	446	323.4	32.3	**22.1**	-	32.9	**27.9**	-	**33**	37.7	-	**34.9(P)**	60.4
C21	L	455	**7.1**	31.2	22	-	**31.3**	33.8	-	**33.6(P)**	52.2	-	**33.6(P)**	100.5
C22	S	457	-	-	**46.2**	-	-	**106.1**	-	-	**3895.3**	-	-	-
C23	S	600	364.6	53.4(P)	**29**	335.1	53.4(P)	**40.1**	357.4	**53.4(P)**	59.9	365	**53.4(P)**	100.2
C24	S	624	-	6705.3	**43.9**	-	-	**61.7**	-	-	**69.6**	-	-	**31753.6**
C25	S	644	**0.67**	32.6(P)	42.4	**0.6**	32.6(P)	64.5	**0.66**	32.6(P)	152.6	**0.67**	32.6(P)	195